9.3

Sviluppo di applicazioni per IBM MQ

Nota

Prima di utilizzare queste informazioni e il prodotto che supportano, leggere le informazioni in “Informazioni particolari” a pagina 1301.

Questa edizione si applica alla versione 9 release 3 di IBM® MQ e a tutte le successive release e modifiche se non diversamente indicato nelle nuove edizioni.

Quando si inviano informazioni a IBM, si concede a IBM un diritto non esclusivo di utilizzare o distribuire le informazioni in qualsiasi modo ritenga appropriato senza incorrere in alcun obbligo verso l'utente.

© Copyright International Business Machines Corporation 2007, 2024.

Indice

Sviluppo di applicazioni....5
Concetti dello sviluppo di applicazioni....7
Azioni che le applicazioni possono eseguire....7
Applicazioni, nomi applicazione e istanze applicazione....9
Programmi applicativi che utilizzano MQI....10
Utilizzo delle connessioni client per la connessione a più gestori code IBM MQ....11
Sviluppo di applicazioni client flessibili e scalabili....14
Applicazioni orientate agli oggetti....16
IBM MQ messaggi....18
Preparazione ed esecuzione di applicazioni di Microsoft Transaction Server....50
Considerazioni sulla progettazione per applicazioni IBM MQ....51
Specifica del nome dell'applicazione nei linguaggi di programmazione supportati....54
Tecniche di progettazione per i messaggi....59
Considerazioni sulla progettazione dell'applicazione e sulle prestazioni....61
Tecniche di progettazione per applicazioni avanzate....63
Considerazioni sulla progettazione e sulle prestazioni per le applicazioni IBM i....65
Considerazioni sulla progettazione per applicazioni Linux on Power Systems - Little Endian....66
Considerazioni sulla progettazione e sulle prestazioni per le applicazioni z/OS....67
Applicazioni bridge IMS e IMS su IBM MQ for z/OS....71
Sviluppo di applicazioni JMS/Jakarta Messaging e Java....83
Utilizzo di IBM MQ classes for JMS/Jakarta Messaging....84
Utilizzo di IBM MQ classes for Java....349
Utilizzo dell'adattatore di risorse IBM MQ....437
Utilizzo congiunto di IBM MQ e WebSphere Application Server....500
Utilizzo del pacchetto IBM MQ Headers....517
Configurazione di IBM MQ su IBM i con Java e JMS....520
Sviluppo di applicazioni Java utilizzando un repository Maven....527
Sviluppo di applicazioni C + +....528
Programmi di esempio C++....531
Considerazioni sul linguaggio C++....535
Messaggistica in C++....539
Creazione di programmi C++ IBM MQ....546
Sviluppo di applicazioni .NET....556
InstallazioneIBM MQ classes for .NET....557
InstallazioneIBM MQ classes for .NET Framework....563
Opzioni per la connessione di IBM MQ classes for .NET a un gestore code....563
Applicazioni di esempio per .NET....564
Configurazione del gestore code per l'accettazione delle connessioni client TCP/IP....567
Transazioni distribuite in .NET....567
Scrittura e distribuzione di programmi IBM MQ .NET....579
Sviluppo di applicazioni XMS .NET....616
Stili di messaggistica supportati da XMS....616
Il modello oggetto XMS....617
Il modello di messaggio XMS....620
InstallazioneIBM MQ classes for XMS .NET....620
Impostazione dell'ambiente del server di messaggistica....625
Utilizzo di applicazioni di esempio XMS....630
Scrittura di applicazioni XMS....633
Scrittura di applicazioni XMS .NET....652
Utilizzo degli oggetti amministrati XMS .NET....657
Come impedire alle applicazioni di utilizzare una versione XMS più recente....665
Protezione delle comunicazioni per applicazioni XMS....665

XMS messaggi.... 668
Sviluppo di applicazioni client AMQP.... 678
MQ Light, Apache Qpid JMSe AMQP (Advanced Message Queuing Protocol).... 680
supporto AMQP 1.0.... 681
Supporto point - to - point sui canali AMQP.... 683
Associazione di campi di messaggi AMQP e IBM MQ.... 684
Affidabilità della consegna dei messaggi.... 691
Topologie per i client AMQP con IBM MQ.... 696
IBM MQ Proprietà del controllo listener AMQP.... 702
Sviluppo di applicazioni REST con IBM MQ.... 703
Messaggistica mediante REST API.... 705
Sviluppo di applicazioni MQI con IBM MQ.... 717
File di definizione dati IBM MQ.... 718
Scrittura di un'applicazione procedurale per l'accodamento.... 721
Scrittura di applicazioni procedurali client.... 915
Uscite utente, uscite API e servizi installabili IBM MQ.... 938
Creazione di un'applicazione procedurale.... 1002
Gestione degli errori del programma procedurale.... 1040
Programmazione multicast.... 1046
Codifica in C.... 1052
Codifica in Visual Basic.... 1055
Codifica in COBOL.... 1056
Codifica in linguaggio assembler System/390 (Message queue interface).... 1056
Codifica di programmi IBM MQ in RPG (soloIBM i).... 1060
Codifica in PL/I (soloz/OS).... 1060
Utilizzo dei programmi procedurali di esempio IBM MQ.... 1060
Sviluppo di applicazioni per Managed File Transfer.... 1224
Specifica dei programmi da eseguire con MFT.... 1224
Utilizzo di Apache Ant con MFT.... 1227
Personalizzare MFT con uscite utente.... 1231
Controllo di MFT inserendo i messaggi nella coda comandi dell'agente.... 1245
Sviluppo di applicazioni per MQ Telemetry.... 1246
IBM MQ Telemetry Transport Programmi di esempio.... 1246
Concetti di programmazione client MQTT.... 1248
Sviluppo di applicazioni Microsoft Windows Communication Foundation con IBM MQ.... 1270
Introduzione al canale personalizzato IBM MQ per WCF con .NET.... 1271
Utilizzo di canali personalizzati IBM MQ per WCF.... 1275
Utilizzo degli esempi WCF.... 1294

Informazioni particolari.... 1301
Informazioni sull'interfaccia di programmazione.... 1302
Marchi.... 1302

Sviluppo di applicazioni per IBM MQ

È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

Nuovo per lo sviluppo di applicazioni per IBM MQ?

Per ulteriori informazioni sullo sviluppo di applicazioni per IBM MQ, visitare IBM Developer:

- IBM MQ Developer Essentials *(scopri le nozioni di base, esegui una demo, codifica di un'applicazione, utilizza esercitazioni più avanzate)*
- IBM MQ Downloads for Developers *(incluse edizioni per sviluppatori gratuite e versioni di prova)*

Inoltre, potrebbe essere più semplice sviluppare le applicazioni se si conoscono i concetti descritti nelle seguenti sezioni:

- “Concetti dello sviluppo di applicazioni” a pagina 7
- “Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51

Supporto per framework e linguaggi orientati agli oggetti

IBM MQ fornisce il supporto principale per le applicazioni sviluppate nei seguenti linguaggi e framework:

- JMS
- Java
- C++
- .NET

Consultare anche “Applicazioni orientate agli oggetti” a pagina 16.

.NET supporta le applicazioni sviluppate in molte lingue. Per illustrare l'utilizzo delle classi IBM MQ per .NET per accedere alle code di IBM MQ , la documentazione del prodotto MQ contiene informazioni per le seguenti lingue:

- C# codice di esempio e applicazioni di esempio
- Applicazioni di esempio C++
- Visual Basic applicazioni di esempio

Consultare “Scrittura e distribuzione di programmi IBM MQ .NET” a pagina 579.

IBM MQ supporta .NET Core per applicazioni in ambienti Windows da IBM MQ 9.1.1 e per applicazioni in ambienti Linux® da IBM MQ 9.1.2. Per ulteriori informazioni, consultare “InstallazioneIBM MQ classes for .NET” a pagina 557.

IBM MQ supporta anche i client AMQP che implementano il protocollo 1.0 di OASIS AMQP.

MQ Light, Apache I client Qpid come Apache Qpid Proton e le API Apache Qpid JMS si basano sul protocollo.

Le API MQ Light sono disponibili all'indirizzo IBM MQ Light.

I client Apache Qpid sono disponibili in QPid Proton.

I seguenti bind di lingua sono forniti così come sono:

- un Bind Go
- un'implementazione API JavaScript che funziona con applicazioni Node.js

© Copyright IBM Corp. 2007, 2024

Supporto per API REST programmatiche

IBM MQ fornisce il supporto per le seguenti API REST programmatiche per inviare e ricevere messaggi:

- IBM MQ messaging REST API
- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM DataPower Gateway

Vedi “Sviluppo di applicazioni REST con IBM MQ” a pagina 703e anche l'esercitazione Introduzione all'API REST di messaggistica IBM MQ nell'area IBM MQ di IBM Developer. Questa esercitazione include esempi nelle seguenti lingue, forniti così come sono, da utilizzare con IBM MQ messaging REST API:

- Esempio Go che utilizza l'API REST di messaggistica MQ
- Esempio Node.js utilizzando il modulo HTTPS
- Esempio Node.js con modulo Promise

Supporto per i linguaggi di programmazione procedurali

IBM MQ fornisce supporto per le applicazioni sviluppate nei seguenti linguaggi di programmazione procedurali:

- C
- Windows Visual Basic (solo sistemiWindows)
- COBOL
- z/OS Assembler (soloIBM MQ for z/OS)
- z/OS PL/I (soloIBM MQ for z/OS)
- IBM i RPG (soloIBM MQ for IBM i)

Queste lingue utilizzano l'interfaccia MQI (Message Queue Interface) per accedere ai servizi di accodamento messaggi. Consultare “Sviluppo di applicazioni MQI con IBM MQ” a pagina 717. Notare che IBM MQ Object Model, utilizzato dai framework e dai linguaggi orientati agli oggetti, fornisce ulteriori funzioni che non sono disponibili per i linguaggi procedurali che utilizzano MQI.

Specifica del nome dell'applicazione

ALW

Prima di IBM MQ 9.1.2, era possibile specificare un nome applicazione su applicazioni client Java o JMS . Da IBM MQ 9.1.2, è inoltre possibile specificare il nome dell'applicazione su ulteriori linguaggi di programmazione. Per ulteriori informazioni, consultare “Specifica del nome dell'applicazione nei linguaggi di programmazione supportati” a pagina 54.

Attività correlate

“Sviluppo di applicazioni per MQ Telemetry” a pagina 1246

“Sviluppo di applicazioni Microsoft Windows Communication Foundation con IBM MQ” a pagina 1270
Il canale personalizzato WCF (Microsoft Windows Communication Foundation) per IBM MQ invia e riceve messaggi tra i servizi e i client WCF.

Riferimenti correlati

“Sviluppo di applicazioni per Managed File Transfer” a pagina 1224
Specificare i programmi da eseguire con Managed File Transfer, utilizzare Apache Ant con Managed File Transfer, personalizzare Managed File Transfer con uscite utente e controllare Managed File Transfer inserendo i messaggi nella coda comandi dell'agent.

Concetti dello sviluppo di applicazioni

È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

Per informazioni sui tipi di applicazione che è possibile scrivere per IBM MQ, consultare “Sviluppo di applicazioni per IBM MQ” a pagina 5 e “Azioni che le applicazioni possono eseguire” a pagina 7.

Concetti correlati

“Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51
Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

Azioni che le applicazioni possono eseguire

È possibile sviluppare applicazioni per inviare e ricevere messaggi necessari per supportare i processi di business. È anche possibile sviluppare applicazioni per gestire i gestori code e le risorse correlate.

Azioni che le applicazioni possono eseguire su IBM MQ for Multiplatforms

Su Multiplatforms, è possibile scrivere applicazioni che eseguono le azioni riportate di seguito:

- Inviare messaggi ad altre applicazioni in esecuzione negli stessi sistemi operativi. Le applicazioni possono essere sullo stesso o su un altro sistema.
- Inviare messaggi alle applicazioni che vengono eseguite su altre piattaforme IBM MQ .
- Utilizzare l'accodamento messaggi da CICS per i sistemi seguenti:
  -

TXSeries per AIX
  -

IBM i
  -

Windows
- Utilizzare l'accodamento dei messaggi dall'interno di Encina per i seguenti sistemi:
  -

AIX
  -

Windows
- Utilizzare l'accodamento dei messaggi da Tuxedo per i seguenti sistemi:
  -

AIX
  - AT&T
  -

Windows
- Utilizzare IBM MQ come gestore transazioni, coordinando gli aggiornamenti effettuati dai gestori risorse esterni nelle unità di lavoro IBM MQ . I seguenti gestori risorse esterni sono supportati e compatibili con l'interfaccia XA X/OPEN
  - Db2
  - Informix
  - Oracle
  - Sybase
- Elaborare diversi messaggi insieme come una singola unità di lavoro di cui è possibile eseguire il commit o il backout.
- Eseguire da un ambiente IBM MQ completo o da un ambiente client IBM MQ .

Azioni che le applicazioni possono eseguire su IBM MQ for z/OS

z/OS

Su z/OS, è possibile scrivere applicazioni che eseguono le azioni riportate di seguito:

- Utilizzare l'accodamento dei messaggi all'interno di CICS o IMS.
- Inviare messaggi tra applicazioni batch, CICSe IMS , selezionando l'ambiente più appropriato per ciascuna funzione.
- Inviare messaggi alle applicazioni che vengono eseguite su altre piattaforme IBM MQ .
- Elaborare diversi messaggi insieme come una singola unità di lavoro di cui è possibile eseguire il commit o il backout.
- Inviare messaggi e interagire con le applicazioni IMS tramite il bridge IMS .
- Partecipa a unità di lavoro coordinate da RRS.

Ogni ambiente all'interno di z/OS ha le sue caratteristiche, vantaggi e svantaggi. Il vantaggio di IBM MQ for z/OS è che le applicazioni non sono collegate a un ambiente, ma possono essere distribuite per sfruttare i vantaggi di ciascun ambiente. Ad esempio, è possibile sviluppare interfacce dell'utente finale utilizzando TSO o CICS, è possibile eseguire moduli ad alta intensità di elaborazione in batch z/OS ed è possibile eseguire applicazioni di database in IMS o CICS. In ogni caso, le varie parti dell'applicazione possono comunicare utilizzando messaggi e code.

I progettisti di applicazioni IBM MQ devono essere consapevoli delle differenze e delle limitazioni imposte da questi ambienti. Ad esempio:

- IBM MQ fornisce funzioni che consentono la comunicazione tra gestori code (nota come *accodamento distribuito*).
- I metodi di commit e di backout delle modifiche differiscono tra gli ambienti batch e CICS .
- IBM MQ for z/OS fornisce supporto nell'ambiente IMS per i programmi di elaborazione dei messaggi in linea (MPP), i programmi IFP (interattivi fast path) e i BMP (batch message processing program). Se si stanno scrivendo programmi batch DL/I, seguire le istruzioni fornite in argomenti come “Creazione di applicazioni batch z/OS” a pagina 1026 e “Considerazioni sul batch z/OS” a pagina 732 per i programmi batch z/OS .
- Sebbene più istanze di IBM MQ for z/OS possano esistere su un singolo sistema z/OS , una regione CICS può connettersi a un solo gestore code alla volta. Tuttavia, è possibile connettere più di una regione CICS allo stesso gestore code. Negli ambienti batch IMS e z/OS , i programmi possono connettersi a più di un gestore code.
- IBM MQ for z/OS consente alle code locali di essere condivise da un gruppo di gestori code, migliorando la velocità di trasmissione e la disponibilità. Tali code vengono denominate *code condivise* e i gestori code formano un *gruppo di condivisione code*, che può elaborare i messaggi sulle stesse code condivise. Le applicazioni batch possono connettersi a uno dei diversi gestori code all'interno di un gruppo di condivisione code specificando il nome del gruppo di condivisione code, invece di un determinato nome gestore code. Ciò è noto come *collegamento batch di gruppo* o più semplicemente *collegamento di gruppo*. Vedere Code condivise e gruppi di condivisione code.

z/OS Le differenze tra gli ambienti supportati e le relative limitazioni sono spiegate ulteriormente in “Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890.

Concetti correlati

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

“Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51
Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

“Scrittura di un'applicazione procedurale per l'accodamento” a pagina 721

Utilizzare queste informazioni per informazioni sulla scrittura delle applicazioni di accodamento, la connessione e la disconnessione da un gestore code, la pubblicazione / sottoscrizione e l'apertura e la chiusura di oggetti.

“Scrittura di applicazioni procedurali client” a pagina 915
Cosa devi sapere per scrivere le applicazioni client su IBM MQ utilizzando un linguaggio procedurale.

“Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

“Utilizzo di IBM MQ classes for Java” a pagina 349
Utilizzare IBM MQ in un ambiente Java . IBM MQ classes for Java consenti a un'applicazione Java di connettersi a IBM MQ come client IBM MQ o di connettersi direttamente a un gestore code IBM MQ .

“Sviluppo di applicazioni .NET” a pagina 556
IBM MQ classes for .NET consentire alle applicazioni .NET di connettersi a IBM MQ come IBM MQ MQI client o di connettersi direttamente a un server IBM MQ .

“Sviluppo di applicazioni C + +” a pagina 528
IBM MQ fornisce classi C+ + equivalenti a oggetti IBM MQ e alcune classi aggiuntive equivalenti a tipi di dati array. Fornisce un certo numero di funzioni non disponibili tramite MQI.

“Creazione di un'applicazione procedurale” a pagina 1002
È possibile scrivere un'applicazione IBM MQ in uno dei diversi linguaggi procedurali ed eseguire l'applicazione su diverse piattaforme.

Attività correlate

“Utilizzo dei programmi procedurali di esempio IBM MQ” a pagina 1060
Questi programmi di esempio sono scritti in linguaggi procedurali e dimostrano gli usi tipici di MQI (Message Queue Interface). Programmi IBM MQ su diverse piattaforme.

“Sviluppo di applicazioni Microsoft Windows Communication Foundation con IBM MQ” a pagina 1270
Il canale personalizzato WCF (Microsoft Windows Communication Foundation) per IBM MQ invia e riceve messaggi tra i servizi e i client WCF.

Protezione

Multi Applicazioni, nomi applicazione e istanze applicazione

Prima di iniziare a progettare e scrivere le applicazioni, acquisire familiarità con i concetti di base di applicazioni, nomi di applicazioni e istanze di applicazioni.

Applicazioni

Multi

Le connessioni a un gestore code vengono considerate dalla stessa *applicazione* se forniscono lo stesso *nome applicazione*. Il nome dell'applicazione viene visualizzato come attributo APPLTAG del comando DISPLAY CONN (*) TYPE CONN.

Note:

1. Per le applicazioni che utilizzano una versione di IBM MQ client precedente al IBM MQ 9.1.2, il nome dell'applicazione viene automaticamente impostato da IBM MQ client. Il suo valore dipende dal linguaggio di programmazione dell'applicazione e dalla piattaforma su cui è in esecuzione l'applicazione. Per ulteriori informazioni, consultare PutApplName .
2. Per le applicazioni IBM MQ client che utilizzano un IBM MQ client all'indirizzo IBM MQ 9.1.2 o successive, è possibile impostare il nome dell'applicazione su un valore specifico. Nella maggior parte dei casi, ciò non richiede modifiche al codice dell'applicazione o la necessità di ricompilare l'applicazione. Consultare “Utilizzo del nome applicazione nei linguaggi di programmazione supportati” a pagina 55 per ulteriori informazioni.

Istanze dell'applicazione

Multi

Le connessioni sono ulteriormente suddivise in *istanze dell'applicazione*. Un'istanza di un'applicazione è un insieme di connessioni strettamente correlate che forniscono una 'unità di esecuzione' per tale applicazione. In genere, questo è un singolo processo del sistema operativo, che può avere un numero di thread e connessioni IBM MQ associate.

In IBM MQ for Multiplatforms un'istanza dell'applicazione è associata a una Tag di connessionespecifica. Il gestore code associa automaticamente nuove connessioni a un'istanza dell'applicazione esistente, quando può vedere che sono correlate.

Note:

- Se si utilizzano le connessioni client, questi processi potrebbero connettersi al gestore code su uno o più canali in esecuzione.
- Nelle applicazioni JMS , un'istanza dell'applicazione viene associata a una connessione JMS specifica e a tutte le sessioni JMS associate.

Le istanze dell'applicazione sono particolarmente importanti su IBM MQ for Multiplatforms quando si utilizza il bilanciamento automatico dell'applicazionedel cluster uniforme. Su piattaforme IBM MQ for Multiplatforms , è possibile visualizzare le istanze dell'applicazione attualmente connesse utilizzando il comando DISPLAY APSTATUS .

In alcuni casi, il gestore code non può eseguire correttamente la connessione all'associazione dell'istanza dell'applicazione, in particolare:

- Se vengono effettuate più connessioni su una conversazione condivisa dallo stesso processo, utilizzando nomi di applicazione differenti.
- Se le librerie client di livello precedente sono in uso. Ad esempio, installazioni client IBM MQ JMS in IBM MQ 9.1.2 e versioni precedenti.

In queste situazioni, se le applicazioni non si definiscono come ricollegabili, ciò sarà consentito, ma alcuni dei raggruppamenti di istanze dell'applicazione potrebbero non essere corretti. Se una qualsiasi delle connessioni viene dichiarata come MQCNO_RECONNECT, ciò influisce significativamente negativamente sul bilanciamento dell'applicazione; la chiamata MQCONN, quindi, verrà rifiutata con MQCNO_RECONNECT_INCOMPATIBILI.

Concetti correlati

“Specifica del nome dell'applicazione nei linguaggi di programmazione supportati” a pagina 54
Prima di IBM MQ 9.2.0, era già possibile specificare un nome applicazione su applicazioni client Java o JMS . Da IBM MQ 9.2.0 questa funzione è estesa ad altri linguaggi di programmazione su IBM MQ for Multiplatforms.

Programmi applicativi che utilizzano MQI

I programmi applicativi IBM MQ necessitano di determinati oggetti prima che possano essere eseguiti correttamente.

Figura 1 a pagina 11 mostra un'applicazione che rimuove i messaggi da una coda, li elabora e invia alcuni risultati a un'altra coda sullo stesso gestore code.

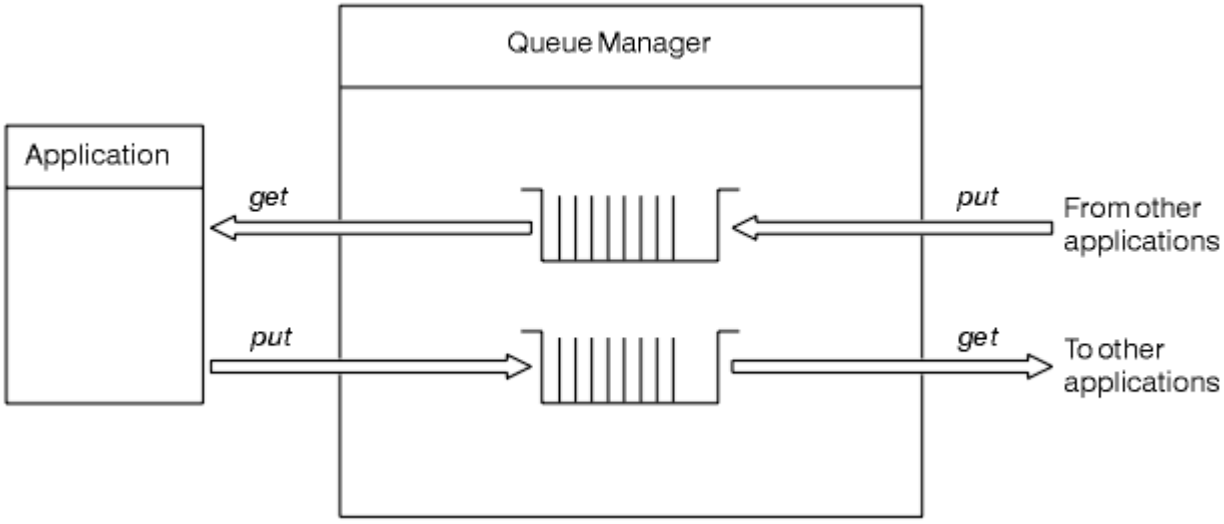


Figura 1. Code, messaggi e applicazioni

Mentre le applicazioni possono inserire i messaggi nelle code locali o remote (utilizzando MQPUT), possono ottenere i messaggi solo direttamente dalle code locali (utilizzando MQGET).

Prima di poter eseguire questa applicazione, è necessario soddisfare le seguenti condizioni:

- Il gestore code deve esistere ed essere in esecuzione.
- È necessario definire la prima coda dell'applicazione da cui devono essere rimossi i messaggi.
- È necessario definire anche la seconda coda, in cui l'applicazione inserisce i messaggi.
- L'applicazione deve essere in grado di connettersi al gestore code. Per fare ciò deve essere collegato a IBM MQ. Consultare “Creazione di un'applicazione procedurale” a pagina 1002.
- Anche le applicazioni che inserano i messaggi nella prima coda devono connettersi a un gestore code. Se sono remoti, devono anche essere impostati con code di trasmissione e canali. Questa parte del sistema non viene visualizzata in Figura 1 a pagina 11.

Utilizzo delle connessioni client per la connessione a più gestori code IBM MQ

È possibile configurare le applicazioni connesse al client per connettersi a più di un gestore code (per motivi di bilanciamento del carico o di disponibilità del servizio).

I meccanismi principali per ottenere ciò nel client IBM MQ sono l'uso delle tabelle di definizione del canale client, consultare Configurazione delle tabelle di definizione del canale cliento gli elenchi di connessioni.

È anche possibile ottenere un comportamento simile utilizzando prodotti di bilanciamento del carico esterni o impacchettando il codice di connessione IBM MQ in uno 'stub' che può reindirizzare i nomi host o gli indirizzi IP.

Ognuna di queste tecniche viene fornita con alcune restrizioni e potrebbe essere più o meno adatta a particolari requisiti di applicazione. Le seguenti sezioni, anche se non esaustive, descrivono aspetti particolari da considerare e l'effetto di questi diversi approcci su questi aspetti.

IBM MQ cluster uniformi, vedere Informazioni sui cluster uniformi, fornisce un potente meccanismo per ottenere la scalabilità orizzontale delle applicazioni su più gestori code, basandosi sul meccanismo di base della CCDT per fornire più destinazioni. I cluster uniformi possono fornire funzionalità oltre a quelle possibili utilizzando un programma di bilanciamento del carico esterno non consapevole dei protocolli IBM MQ sottostanti ed evitare alcuni dei problemi illustrati di seguito, pertanto, considera l'utilizzo di un cluster uniforme piuttosto che di altre tecniche, laddove applicabile.

Attenzione: Si consiglia di utilizzare con attenzione le applicazioni che utilizzano IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, inclusi quelli che utilizzano uno degli adattatori di risorse IBM MQ , che si collegano ai gestori code utilizzando le tecnologie di bilanciamento del carico. Se si verificano dei problemi, ricrearli senza tentare di utilizzare il bilanciamento del carico.

Ci sono diversi problemi, tutti che fanno sì che tali connessioni siano nel migliore dei casi problematiche e nel peggiore del tutto inaffidabili:

- Particolare attenzione è richiesta quando si connette un'applicazione che effettua più connessioni al gestore code utilizzando qualsiasi forma di bilanciamento del carico. Ciò include tutte le applicazioni che utilizzano le IBM MQ Classi per JMS/Jakarta Messaging poiché creano più connessioni IBM MQ nell'utilizzo generale. Se si utilizza un programma di bilanciamento del carico esterno o uno stub del codice personalizzato, è necessario instradare le connessioni dalla stessa istanza dell'applicazione allo stesso gestore code in qualsiasi momento.
- L'utilizzo della gestione delle transazioni XA o di JTA (Java Transaction API) si basa sulla possibilità di connettersi in modo congruente allo stesso gestore code - in pratica è improbabile che ciò sia pratico con qualsiasi forma di bilanciamento del carico.
- - La gestione del cluster uniforme si basa sulla capacità di indicare ai client di riconnettersi a specifici gestori code senza interferenze. Non è consigliabile tentare di combinare il bilanciamento del carico esterno con l'uso di cluster uniformi

È necessario utilizzare la funzione cluster uniforme IBM MQ per ottenere il ridimensionamento orizzontale delle applicazioni su più gestori code, piuttosto che le tecnologie di bilanciamento del carico esterne. Consultare Configurazione di un cluster uniformee i seguenti argomenti, per informazioni sui cluster uniformi, incluso il modo in cui vengono creati e utilizzati i cluster uniformi.

Termini utilizzati in queste informazioni

CCDT - multi - QMGR
Indica un file CCDT che contiene più canali di connessione client (CLNTCONN) con lo stesso gruppo, ovvero l'attributo QMNAME CLNTCONN (connessione client nome gestore code), in cui voci CLNTCONN differenti si risolvono in gestori code differenti.

Questo è diverso da un file CCDT che contiene più voci CLNTCONN che sono semplicemente nomi host o indirizzi IP differenti per lo stesso gestore code a più istanze, che è un approccio che è possibile scegliere di combinare con uno stub di codice.

Se si sceglie un approccio a più gestori code CCDT, è necessario scegliere se assegnare una priorità alle voci o disporre di WLM (work load management) randomizzato:

Minacce
Utilizzare più voci ordinate alfabeticamente con gli attributi CLNTWGHT (1) e AFFINITY (PREFERRED) per ricordare l'ultima buona connessione.

Randomizzato
Utilizzare gli attributi CLNTWGHT (1) e AFFINITY (NONE). È possibile regolare il peso WLM su server IBM MQ con scalabilità differente regolando CLNTWGHT

Nota: Si dovrebbero evitare grandi differenze di CLNTWGHT tra i canali.

Bilanciamento del carico
Indica un dispositivo di rete con un indirizzo IP virtuale (VIP) configurato con il controllo della porta dei listener TCP/IP di più gestori code IBM MQ . La configurazione del VIP nel dispositivo di rete dipende dal dispositivo di rete che si sta utilizzando.

Le seguenti scelte sono relative solo alle applicazioni che inviano messaggi o che avviano la messaggistica di richiesta e risposta sincrona. Le considerazioni per le applicazioni che gestiscono tali messaggi e richieste, ad esempio, i listener sono completamente separati e discussi in dettaglio in "Connessione di un listener di messaggi a una coda".

Scalabilità della modifica del codice richiesta per le applicazioni esistenti che si connettono a un singolo gestore code

Elenco CONNAME, CCDT multi - QMGR e programma di bilanciamento del carico
MQCONN ("QMNAME") a MQCONN ("*QMNAME")

Il nome del gestore code potrebbe trovarsi nella configurazione JNDI (Java Naming and Directory Interface) per le applicazioni Java Platform, Enterprise Edition (Java EE). In caso contrario, è necessario modificare il codice di un carattere.

Stub di codice
Sostituire la logica di connessione JMS o MQI esistente con uno stub di codice.

Supporto per diverse strategie WLM

Elenco CONNAME
Solo con priorità.

Ciò potrebbe avere un effetto negativo sul codice.

MultiQMGR CCDT
Con priorità o casuale.

È probabile che ciò non abbia alcun impatto sul codice.

Bilanciamento del carico
Qualsiasi, inclusa ogni connessione per tutti i messaggi.

È probabile che ciò abbia un effetto positivo sul codice.

Stub di codice
Qualsiasi, incluso ogni messaggio per tutti i messaggi.

È probabile che ciò abbia un effetto positivo sul codice.

Sovraccarico delle prestazioni mentre il gestore code primario non è disponibile

Elenco CONNAME
Cerca sempre il primo nell'elenco.

Ciò potrebbe avere un effetto negativo sul codice.

MultiQMGR CCDT
Ricorda l'ultima buona connessione.

È probabile che ciò abbia un effetto positivo sul codice.

Bilanciamento del carico
Il monitoraggio della porta evita i gestori code errati.

È probabile che ciò abbia un effetto positivo sul codice.

Stub di codice
Può ricordare l'ultima buona connessione e riprovare in modo intelligente.

È probabile che ciò abbia un effetto positivo sul codice.

Supporto transazione XA

Elenco CONNAME, CCDT multi - QMGR e programma di bilanciamento del carico
Il gestore transazioni deve memorizzare le informazioni di recupero che si riconnette alla stessa risorsa del gestore code.

Generalmente, una chiamata MQCONN che si risolve in gestori code differenti invalida questa condizione. Ad esempio, in Java EE, una singola factory di connessione deve risolversi in un singolo gestore code quando si utilizza XA.

Ciò potrebbe avere un effetto negativo sul codice.

Stub di codice
Lo stub di codice può soddisfare requisiti XA per un gestore transazioni, ad esempio, più factory di connessione.

È probabile che ciò abbia un effetto positivo sul codice.

Flessibilità dell'amministratore per nascondere le modifiche dell'infrastruttura dalle app

Elenco CONNAME
Solo DNS.

Ciò potrebbe avere un effetto negativo sul codice.

MultiQMGR CCDT
DNS e file - system condiviso o file - system condiviso o file push CCDT.

È probabile che ciò non abbia alcun impatto sul codice.

Bilanciamento del carico
Indirizzo IP virtuale dinamico (VIP).

È probabile che ciò abbia un effetto positivo sul codice.

Stub di codice
Voci CCDT del gestore code singolo o DNS.

È probabile che ciò non abbia alcun impatto sul codice.

Evitare interruzioni della manutenzione pianificata

Esiste un'altra situazione che è necessario considerare e pianificare, ovvero come evitare interruzioni delle applicazioni, ad esempio, errori e timeout visibili per gli utenti finali, durante la manutenzione pianificata di un gestore code. L'approccio migliore per evitare interruzioni è quello di rimuovere tutto il lavoro da un gestore code prima che venga arrestato.

Considerare uno scenario di richiesta e risposta. Si desidera che tutte le richieste in corso vengano completate e che le risposte vengano elaborate dalla domanda, ma non si desidera che ulteriori lavori vengano inoltrati nel sistema. La semplice sospensione del gestore code non soddisfa questa necessità, poiché le applicazioni ben codificate ricevono un codice di ritorno RC2161 MQRC_Q_MGR_QUIESCING, prima di ricevere i relativi messaggi di risposta per le richieste incomplete.

È possibile impostare PUT (DISABLED) sulle code di richieste utilizzate per inoltrare il lavoro, lasciando le code di risposta sia PUT (ENABLED) che GET (ENABLED). In questo modo, è possibile monitorare la profondità delle code di richiesta, trasmissione e risposta. Una volta che tutte si sono stabilizzate, vale a dire che le richieste in corso sono complete o in timeout, è possibile arrestare il gestore code.

Tuttavia, è richiesta una buona codifica nelle applicazioni richiedenti per gestire una coda di richieste PUT (DISABLED), che risulta nel codice di ritorno RC2051 errore MQRC_PUT_INIITED, quando si tenta di inviare un messaggio.

Tenere presente che l'eccezione non si verifica quando si crea la connessione a IBM MQo quando si apre la coda di richieste. L'eccezione si verifica solo quando viene effettuato un tentativo di invio di un messaggio, utilizzando la chiamata MQPUT.

La creazione di uno stub di codice che include questa logica di gestione degli errori per gli scenari di richiesta e risposta e la richiesta ai team dell'applicazione di utilizzare tale stub di codice in futuro, può aiutare a sviluppare le applicazioni con un comportamento coerente.

V 9.3.0 Sviluppo di applicazioni client flessibili e scalabili

Per la tolleranza degli errori e la scalabilità, la distribuzione delle applicazioni client che supportano le opzioni di connessione in cluster uniformi consente alle istanze dell'applicazione di essere ribilanciate tra i gestori code.

Consultare Informazioni sui cluster uniformi per una panoramica dei cluster uniformi.

Idealmente, questo ribilanciamento è invisibile all'applicazione, ma solo alcuni tipi di applicazione sono adatti per questo tipo di distribuzione e potrebbe essere necessario prendere in considerazione la progettazione dell'applicazione.

Queste considerazioni si suddividono in due categorie principali:

- Rari *percorsi di errore* che potrebbero già esistere per le applicazioni riconnettibili, ma diventano più probabili quando vengono distribuiti in un cluster uniforme. Ad esempio, dopo una riconnessione, viene eseguito il backout di qualsiasi unità di lavoro in fase di elaborazione e i cursori di esplorazione vengono reimpostati. Questo potrebbe essere un evento raro per la tua applicazione riconnettibile nel suo ambiente corrente e, quindi, non gestito nel modo più pulito possibile dal codice dell'applicazione. La revisione della logica dell'applicazione, per garantire una gestione appropriata per tali situazioni, consente di evitare problemi imprevisti.
- *Affinità* con uno specifico gestore code. Se si sa che un'applicazione deve sempre riconnettersi allo stesso gestore code o a un gestore code specifico, è necessario che l'applicazione sia configurata per riconnettersi a tale gestore code o che non sia abilitata la connessione a tale gestore code. Tuttavia, queste affinità potrebbero essere temporanee, ad esempio in attesa di un messaggio di risposta. L'influenza dell'algoritmo di bilanciamento per tenere conto di queste affinità dal codice dell'applicazione viene discusso nella seguente sezione. Per ulteriori dettagli su queste opzioni e su come ottenere un approccio simile tramite la configurazione, piuttosto che il codice dell'applicazione, vedi Influencing application re-balancing in uniform clusters.

Influenza delle opzioni di riconnessione in MQI

Consultare Opzioni di riconnessione per ulteriori informazioni su MQCNO_RECONNECT.

Se si sa che un'applicazione deve sempre riconnettersi allo stesso gestore code o a un gestore code specifico, deve essere configurata come MQCNO_RECONNECT_Q_MGR o MQCNO_RECONNECT_DISABLED.

Influenza dell'algoritmo di bilanciamento in MQI

Tuttavia, è possibile che si desideri controllare o influenzare tale comportamento di ribilanciamento per soddisfare le necessità di specifici tipi di applicazione; ad esempio, ridurre le interruzioni delle transazioni in corso o garantire che le applicazioni del richiedente ricevano le relative risposte prima di essere spostate.

Alcuni comportamenti desiderati predefiniti vengono assunti e discussi in Influencing application re-balancing in uniform clusters. È inoltre possibile influire sul comportamento di applicazioni specifiche al momento della configurazione o della distribuzione tramite il file client.ini , come descritto in tale argomento.

In altre situazioni, potrebbe avere più senso rendere il comportamento di bilanciamento e i requisiti parte della logica dell'applicazione. In questi casi, le stesse caratteristiche rilevanti dell'applicazione possono essere fornite a IBM MQ quando ci si connette al gestore code sulla chiamata MQCONNX, in una struttura denominata MQBNO (opzioni di bilanciamento).

Se si fornisce una struttura MQBNO, è necessario fornire tutte le informazioni richieste da IBM MQ per prendere una decisione su come e quando all'applicazione deve essere richiesto di riconnettersi a un gestore code differente.

È necessario fornire:

- Il **Type** dell'applicazione
- Il **Timeout** in cui l'istanza viene ribilanciata indipendentemente dallo stato
- Qualsiasi **BalanceOptions** speciale

L'eccezione a questo, è che è possibile lasciare il timeout come MQBNO_TIMEOUT_DEFAULT, se necessario. In questo caso, il timeout si risolve in un qualsiasi valore nel file client.ini , nell'applicazione o nelle stanze globali, se fornite, e in caso contrario, nel valore predefinito di base di 10 secondi.

Consultare MQBNO per i dettagli sul formato di questa struttura.

Per le applicazioni .NET, consultare Influencing application re-balancing in .NET per ulteriori informazioni.

Applicazioni orientate agli oggetti

IBM MQ fornisce supporto per JMS, Java, C + + e .NET. Questi linguaggi e framework utilizzano IBM MQ Object Model, che fornisce classi che forniscono la stessa funzionalità delle chiamate e delle strutture IBM MQ .

Alcuni dei linguaggi e dei framework che utilizzano il modello oggetto IBM MQ forniscono funzioni aggiuntive che non sono disponibili quando si utilizzano i linguaggi procedurali con MQI (message queue interface).

Per dettagli sulle classi, i metodi e le proprietà forniti da questo modello, vedere “Il modello oggetto IBM MQ” a pagina 17.

JMS

IBM MQ fornisce classi che implementano le specifiche Jakarta Messaging 3.0 e Java Message Service 2.0 . Per i dettagli su IBM MQ classes for JMS, consultare Utilizzo IBM MQ classes for JMS. Per informazioni sulle differenze tra IBM MQ classes for Java e IBM MQ classes for JMS, per decidere quale utilizzare, consultare “Sviluppo di applicazioni JMS/Jakarta Messaging e Java” a pagina 83.

IBM MQ Message Service Client (XMS) for C/C++ e IBM MQ Message Service Client (XMS) for .NET forniscono un'API (application programming interface) denominata XMS che ha la stessa serie di interfacce di Java Message Service (JMS) API. Per ulteriori informazioni, fare riferimento a “Sviluppo di applicazioni XMS .NET” a pagina 616.

Java

Consultare Utilizzo di IBM MQ classes for Java per informazioni sulla codifica dei programmi utilizzando IBM MQ Object Model in Java.

Stabilized IBM non apporterà ulteriori miglioramenti a IBM MQ classes for Java e saranno funzionalmente stabilizzati al livello fornito in IBM MQ 8.0. Per informazioni sulle differenze tra IBM MQ classes for Java e IBM MQ classes for JMS per decidere quale utilizzare, consultare “Sviluppo di applicazioni JMS/Jakarta Messaging e Java” a pagina 83.

C++

IBM MQ fornisce classi C+ + equivalenti a oggetti IBM MQ e alcune classi aggiuntive equivalenti a tipi di dati array. Fornisce un certo numero di funzioni non disponibili tramite MQI. Consultare Utilizzo di C++ per informazioni sulla codifica di programmi che utilizzano IBM MQ Object Model in C + +. Message Service Clients for C/C++ e .NET forniscono un'API (application programming interface) denominata XMS che ha la stessa serie di interfacce di Java Message Service (JMS) API.

.NET

Consultare Sviluppo di applicazioni .NET per informazioni sulla codifica dei programmi .NET mediante le classi IBM MQ .NET . Message Service Clients per C/C++ e .NET forniscono un'API (application programming interface) denominata XMS che ha la stessa serie di interfacce di Java Message Service (JMS) API.

Concetti correlati

“Sviluppo di applicazioni MQI con IBM MQ” a pagina 717
IBM MQ fornisce il supporto per C, Visual Basic, COBOL, Assembler, RPG, pTALe PL/I. Questi linguaggi procedurali utilizzano l'interfaccia MQI (Message Queue Interface) per accedere ai servizi di accodamento messaggi.

Panoramica tecnica

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

Riferimenti correlati

Riferimento per lo sviluppo di applicazioni

Il modello oggetto IBM MQ

IBM MQ Object Model è composto da classi, metodi e proprietà.

IBM MQ Object Model è costituito da:

- *Classi* che rappresentano concetti familiari di IBM MQ quali gestori code, code e messaggi.
- *Metodi* su ciascuna classe corrispondente a chiamate MQI.
- *Proprietà* su ciascuna classe corrispondente ad attributi di oggetti IBM MQ .

Quando si crea una applicazione IBM MQ utilizzando il modello oggetto IBM MQ , si creano istanze di queste classi nell'applicazione. Un'istanza di una classe nella programmazione orientata agli oggetti viene denominata *oggetto*. Quando un oggetto viene creato, si interagisce con l'oggetto esaminando o impostando i valori delle proprietà dell'oggetto (l'equivalente dell'emissione di una chiamata MQINQ o MQSET) e effettuando chiamate di metodo sull'oggetto (l'equivalente dell'emissione di altre chiamate MQI).

Classi

IBM MQ Object Model fornisce la seguente serie di classi di base.

L'effettiva implementazione del modello varia leggermente tra i diversi ambienti orientati agli oggetti supportati.

MQQueueManager
Un oggetto della classe MQQueueManager rappresenta una connessione a un gestore code. Dispone di metodi per Connect (), Disconnect (), Commit () e Backout () (l'equivalente di MQCONN o MQCONNX, MQDISC, MQCMIT e MQBACK). Dispone di proprietà corrispondenti agli attributi di un gestore code. L'accesso a una proprietà dell'attributo del gestore code si connette implicitamente al gestore code, se non è già connesso. L'eliminazione di un oggetto MQQueueManager si disconnette implicitamente dal gestore code.

MQQUEUE
Un oggetto della classe MQQueue rappresenta una coda. Dispone di metodi per inserire () e richiamare () i messaggi da e verso la coda (l'equivalente di MQPUT e MQGET). Ha proprietà corrispondenti agli attributi di una coda. L'accesso a una proprietà dell'attributo della coda o l'emissione di una chiamata al metodo Put () o Get () apre implicitamente la coda (l'equivalente di MQOPEN). L'eliminazione di un oggetto MQQueue chiude implicitamente la coda (l'equivalente di MQCLOSE).

MQArgomento
Un oggetto della classe MQTopic rappresenta un argomento. Dispone di metodi per inserire () (pubblicare) e Get () (ricevere o sottoscrivere) i messaggi da e verso l'argomento (l'equivalente di MQPUT e MQGET). Ha proprietà corrispondenti agli attributi di un argomento. È possibile accedere a un oggetto MQTopic solo per la pubblicazione o la sottoscrizione, non contemporaneamente. Quando viene utilizzato per ricevere i messaggi, l'oggetto MQTopic può essere creato con una sottoscrizione non gestita o gestita e come sottoscrittore durevole o non durevole - vengono forniti più costruttori sovraccaricati per questi diversi scenari.

Messaggio MQT
Un oggetto della classe MQMessage rappresenta un messaggio da inserire in una coda o da ottenere da una coda. Contiene un buffer e incapsula sia i dati dell'applicazione che MQMD. Dispone di proprietà corrispondenti ai campi MQMD e ai metodi che consentono di scrivere e leggere i dati utente di diversi tipi (ad esempio, stringhe, numeri interi lunghi, numeri interi brevi, byte singoli) nel e dal buffer.

Opzioni MQPutMessage
Un oggetto della classe Opzioni MQPutMessagerappresenta la struttura MQPMO. Ha proprietà corrispondenti ai campi MQPMO.

Opzioni MQGetMessage
Un oggetto della classe Opzioni MQGetMessagerappresenta la struttura MQGMO. Ha proprietà corrispondenti ai campi MQGMO.

Processo MQ
Un oggetto della classe MQProcess rappresenta una definizione di processo (utilizzata con il trigger). Dispone di proprietà che rappresentano gli attributi di una definizione di processo.

Multi **MQDistributionList**
Un oggetto della classe MQDistributionList rappresenta un elenco di distribuzione (utilizzato per inviare più messaggi con un singolo MQPUT). Contiene un elenco di oggetti MQDistributionListItem.

Multi **MQDistributionListElemento**
Un oggetto della classe di elementi MQDistributionListrappresenta una singola destinazione dell'elenco di distribuzione. Comprende le strutture MQOR, MQRR e MQPMR e dispone di proprietà corrispondenti ai campi di tali strutture.

riferimenti a oggetti

In un programma IBM MQ che utilizza MQI, IBM MQ restituisce gli handle di connessione e gli handle di oggetto al programma.

Questi handle devono essere passati come parametri nelle successive chiamate IBM MQ . Con IBM MQ Object Model, questi handle sono nascosti dal programma applicativo. Invece, la creazione di un oggetto da una classe determina la restituzione di un riferimento oggetto al programma applicativo. È questo riferimento oggetto che viene utilizzato quando si effettuano chiamate di metodo e accessi di proprietà rispetto all'oggetto.

Codici di ritorno

L'emissione di una chiamata di metodo o l'impostazione di un valore di proprietà comporta l'impostazione di codici di ritorno.

Questi codici di ritorno sono un codice di completamento e un codice motivo e sono essi stessi proprietà dell'oggetto. I valori del codice di completamento e del codice motivo sono gli stessi definiti per MQI, con alcuni valori aggiuntivi specifici dell'ambiente orientato agli oggetti.

IBM MQ messaggi

Un messaggio IBM MQ è costituito da proprietà del messaggio e dati dell'applicazione. L'MQMD (message queuing message descriptor) contiene le informazioni di controllo che accompagnano i dati dell'applicazione quando un messaggio viaggia tra le applicazioni mittente e ricevente.

Parti di un messaggio

I messaggi IBM MQ sono composti da due parti:

- Proprietà dei messaggi
- Dati applicazione

Figura 2 a pagina 19 rappresenta un messaggio e mostra come è diviso logicamente in proprietà del messaggio e dati dell'applicazione.

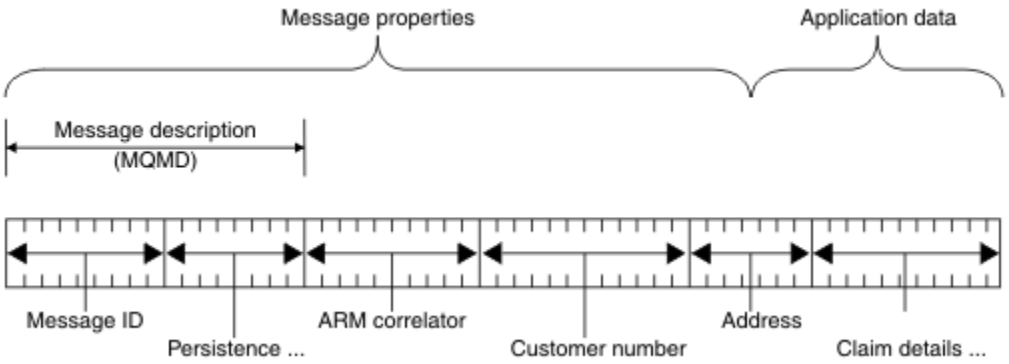


Figura 2. Rappresentazione di un messaggio

I dati dell'applicazione trasmessi in un messaggio IBM MQ non vengono modificati da un gestore code a meno che su di essi non venga eseguita la conversione dei dati. Inoltre, IBM MQ non pone alcuna limitazione sul contenuto di questi dati. La lunghezza dei dati in ogni messaggio non può superare il valore dell'attributo **MaxMsgLength** sia della coda che del gestore code.

ALW Su AIX, Linux, and Windows, l'attributo *MaxMsgLength* del gestore code e della coda assume il valore predefinito di 4 MB (4 194 304 byte) che è possibile modificare fino a un massimo di 100 MB (104 857 600 byte), se necessario.

IBM i Su IBM i, l'attributo *MaxMsgLength* del gestore code e della coda assume il valore predefinito di 4 MB (4 194 304 byte) che è possibile modificare fino a un massimo di 100 MB (104 857 600 byte), se necessario. Se si intende utilizzare messaggi IBM MQ superiori a 15 MB su IBM i, consultare “Creazione dell'applicazione procedurale su IBM i” a pagina 1008.

z/OS Su z/OS, l'attributo **MaxMsgLength** del gestore code è fissato a 100 MB e l'attributo **MaxMsgLength** della coda è impostato per default su 4 MB (4 194 304 byte) che è possibile modificare fino a un massimo di 100 MB, se necessario.

In alcune circostanze, rendere i messaggi leggermente più brevi del valore dell'attributo **MaxMsgLength** . Per ulteriori informazioni, consultare “I dati nel messaggio” a pagina 758.

Si crea un messaggio quando si utilizzano le chiamate MQPUT o MQPUT1 MQI. Come input per queste chiamate, si forniscono le informazioni di controllo (come la priorità del messaggio e il nome di una coda di risposta) e i propri dati, quindi la chiamata inserisce il messaggio in una coda. Per ulteriori informazioni su queste chiamate, fare riferimento a MQPUT e MQPUT1 .

Descrittore messaggio

È possibile accedere alle informazioni di controllo del messaggio utilizzando la struttura MQMD, che definisce il *descrittore del messaggio*.

Per una descrizione completa della struttura MQMD, consultare MQMD - Descrittore messaggio.

Per una descrizione di come utilizzare i campi in MQMD che contengono informazioni sull'origine del messaggio, consultare “Contesto messaggio” a pagina 48.

Esistono diverse versioni del descrittore del messaggio. Ulteriori informazioni per raggruppare e segmentare i messaggi (consultare “Gruppi di messaggi” a pagina 45) sono fornite nella Versione 2 del descrittore del messaggio (o MQMDE). È uguale al descrittore del messaggio della Versione 1, ma contiene campi supplementari. Questi campi sono descritti in MQMDE - Message descriptor extension.

Tipi di messaggio

Esistono quattro tipi di messaggi definiti da IBM MQ.

Questi quattro messaggi sono:

- pacchetto dati
- Messaggi di richiesta
- Messaggi di risposta
- Messaggi di report
  - Tipi di messaggio di prospetto
  - Opzioni messaggio report

Le applicazioni possono utilizzare i primi tre tipi di messaggi per passare le informazioni tra loro. Il quarto tipo, report, viene utilizzato dalle applicazioni e dai gestori code per riportare informazioni sugli eventi, ad esempio la ricorrenza di un errore.

Ogni tipo di messaggio è identificato da un valore MQMT_ *. È inoltre possibile definire i propri tipi di messaggio. Per l'intervallo di valori che è possibile utilizzare, consultare MsgType.

Datagrammi

Utilizzare un *datagramma* quando non è necessaria una risposta dall'applicazione che riceve il messaggio (ossia, richiama il messaggio dalla coda).

Un esempio di un'applicazione che potrebbe utilizzare i datagrammi è quella che visualizza le informazioni di volo in una lounge dell'aeroporto. Un messaggio potrebbe contenere i dati per un'intera schermata di informazioni sul volo. È improbabile che un'applicazione di questo tipo richieda un riconoscimento per un messaggio perché probabilmente non importa se un messaggio non viene consegnato. L'applicazione invia un messaggio di aggiornamento dopo un breve periodo di tempo.

Messaggi di richiesta

Utilizzare il *messaggio di richiesta* quando si desidera una risposta dall'applicazione che riceve il messaggio.

Un esempio di applicazione che potrebbe utilizzare i messaggi di richiesta è quello che visualizza il saldo di un conto corrente. Il messaggio di richiesta potrebbe contenere il numero del conto e il messaggio di risposta potrebbe contenere il saldo del conto.

Se si desidera collegare il messaggio di risposta al messaggio di richiesta, esistono due opzioni:

- Rendere responsabile l'applicazione che gestisce il messaggio di richiesta per assicurarsi che immetta le informazioni nel messaggio di replica correlato al messaggio di richiesta.
- Utilizzare il campo report nel descrittore del messaggio del messaggio di richiesta per specificare il contenuto dei campi *MsgId* e *CorrelId* del messaggio di risposta:
  - È possibile richiedere che *MsgId* o *CorrelId* del messaggio originale vengano copiati nel campo *CorrelId* del messaggio di risposta (l'azione predefinita è copiare *MsgId*).
  - È possibile richiedere che venga creato un nuovo *MsgId* per il messaggio di replica o che il *MsgId* del messaggio originale venga copiato nel campo *MsgId* del messaggio di replica (l'azione predefinita è la generazione di un nuovo identificativo del messaggio).

Messaggi di risposta

Utilizzare un *messaggio di risposta* quando si risponde ad un altro messaggio.

Quando si crea un messaggio di risposta, rispettare le opzioni impostate nel descrittore del messaggio a cui si risponde. Le opzioni del report specificano il contenuto dei campi identificativo del messaggio (*MsgId*) e identificativo di correlazione (*CorrelId*). Questi campi consentono all'applicazione che riceve la risposta di correlarla alla sua richiesta originale.

Messaggi di report

I *messaggi di report* informano le applicazioni su eventi quali la ricorrenza di un errore durante l'elaborazione di un messaggio.

Possono essere generati da:

- Un gestore code,
- Un agente del canale dei messaggi (ad esempio, se non è in grado di consegnare il messaggio) oppure
- Un'applicazione (ad esempio, se non può utilizzare i dati nel messaggio).

I messaggi di report possono essere generati in qualsiasi momento e potrebbero arrivare su una coda quando l'applicazione non li prevede.

Tipi di messaggi di report

Quando si inserisce un messaggio su una coda, è possibile scegliere di ricevere:

- Un *messaggio di report di eccezioni*. Questo viene inviato in risposta a un messaggio con l'indicatore delle eccezioni impostato. Viene generato dall'MCA (message channel agent) o dall'applicazione.
- Un *messaggio di report scadenza*. Ciò indica che un'applicazione ha tentato di richiamare un messaggio che aveva raggiunto la soglia di scadenza; il messaggio è contrassegnato per essere eliminato. Questo tipo di report viene generato dal gestore code.
- Un *messaggio di report di conferma di arrivo (COA)*. Ciò indica che il messaggio ha raggiunto la coda di destinazione. Viene generato dal gestore code.
- Un *messaggio di report COD (Conferma di consegna)*. Ciò indica che il messaggio è stato richiamato da un'applicazione ricevente. Viene generato dal gestore code.
- Un *messaggio di report PAN (positive action notification)*. Ciò indica che una richiesta è stata eseguita correttamente (ovvero, l'azione richiesta nel messaggio è stata eseguita correttamente). Questo tipo di report viene generato dall'applicazione.
- Un *messaggio di report NAN (negative action notification)*. Ciò indica che una richiesta non è stata eseguita correttamente (ovvero, l'azione richiesta nel messaggio non è stata eseguita correttamente). Questo tipo di report viene generato dall'applicazione.

Nota: Ogni tipo di messaggio di report contiene uno dei seguenti:

- L'intero messaggio originale
- I primi 100 byte di dati nel messaggio originale
- Nessun dato dal messaggio originale

È possibile richiedere più di un tipo di messaggio di report quando si inserisce un messaggio in una coda. Se si seleziona il messaggio di conferma della consegna e le opzioni del messaggio di report di eccezioni, se il messaggio non viene consegnato, si riceve un messaggio di report di eccezioni. Tuttavia, se si seleziona solo l'opzione del messaggio di report di conferma della consegna e il messaggio non viene consegnato, non si riceve un messaggio di report di eccezioni.

I messaggi di report richiesti, quando vengono soddisfatti i criteri per la generazione di un determinato messaggio, sono gli unici che si ricevono.

Opzioni del messaggio di report

È possibile *eliminare* un messaggio dopo che si è verificata un'eccezione. Se si seleziona l'opzione di eliminazione e si è richiesto un messaggio di report di eccezioni, il messaggio di report viene inviato a *ReplyToQ* e *ReplyToQMgr* e il messaggio originale viene eliminato.

Nota: Un vantaggio di questo è che è possibile ridurre il numero di messaggi che vanno alla coda di messaggi non recapitabili. Tuttavia, significa che l'applicazione, a meno che non invii solo messaggi datagramma, deve gestire i messaggi restituiti. Quando viene generato un messaggio di report di eccezioni, eredita la persistenza del messaggio originale.

Se un messaggio di report non può essere consegnato (se la coda è piena, ad esempio), il messaggio di report viene posizionato nella coda di messaggi non recapitabili.

Se si desidera ricevere un messaggio di report, specificare il nome della coda di risposta nel campo *ReplyToQ* ; altrimenti MQPUT o MQPUT1 del messaggio originale non riesce con MQRC_MISSING_REPLY_TO_Q.

È possibile utilizzare altre opzioni di report nel descrittore del messaggio (MQMD) di un messaggio per specificare il contenuto dei campi *MsgId* e *CorrelId* di qualsiasi messaggio di report creato per il messaggio:

- È possibile richiedere che il *MsgId* o il *CorrelId* del messaggio originale vengano copiati nel campo *CorrelId* del messaggio di report. L'azione predefinita consiste nel copiare l'identificativo del messaggio. Utilizzare MQRO_COPY_MSG_ID_TO_CORRELID poiché abilita il mittente di un messaggio a correlare il messaggio di risposta o di report con il messaggio originale. L'identificativo di correlazione del messaggio di risposta o di report è identico all'identificativo del messaggio originale.
- È possibile richiedere che venga generato un nuovo *MsgId* per il messaggio del report o che il *MsgId* del messaggio originale venga copiato nel campo *MsgId* del messaggio del report. L'azione predefinita è la generazione di un nuovo identificativo di messaggio. Utilizzare MQRO_NEW_MSG_ID in quanto garantisce che ogni messaggio nel sistema abbia un identificativo di messaggio diverso e possa essere distinto in modo non ambiguo da tutti gli altri messaggi nel sistema.
- Le applicazioni specializzate potrebbero dover utilizzare MQRO_PASS_MSG_ID o MQRO_PASS_CORREL_ID. Tuttavia, è necessario progettare l'applicazione che legge i messaggi dalla coda per essere certi che funzioni correttamente quando, ad esempio, la coda contiene più messaggi con lo stesso identificativo.

  Le applicazioni server devono controllare le impostazioni di questi indicatori nel messaggio di richiesta e impostare in modo appropriato i campi *MsgId* e *CorrelId* nel messaggio di risposta o di report.

  Le applicazioni che agiscono come intermediari tra un'applicazione richiedente e un'applicazione server non devono controllare le impostazioni di questi indicatori. Ciò è dovuto al fatto che queste applicazioni in genere devono inoltrare il messaggio all'applicazione server senza modificare i campi *MsgId*, *CorrelId*e *Report* . Ciò consente all'applicazione server di copiare *MsgId* dal messaggio originale nel campo *CorrelId* del messaggio di replica.

Quando si genera un report su un messaggio, le applicazioni server devono verificare se una di queste opzioni è stata impostata.

Per ulteriori informazioni su come utilizzare i messaggi di report, consultare Report.

Per indicare la natura del report, i gestori code utilizzano una serie di codici di feedback. Inseriscono questi codici nel campo *Feedback* del descrittore di messaggi di un messaggio di report. I gestori code possono anche restituire i codici motivo MQI nel campo *Feedback* . IBM MQ definisce un intervallo di codici di feedback per le applicazioni da utilizzare.

Per ulteriori informazioni sul feedback e sui codici di errore, consultare Feedback.

Un esempio di programma che potrebbe utilizzare un codice di feedback è quello che controlla i carichi di lavoro di altri programmi che servono una coda. Se c'è più di un'istanza di un programma che serve una coda, e il numero di messaggi che arrivano sulla coda non lo giustifica più, un tale programma può inviare un messaggio di report (con il codice di feedback MQFB_QUIT) a uno dei programmi di servizio per indicare che il programma deve terminare la sua attività. Un programma di controllo potrebbe utilizzare la chiamata MQINQ per scoprire quanti programmi servono una coda.

Multi *Report e messaggi segmentati*

Non supportato su IBM MQ for z/OS.

Se un messaggio è segmentato e si richiede la creazione di report, è possibile che si ricevano più report di quelli che si sarebbero avuti se il messaggio non fosse stato segmentato.

Per una descrizione dei messaggi segmentati, consultare “Segmentazione del messaggio” a pagina 793.

Per i report generati da IBM MQ

Se si segmentano i messaggi o si consente al gestore code di farlo, esiste un solo caso in cui è possibile prevedere di ricevere un singolo report per l'intero messaggio. Ciò si verifica quando si richiedono solo report COD e si è specificato MQGMO_COMPLETE_MSG sull'applicazione di richiamo.

In altri casi, l'applicazione deve essere preparata per gestire diversi report, di solito uno per ciascun segmento.

Nota: Se si segmentano i messaggi e sono necessari solo i primi 100 byte dei dati del messaggio originale da restituire, modificare l'impostazione delle opzioni del report per richiedere i report senza dati per i segmenti che hanno un offset di 100 o più. Se non si esegue questa operazione e si lascia l'impostazione in modo che ogni segmento richieda 100 byte di dati e si richiamano i messaggi di report con un singolo MQGET che specifica MQGMO_COMPLETE_MSG, i report si assemblano in un messaggio di grandi dimensioni contenente 100 byte di dati di lettura ad ogni offset appropriato. Se ciò si verifica, è necessario un buffer di grandi dimensioni oppure è necessario specificare MQGMO_ACCEPT_TRUNCATED_MSG.

Per i report generati dalle applicazioni

Se l'applicazione genera i report, copiare sempre le intestazioni IBM MQ presenti all'inizio dei dati del messaggio originale nei dati del messaggio del report.

Quindi, aggiungere nessuno, 100 byte o tutti i dati del messaggio originale (o qualsiasi altra quantità che si includa di solito) ai dati del messaggio di report.

È possibile riconoscere le intestazioni IBM MQ che devono essere copiate esaminando i nomi formato successivi, iniziando con MQMD e proseguendo attraverso le intestazioni presenti. I seguenti nomi `Format` indicano queste intestazioni IBM MQ :

- MQMDE
- MQDLH
- QQMQX
- MQIIH
- MQH*

MQH* indica qualsiasi nome che inizia con i caratteri MQH.

Il nome `Format` si verifica in posizioni specifiche per MQDLH e MQXQH, ma per le altre intestazione IBM MQ si verifica nella stessa posizione. La lunghezza dell'intestazione è contenuta in un campo che si verifica anche nella stessa posizione per MQMDE, MQIMSe tutte le intestazioni MQH*.

Se si utilizza un MQMD Versione 1 e si crea un report su un segmento, un messaggio in un gruppo o un messaggio per cui è consentita la segmentazione, i dati del report devono iniziare con un MQMDE. Impostare il campo *`OriginalLength`* sulla lunghezza dei dati del messaggio originale, escludendo le lunghezze delle intestazioni IBM MQ trovate.

Recupero di report

Se si richiedono i report COA o COD, è possibile richiedere che vengano riassemblati con MQGMO_COMPLETE_MSG.

MQGET con MQGMO_COMPLETE_MSG viene soddisfatto quando nella coda sono presenti messaggi di report sufficienti (di un singolo tipo, ad esempio COA, e con lo stesso *`GroupId`*) per rappresentare un messaggio originale completo. Ciò è vero anche se i messaggi di report stessi non contengono i dati originali completi; il campo *`OriginalLength`* in ogni messaggio di report fornisce la lunghezza dei dati originali rappresentati da quel messaggio di report, anche se i dati stessi non sono presenti.

È possibile utilizzare questa tecnica anche se sulla coda sono presenti diversi tipi di report (ad esempio, COA e COD), poiché un MQGET con MQGMO_COMPLETE_MSG riassembla i messaggi di report solo se hanno lo stesso codice *`Feedback`* . Tuttavia, di solito non è possibile utilizzare questa tecnica per i report di eccezioni, poiché, in genere, questi hanno codici *`Feedback`* differenti.

È possibile utilizzare questa tecnica per ottenere un'indicazione positiva dell'arrivo dell'intero messaggio. Tuttavia, nella maggior parte dei casi è necessario considerare la possibilità che alcuni segmenti arrivino mentre altri potrebbero generare un'eccezione (o la scadenza, se consentito). Non è possibile utilizzare MQGMO_COMPLETE_MSG in questo caso, poiché, in generale, è possibile ottenere codici *Feedback* diversi per segmenti diversi e più di un report per un segmento. Tuttavia, è possibile utilizzare MQGMO_ALL_SEGMENTS_AVAILABLE.

Per consentire ciò, potrebbe essere necessario richiamare i report man mano che arrivano e creare un'immagine nell'applicazione di ciò che è accaduto al messaggio originale. È possibile utilizzare il campo *GroupId* nel messaggio di report per correlare i report con il *GroupId* del messaggio originale e il campo *Feedback* per identificare il tipo di ogni messaggio di report. Il modo in cui si esegue questa operazione dipende dai requisiti dell'applicazione.

Un approccio è il seguente:

- Richiedere report COD e report di eccezioni.
- Dopo un periodo di tempo specifico, verificare se è stata ricevuta una serie completa di report COD utilizzando MQGMO_COMPLETE_MSG. In tal caso, l'applicazione sa che l'intero messaggio è stato elaborato.
- In caso contrario, e i report di eccezione relativi a questo messaggio sono presenti, gestire il problema come per i messaggi non segmentati, ma assicurarsi di ripulire i segmenti orfani ad un certo punto.
- Se ci sono segmenti per cui non ci sono report di alcun tipo, i segmenti originali (o i report) potrebbero essere in attesa che un canale venga riconnesso o la rete potrebbe essere sovraccaricata a un certo punto. Se non è stato ricevuto alcun report di eccezione (o se si pensa che quelli di cui si dispone potrebbero essere solo temporanei), è possibile decidere di lasciare che la propria applicazione attenda un po' più a lungo.

  Come in precedenza, ciò è simile alle considerazioni che si hanno quando si gestiscono messaggi non segmentati, ad eccezione del fatto che è necessario considerare anche la possibilità di ripulire i segmenti orfani.

Se il messaggio originale non è critico (ad esempio, se si tratta di una query o di un messaggio che può essere ripetuto successivamente), impostare una scadenza per garantire che i segmenti orfani vengano rimossi.

Gestori code di livello precedente

Quando un report viene generato da un gestore code che supporta la segmentazione, ma viene ricevuto su un gestore code che non supporta la segmentazione, la struttura MQMDE (che identifica i *Offset* e *OriginalLength* rappresentati dal report) viene sempre inclusa nei dati del report, oltre a zero, 100 byte o tutti i dati originali nel messaggio.

Tuttavia, se un segmento di un messaggio passa attraverso un gestore code che non supporta la segmentazione, se viene generato un report, la struttura MQMDE nel messaggio originale viene trattata esclusivamente come dati. Non viene pertanto incluso nei dati del prospetto se sono stati richiesti zero byte dei dati originali. Senza MQMDE, il messaggio di report potrebbe non essere utile.

Richiedere almeno 100 byte di dati nei report se esiste la possibilità che il messaggio possa viaggiare attraverso un gestore code di livello precedente.

Formato delle informazioni di controllo del messaggio e dei dati del messaggio

Il gestore code è interessato solo al formato delle informazioni di controllo all'interno di un messaggio, mentre le applicazioni che gestiscono il messaggio sono interessate al formato delle informazioni di controllo e dei dati.

Formato delle informazioni di controllo del messaggio

Le informazioni di controllo nei campi stringa di caratteri del descrittore del messaggio devono trovarsi nella serie di caratteri utilizzata dal gestore code.

L'attributo **CodedCharSetId** dell'oggetto gestore code definisce questa serie di caratteri. Le informazioni di controllo devono essere contenute in questa serie di caratteri perché, quando le applicazioni passano i messaggi da un gestore code a un altro, gli agent del canale dei messaggi che trasmettono i messaggi utilizzano il valore di questo attributo per determinare quale conversione dati eseguire.

Formato dei dati del messaggio

È possibile specificare uno dei seguenti elementi:

- Il formato dei dati dell'applicazione
- La serie di caratteri dei dati carattere
- Il formato dei dati numerici

Per eseguire questa operazione, utilizzare questi campi:

Format
Indica al destinatario di un messaggio il formato dei dati dell'applicazione nel messaggio.

Quando il gestore code crea un messaggio, in alcune circostanze utilizza il campo *Format* per identificare il formato di tale messaggio. Ad esempio, quando un gestore code non è in grado di consegnare un messaggio, lo inserisce in una coda di messaggi non recapitabili (messaggi non recapitati). Aggiunge un'intestazione (che contiene ulteriori informazioni di controllo) al messaggio e modifica il campo *Format* per visualizzarlo.

Il gestore code ha un certo numero di *formati integrati* con nomi che iniziano con MQ, ad esempio MQFMT_STRING. Se questi non soddisfano le proprie esigenze, è possibile definire i propri formati (*formati definiti dall'utente*), ma non è necessario utilizzare i nomi che iniziano con MQ per tali formati.

Quando si creano e si utilizzano i propri formati, è necessario scrivere un'uscita di conversione dati per supportare un programma che riceve il messaggio utilizzando MQGMO_CONVERT.

CodedCharSetId
Definisce la serie di caratteri dei dati nel messaggio. Se si desidera impostare questa serie di caratteri su quella del gestore code, è possibile impostare questo campo sulla costante MQCCSI_Q_MGR o MQCCSI_INHERIT.

Quando si ottiene un messaggio da una coda, confrontare il valore del campo *CodedCharSetId* con il valore previsto dall'applicazione. Se i due valori differiscono, potrebbe essere necessario convertire i dati carattere nel messaggio o utilizzare un'uscita del messaggio di conversione dati, se disponibile.

Encoding
Descrive il formato dei dati dei messaggi numerici che contengono numeri interi binari, interi decimali compressi e numeri a virgola mobile. Viene generalmente codificato in base alla particolare macchina su cui è in esecuzione il gestore code.

Quando si inserisce un messaggio su una coda, generalmente si specifica la costante MQENC_NATIVE nel campo *Encoding* . Ciò significa che la codifica dei dati del messaggio è la stessa della macchina su cui è in esecuzione l'applicazione.

Quando si riceve un messaggio da una coda, confrontare il valore del campo *Encoding* nel descrittore del messaggio con il valore della costante MQENC_NATIVE sulla macchina. Se i due valori differiscono, potrebbe essere necessario convertire qualsiasi dato numerico nel messaggio o utilizzare un'uscita del messaggio di conversione dati, se disponibile.

Conversione dati applicazione

È possibile che i dati dell'applicazione debbano essere convertiti nella serie di caratteri e nella codifica richiesta da un'altra applicazione quando si tratta di piattaforme differenti.

Può essere convertito nel gestore code di invio o nel gestore code di ricezione. Se la libreria di formati integrati non soddisfa le proprie esigenze, è possibile definirne di propri. Il tipo di conversione dipende dal formato del messaggio specificato nel campo formato del descrittore del messaggio, MQMD.

Nota: I messaggi con MQFMT_NONE specificato non vengono convertiti.

Conversione sul gestore code di invio

Impostare l'attributo del canale CONVERT su YES se è necessario l'MCA (message channel agent) di invio per convertire i dati dell'applicazione.

La conversione viene eseguita sul gestore code di invio per alcuni formati integrati e per i formati definiti dall'utente se viene fornita un'uscita utente adatta.

Formati integrati

Eccone alcune:

- Messaggi che sono tutti caratteri (utilizzando il nome formato MQFMT_STRING)
- Messaggi definiti da IBM MQ , ad esempio Programmable Command Formats

  IBM MQ utilizza i messaggi Programmable Command Format per i messaggi e gli eventi di gestione (in questo caso, il nome del formato utilizzato è MQFMT_ADMIN). È possibile utilizzare lo stesso formato (utilizzando il nome formato MQFMT_PCF) per i propri messaggi e sfruttare la conversione dei dati integrata.

I formati integrati del gestore code hanno tutti nomi che iniziano con MQFMT. Sono elencati e descritti in Formato.

Formati definiti dall'applicazione

Per i formati definiti dall'utente, la conversione dei dati dell'applicazione deve essere eseguita da un programma di uscita di conversione dati (per ulteriori informazioni, consultare “Scrittura delle uscite di conversione dati” a pagina 985). In un ambiente client-server, l'uscita viene caricata sul server e la conversione avviene lì.

Conversione nel gestore code di ricezione

I dati del messaggio dell'applicazione possono essere convertiti dal gestore code di ricezione per i formati integrati e definiti dall'utente.

La conversione viene eseguita durante l'elaborazione di una chiamata MQGET se si specifica l'opzione MQGMO_CONVERT. Per i dettagli, consultare Opzioni

Serie di caratteri codificati

I prodotti IBM MQ supportano le serie di caratteri codificati fornite dal sistema operativo sottostante.

Quando si crea un gestore code, il CCSID (coded character set ID) del gestore code utilizzato si basa su quello dell'ambiente sottostante. Se si tratta di una codepage mista, IBM MQ utilizza la parte SBCS della codepage mista come CCSID del gestore code.

Per la conversione dei dati generali, se il sistema operativo sottostante supporta le codepage DBCS, IBM MQ può utilizzarla.

Consultare la documentazione relativa al proprio sistema operativo per dettagli sulle serie di caratteri codificati supportate.

È necessario considerare la conversione dei dati dell'applicazione, i nomi di formato e le uscite utente quando si scrivono applicazioni che si estendono su più piattaforme. Consultare “Scrittura delle uscite di conversione dati” a pagina 985 per informazioni sul richiamo e la scrittura delle uscite di conversione dati.

Priorità dei messaggi

È possibile impostare la priorità del messaggio su un valore numerico oppure lasciare che il messaggio assuma la priorità predefinita della coda.

Impostare la priorità di un messaggio (nel campo *Priority* della struttura MQMD) quando si inserisce il messaggio su una coda. È possibile impostare un valore numerico per la priorità oppure lasciare che il messaggio assuma la priorità predefinita della coda.

L'attributo **MsgDeliverySequence** della coda determina se i messaggi sulla coda vengono memorizzati nella sequenza FIFO (first in, first out) o in FIFO all'interno della sequenza di priorità. Se questo attributo è impostato su MQMDS_PRIORITY, i messaggi vengono accodati con la priorità specificata nel campo *Priority* dei relativi descrittori di messaggi; ma se è impostato su MQMDS_FIFO, i messaggi vengono accodati con la priorità predefinita della coda. I messaggi di uguale priorità vengono memorizzati nella coda in ordine di arrivo.

L'attributo **DefPriority** di una coda imposta il valore di priorità predefinito per i messaggi inseriti in tale coda. Questo valore viene impostato quando la coda viene creata, ma può essere modificato successivamente. Le code alias e le definizioni locali delle code remote, possono avere priorità predefinite diverse dalle code di base in cui vengono risolte. Se nel percorso di risoluzione è presente più di una definizione di coda (consultare “Risoluzione nomi” a pagina 745), la priorità predefinita viene presa dal valore (al momento dell'operazione di inserimento) dell'attributo di **DefPriority** della coda specificata nel comando di apertura.

Il valore dell'attributi **MaxPriority** del gestore code è la massima priorità che è possibile assegnare a un messaggio elaborato da tale gestore code. Non è possibile modificare il valore di questo attributo. In IBM MQ, l'attributo ha il valore 9; è possibile creare messaggi con priorità comprese tra 0 (il più basso) e 9 (il più alto).

Proprietà dei messaggi

Utilizzare le proprietà del messaggio per consentire a un'applicazione di selezionare i messaggi da elaborare o per richiamare le informazioni su un messaggio senza accedere alle intestazioni MQMD o MQRFH2 . Inoltre, facilitano la comunicazione tra applicazioni IBM MQ e JMS .

Una proprietà del messaggio è un dato associato a un messaggio, costituito da un nome testuale e un valore di un tipo particolare. Le proprietà dei messaggi vengono utilizzate dai selettori dei messaggi per filtrare le pubblicazioni sugli argomenti o per richiamare in modo selettivo i messaggi dalle code. Le proprietà del messaggio possono essere utilizzate per includere i dati di business o le informazioni di stato senza doverle memorizzare nei dati dell'applicazione. Le applicazioni non devono accedere ai dati nelle intestazioni MQ Message Descriptor (MQMD) o MQRFH2 perché è possibile accedere ai campi in queste strutture di dati come proprietà del messaggio utilizzando le chiamate alla funzione MQI (Message Queue Interface).

L'utilizzo delle proprietà del messaggio in IBM MQ imita l'utilizzo delle proprietà in JMS. Ciò significa che è possibile impostare le proprietà in un'applicazione JMS e richiamarle in un'applicazione procedurale IBM MQ o viceversa. Per rendere una proprietà disponibile a un'applicazione JMS , assegnarle il prefisso "usr"; è quindi disponibile (senza il prefisso) come proprietà utente del messaggio JMS . Ad esempio, la proprietà IBM MQ *usr.myproperty* (una stringa di caratteri) è accessibile a un'applicazione JMS utilizzando la JMS chiamata `message.getStringProperty('myproperty')`. Notare che le applicazioni JMS non sono in grado di accedere alle proprietà con il prefisso "usr" se contengono due o più U+002E (".") caratteri. Una proprietà senza prefisso e senza U+002E (".") viene considerato come se avesse il prefisso "usr". Al contrario, è possibile accedere a una proprietà utente impostata in una applicazione JMS in un'applicazione IBM MQ aggiungendo "usr." al nome proprietà richiesto in una chiamata MQINQMP.

Proprietà del messaggio e lunghezza del messaggio

Utilizzare l'attributo del gestore code *MaxPropertiesLength* per controllare la dimensione delle proprietà che possono fluire con qualsiasi messaggio in un gestore code IBM MQ .

In generale, quando si usa MQSETMP per impostare le proprietà, la dimensione di una proprietà è la lunghezza del nome della proprietà in byte, più la lunghezza del valore della proprietà in byte come passato nella chiamata MQSETMP. È possibile che la serie di caratteri del nome della proprietà e il

valore della proprietà vengano modificati durante la trasmissione del messaggio alla relativa destinazione perché possono essere convertiti in Unicode; in questo caso, la dimensione della proprietà potrebbe essere modificata.

In una chiamata MQPUT o MQPUT1 , le proprietà del messaggio non vengono conteggiate per la lunghezza del messaggio per la coda e il gestore code, ma vengono conteggiate per la lunghezza delle proprietà percepite dal gestore code (indipendentemente dal fatto che siano state impostate utilizzando o meno le chiamate MQI della proprietà del messaggio).

Se la dimensione delle proprietà supera la lunghezza massima, il messaggio viene rifiutato con MQRC_PROPERTIES_TOO_BIG. Poiché la dimensione delle proprietà dipende dalla relativa rappresentazione, è necessario impostare la lunghezza massima delle proprietà ad un livello lordo.

È possibile che un'applicazione inserisca correttamente un messaggio con un buffer maggiore del valore di *MaxMsgLength*, se il buffer include le proprietà. Questo perché, anche quando rappresentate come elementi MQRFH2 , le proprietà del messaggio non vengono conteggiate per la lunghezza del messaggio. I campi di intestazione MQRFH2 vengono aggiunti alla lunghezza delle proprietà solo se una o più cartelle sono contenute e ogni cartella nell'intestazione contiene proprietà. Se una o più cartelle sono contenute nell'intestazione MQRFH2 e qualsiasi cartella non contiene proprietà, i campi dell'intestazione MQRFH2 vengono conteggiati per la lunghezza del messaggio.

Su una chiamata MQGET, le proprietà del messaggio non vengono conteggiate nella lunghezza del messaggio per quanto riguarda la coda e il gestore code. Tuttavia, poiché le proprietà vengono conteggiate separatamente, è possibile che il buffer restituito da una chiamata MQGET sia maggiore del valore dell'attributo *MaxMsgLength* .

Non fare in modo che le applicazioni interrogino il valore di *MaxMsgLength* e quindi assegnino un buffer di questa dimensione prima di chiamare MQGET; assegnare invece un buffer che si consideri sufficientemente grande. Se MQGET ha esito negativo, assegnare un buffer guidato dalla dimensione del parametro *DataLength* .

Il parametro *DataLength* della chiamata MQGET restituisce la lunghezza in byte dei dati dell'applicazione e tutte le proprietà restituite nel buffer fornito, se non è stato specificato un handle del messaggio nella struttura MQGMO.

Il parametro *Buffer* della chiamata MQPUT contiene i dati del messaggio dell'applicazione da inviare e tutte le proprietà rappresentate nei dati del messaggio.

Esiste un limite di lunghezza di 100 MB per le proprietà del messaggio, escluso il descrittore del messaggio o l'estensione per ogni messaggio.

La dimensione di una proprietà nella sua rappresentazione interna è la lunghezza del nome, più la dimensione del suo valore, più alcuni dati di controllo per la proprietà. Ci sono anche alcuni dati di controllo per la serie di proprietà dopo che una proprietà è stata aggiunta al messaggio.

Nomi proprietà

Un nome proprietà è una stringa di caratteri. Alcune restrizioni si applicano alla sua lunghezza e alla serie di caratteri che possono essere utilizzati.

Un nome proprietà è una stringa di caratteri sensibile al maiuscolo / minuscolo, limitata a +4095 caratteri a meno che non sia diversamente limitato dal contesto. Questo limite è contenuto nella costante MQ_MAX_PROPERTY_NAME_LENGTH.

Se si supera questa lunghezza massima quando si utilizza una chiamata MQI della proprietà del messaggio, la chiamata ha esito negativo con codice motivo MQRC_PROPERTY_NAME_LENGTH_ERR.

Poiché non esiste una lunghezza massima del nome della proprietà in JMS, è possibile per un'applicazione JMS impostare un nome proprietà JMS valido che non sia un nome proprietà IBM MQ valido quando viene memorizzato in una struttura MQRFH2 .

In questo caso, quando analizzato, vengono utilizzati solo i primi 4095 caratteri del nome della proprietà; i seguenti caratteri vengono troncati. Ciò potrebbe far sì che un'applicazione che utilizza i selettori non riesca a corrispondere a una stringa di selezione o a corrispondere a una stringa quando non previsto,

poiché più di una proprietà potrebbe troncare allo stesso nome. Quando un nome proprietà viene troncato, WebSphereMQ emette un messaggio di log degli errori.

Tutti i nomi proprietà devono seguire le regole definite dalla specifica del linguaggio Java per gli identificativi Java , con l'eccezione che il carattere Unicode U+002E (.) è consentito come parte del nome, ma non l'inizio. Le regole per gli identificatori Java sono uguali a quelle contenute nella specifica JMS per i nomi proprietà.

I caratteri spazio e gli operatori di comparazione non sono consentiti. I valori null incorporati sono consentiti in un nome proprietà ma non sono consigliati. Se si utilizzano valori null incorporati, ciò impedisce l'utilizzo della costante MQVS_NULL_TERMINATED quando viene utilizzata con la struttura MQCHARV per specificare le stringhe di lunghezza variabile.

Mantenere i nomi delle proprietà semplici, poiché le applicazioni possono selezionare i messaggi in base ai nomi delle proprietà e la conversione tra la serie di caratteri del nome e del selettore potrebbe causare un errore imprevisto della selezione.

I nomi delle proprietà IBM MQ utilizzano il carattere U+002E (.) per il raggruppamento logico delle proprietà. Divide lo spazio dei nomi per le proprietà. Le proprietà con i seguenti prefissi, in qualsiasi combinazione di lettere minuscole o maiuscole, sono riservate per l'uso da parte del prodotto:

- `mcd`
- `jms`
- `usr`
- `mq`
- `sib`
- `wmq`
- `Root`
- `Body`
- `Properties`

Un buon modo per evitare conflitti di nomi consiste nel garantire che tutte le applicazioni prefissino le proprietà del messaggio con il nome del dominio Internet. Ad esempio, se si sta sviluppando un'applicazione utilizzando il nome dominio `ourcompany.com` è possibile denominare tutte le proprietà con il prefisso `com.ourcompany`. Questa convenzione di denominazione consente inoltre una facile selezione delle proprietà; ad esempio, un'applicazione può analizzare tutte le proprietà del messaggio a partire da `com.ourcompany.%`.

Per ulteriori informazioni sull'utilizzo dei nomi delle proprietà, consultare Limitazioni dei nomi delle proprietà .

Limitazioni nome proprietà
Quando si assegna un nome a una proprietà, è necessario osservare alcune regole.

Le seguenti limitazioni si applicano ai nomi proprietà:

1. Una proprietà non deve iniziare con le seguenti stringhe:
   - "JMS" - riservato all'utilizzo da parte di IBM MQ classes for JMS.
   - "usr.JMS" - non valido.

   Le uniche eccezioni sono le seguenti proprietà che forniscono sinonimi per proprietà JMS :

Proprietà	**Sinonimo di**
JMSCorrelationID	Root.MQMD.CorrelId o jms.Cid
JMSDeliveryMode	Root.MQMD.Persistence o jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root.MQMD.Expiry o jms.Exp

Proprietà	Sinonimo di
JMSMessageID	Root.MQMD.MsgId
JMSPriority	Root.MQMD.Priority o jms.Pri
JMSRedelivered	Root.MQMD.BackoutCount
JMSReplyTo (una stringa codificata come URI)	Root.MQMD.ReplyToQ o Root.MQMD.ReplyToQMgr o jms.Rto
JMSTimestamp	Root.MQMD.PutDate o Root.MQMD.PutTime o jms.Tms
JMSType	mcd.Type o mcd.Set o mcd.Fmt
JMSXAppID	Root.MQMD.PutApplName
JMSXDeliveryCount	Root.MQMD.BackoutCount
JMSXGroupID	Root.MQMD.GroupId o jms.Gid
JMSXGroupSeq	Root.MQMD.MsgSeqNumber o jms.Seq
JMSXUserID	Root.MQMD.UserIdentifier

Questi sinonimi consentono a un'applicazione MQI di accedere alle proprietà JMS in modo simile all'applicazione client IBM MQ classes for JMS . Di queste proprietà, solo JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupIDe JMSXGroupSeq possono essere impostati utilizzando MQI.

Notare che le proprietà JMS_IBM_ * disponibili da IBM MQ classes for JMS non sono disponibili utilizzando MQI. Le applicazioni MQI possono accedere ai campi a cui fanno riferimento le proprietà JMS_IBM_ * in altri modi.

2. Una proprietà non deve essere richiamata, in una combinazione di caratteri minuscoli o maiuscoli, "NULL", "TRUE", "FALSE", "NOT", "E", "OR", "BETWEEN", "LIKE", "IN", "IS" e "ESCAPE". Questi sono i nomi delle parole chiave SQL utilizzate nelle stringhe di selezione.
3. Un nome proprietà che inizia " mq " in qualsiasi combinazione di lettere minuscole o maiuscole e non iniziando "mq_usr" può contenere solo un "." U+002E). Più "." non sono consentiti nelle proprietà con tali prefissi.
4. Due "." i caratteri devono contenere altri caratteri; non è possibile avere un punto vuoto nella gerarchia. Allo stesso modo, un nome proprietà non può terminare con un "." carattere.
5. Se un'applicazione imposta la proprietà "a.b" e quindi la proprietà "a.b.c", non è chiaro se nella gerarchia "b" contenga un valore o un altro raggruppamento logico. Tale gerarchia è "contenuto misto" e non è supportata. L'impostazione di una proprietà che causa contenuto misto non è consentita.

Queste limitazioni vengono applicate dal meccanismo di convalida nel modo seguente:

- I nomi delle proprietà vengono convalidati quando si imposta una proprietà utilizzando la chiamata MQSETMP - Impostare la proprietà del messaggio , se la convalida è stata richiesta quando è stato creato l'handle del messaggio. Se viene effettuato un tentativo di convalida di una proprietà e non riesce a causa di un errore nella specifica del nome della proprietà, il codice di completamento è MQCC_FAILED con il seguente motivo:
  - MQRC_PROPERTY_NAME_ERROR per i motivi 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED per il motivo 5.
- Non è garantito che i nomi delle proprietà specificate direttamente come elementi MQRFH2 vengano convalidati dalla chiamata MQPUT.

Campi del descrittore del messaggio come proprietà
La maggior parte dei campi descrittori di messaggi può essere considerata come proprietà. Il nome della proprietà viene creato aggiungendo un prefisso al nome del campo descrittore del messaggio.

Se un'applicazione MQI desidera identificare una proprietà del messaggio contenuta in un campo del descrittore del messaggio, ad esempio, in una stringa del selettore o utilizzando le API della proprietà del messaggio, utilizzare la seguente sintassi:

Nome proprietà	**campo Descrittore messaggio**
Root.MQMD.*Campo*	*Campo*

Specificare *Field* con lo stesso caso dei campi della struttura MQMD nella dichiarazione del linguaggio C. Ad esempio, il nome proprietà `Root.MQMD.AccountingToken` accede al campo `AccountingToken` del descrittore del messaggio.

I campi `StrucId` e `Version` del descrittore del messaggio non sono accessibili utilizzando la sintassi visualizzata.

I campi del descrittore del messaggio non vengono mai rappresentati in un'intestazione di MQRFH2 come per altre proprietà.

Se i dati del messaggio iniziano con un MQMDE rispettato dal gestore code, è possibile accedere ai campi MQMDE utilizzando la notazione `Root.MQMD.`*`Field`* descritta. In questo caso, i campi MQMDE vengono trattati come parte logicamente di MQMD dalla prospettiva delle proprietà. Consultare Panoramica di MQMDE.

Valori e tipi di dati della proprietà

Una proprietà può essere un valore booleano, una stringa di byte, una stringa di caratteri o un numero intero o a virgola mobile. La proprietà può memorizzare qualsiasi valore valido nell'intervallo del tipo di dati, a meno che non sia altrimenti limitato dal contesto.

Il tipo di dati di un valore di proprietà deve essere uno dei seguenti:

- MQBOOL
- MQBYTE []
- MQCHAR []
- MQFLOAT32
- MQFLOAT64
- MQINT8
- MQINT16
- MQINT32
- MQINT64

Una proprietà può esistere ma non ha un valore definito; è una proprietà null. Una proprietà null è diversa da una proprietà byte (MQBYTE []) o da una proprietà stringa di caratteri (MQCHAR []) in quanto ha un valore definito ma vuoto, ossia uno con un valore di lunghezza zero.

La stringa di byte non è un tipo di dati di proprietà valido in JMS o XMS. Si consiglia di non utilizzare le proprietà della stringa di byte nella cartella *usr* .

Selezione dei messaggi dalle code

È possibile selezionare i messaggi dalle code utilizzando i campi MsgId e CorrelId su una chiamata MQGET oppure utilizzando una SelectionString su una chiamata MQOPEN o MQSUB.

Selettori

Un selettore messaggi è una stringa a lunghezza variabile utilizzata da un'applicazione per registrare il suo interesse solo per quei messaggi che hanno proprietà che soddisfano la query SQL (Structured Query Language) rappresentata dalla stringa di selezione.

Selezione utilizzando le chiamate di funzione MQSUB e MQOPEN

Si utilizza il *SelectionString*, che è una struttura di tipo MQCHARV, per effettuare selezioni utilizzando le chiamate MQSUB e MQOPEN.

La struttura *SelectionString* viene utilizzata per passare una stringa di selezione a lunghezza variabile al gestore code.

Il CCSID associato alla stringa del selettore viene impostato tramite il campo VSCCSID della struttura MQCHARV. Il valore utilizzato deve essere un CCSID supportato per le stringhe selettore. Consultare Conversione codepage per un elenco di codepage supportate.

Se si specifica un CCSID per cui non è supportata alcuna conversione Unicode di IBM MQ , si verifica un errore di MQRC_SOURCE_CCSID_ERROR. Questo errore viene restituito nel momento in cui il selettore viene presentato al gestore code, ossia nella chiamata MQSUB, MQOPEN o MQPUT1 .

Il valore predefinito per il campo *VSCCSID* è MQCCSI_APPL, che indica che il CCSID della stringa di selezione è uguale al CCSID del gestore code o al CCSID del client se è connesso tramite un client. La costante MQCCSI_APPL può tuttavia essere sovrascritta da un'applicazione che la ridefinisce prima della compilazione.

Se il selettore MQCHARV rappresenta una stringa NULL, non viene effettuata alcuna selezione per tale consumatore di messaggi e i messaggi vengono consegnati come se un selettore non fosse stato utilizzato.

La lunghezza massima di una stringa di selezione è limitata solo da quanto può essere descritto dal campo MQCHARV *VSLength*.

La SelectionString viene restituita sull'output da una chiamata MQSUB utilizzando l'opzione di sottoscrizione MQSO_RESUME, se è stato fornito un buffer ed è presente una lunghezza buffer positiva in VSBufSize. Se non si fornisce un buffer, viene restituita solo la lunghezza della stringa di selezione nel campo VSLength di MQCHARV. Se il buffer fornito è inferiore allo spazio richiesto per restituire il campo, nel buffer fornito vengono restituiti solo VSBufSize byte.

Un'applicazione non può modificare una stringa di selezione senza prima chiudere l'handle alla coda (per MQOPEN) o la sottoscrizione (per MQSUB). Una nuova stringa di selezione può essere specificata in una chiamata MQOPEN o MQSUB successiva.

MQOPEN
: Utilizzare MQCLOSE per chiudere l'handle aperto, quindi specificare una nuova stringa di selezione su una chiamata MQOPEN successiva.

MQSUB
: Utilizzare MQCLOSE per chiudere l'handle di sottoscrizione restituito (hSub), quindi specificare una nuova stringa di selezione su una chiamata MQSUB successiva.

Figura 3 a pagina 33 mostra il processo di selezione utilizzando la chiamata MQSUB.

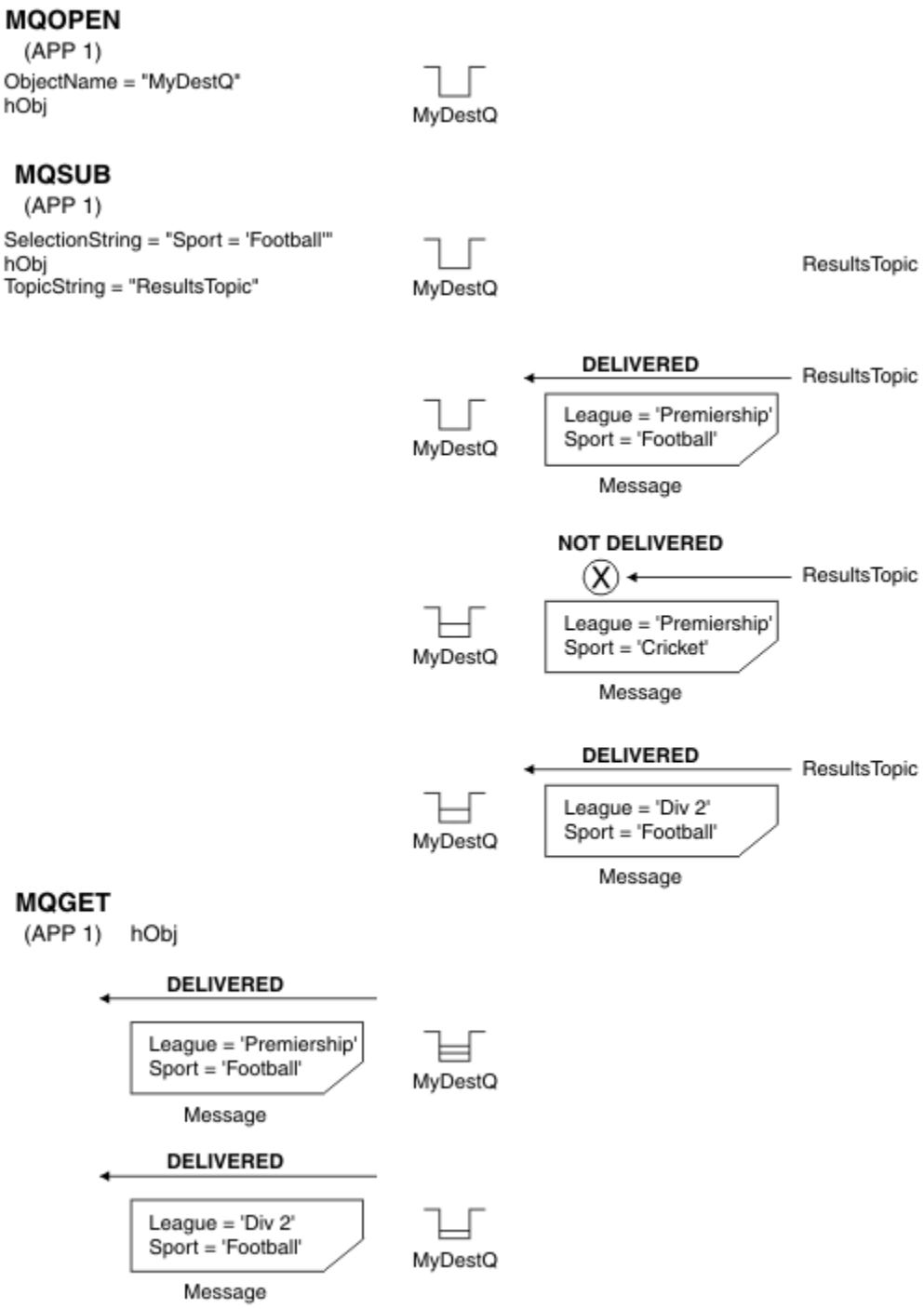


Figura 3. Selezione mediante chiamata MQSUB

È possibile passare un selettore sulla chiamata a MQSUB utilizzando il campo *SelectionString* nella struttura MQSD. L'effetto del passaggio in un selettore su MQSUB è che solo i messaggi pubblicati sull'argomento sottoscritto, che corrispondono a una stringa di selezione fornita, vengono resi disponibili sulla coda di destinazione.

Figura 4 a pagina 34 mostra il processo di selezione utilizzando la chiamata MQOPEN.

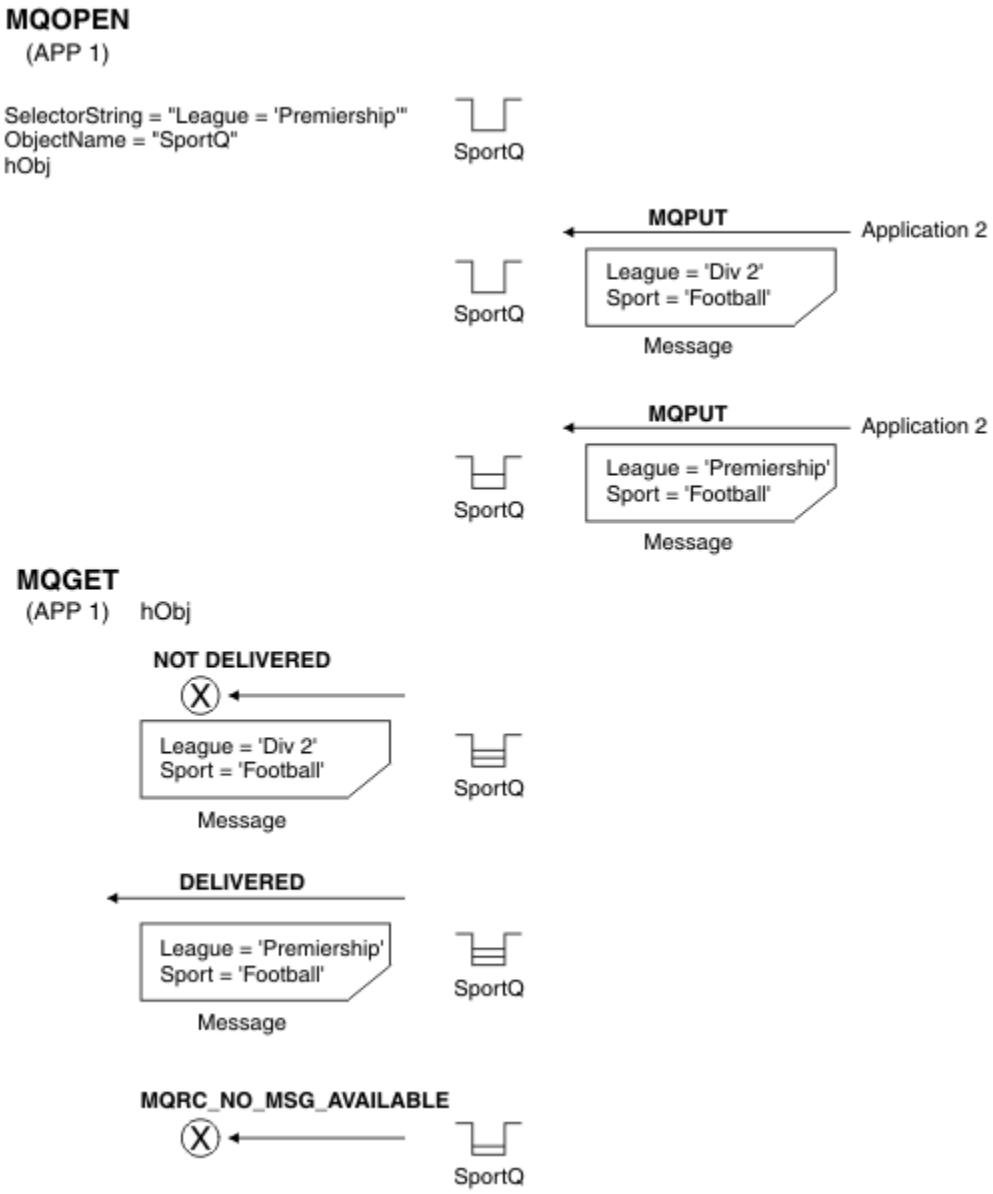


Figura 4. Selezione mediante chiamata MQOPEN

È possibile passare un selettore sulla chiamata a MQOPEN utilizzando il campo *SelectionString* nella struttura MQOD. L'effetto del passaggio in un selettore sulla chiamata MQOPEN è che solo i messaggi sulla coda aperta, che corrispondono a un selettore, vengono consegnati al consumatore di messaggi.

L'utilizzo principale per il selettore nella chiamata MQOPEN è per il caso point - to - point in cui un'applicazione può decidere di ricevere solo quei messaggi su una coda che corrispondono a un selettore. L'esempio precedente mostra uno scenario semplice in cui due messaggi vengono inseriti in una coda aperta da MQOPEN, ma solo uno viene ricevuto dall'applicazione che lo riceve, poiché è l'unico che corrisponde a un selettore.

Notare che le chiamate MQGET successive risultano in MQRC_NO_MSG_AVAILABLE poiché non esistono ulteriori messaggi sulla coda che corrispondono al selettore fornito.

Concetti correlati

“Regole e limitazioni della stringa di selezione” a pagina 41
Familiarizzare con queste regole su come vengono interpretate le stringhe di selezione e le limitazioni di caratteri per evitare potenziali problemi quando si utilizzano i selettori.

Comportamento selezione

Panoramica del comportamento di selezione IBM MQ .

I campi in una struttura MQMDE vengono considerati come le proprietà del messaggio per le proprietà del descrittore del messaggio corrispondenti se MQMD:

- Ha formato MQFMT_MD_EXTENSION
- È immediatamente seguito da una struttura MQMDE valida
- È la versione uno o contiene solo i campi della versione due predefiniti

È possibile che una stringa di selezione si risolva in TRUE o FALSE prima che si verifichi qualsiasi corrispondenza con le proprietà del messaggio. Ad esempio, potrebbe essere il caso se la stringa di selezione è impostata su `"TRUE <> FALSE"`. Tale valutazione iniziale è garantita solo quando non vi sono riferimenti di proprietà del messaggio nella stringa di selezione.

Se una stringa di selezione si risolve in TRUE prima che vengano considerate le proprietà del messaggio, vengono consegnati tutti i messaggi pubblicati nell'argomento sottoscritto dal consumer. Se una stringa di selezione si risolve in FALSE prima che vengano prese in considerazione le proprietà del messaggio, viene restituito un codice motivo di MQRC_SELECTOR_ALWAYS_FALSE e il codice di completamento MQCC_FAILED nella chiamata della funzione che ha presentato il selettore.

Anche se un messaggio non contiene proprietà del messaggio (diverse dalle proprietà dell'intestazione), può essere ancora idoneo per la selezione. Se una stringa di selezione fa riferimento a una proprietà del messaggio che non esiste, si presume che questa proprietà abbia il valore NULL o 'Sconosciuto'.

Ad esempio, un messaggio potrebbe ancora soddisfare una stringa di selezione come `'Color IS NULL'`, dove `'Color'` non esiste come proprietà del messaggio nel messaggio.

La scelta può essere eseguita solo sulle proprietà associate a un messaggio, non sul messaggio stesso, a meno che non sia disponibile un provider di selezione messaggi esteso. La selezione può essere eseguita sul payload del messaggio solo se è disponibile un provider di selezione del messaggio esteso.

Ad ogni proprietà del messaggio è associato un tipo. Quando si esegue una selezione, è necessario verificare che i valori utilizzati nelle espressioni per verificare le proprietà del messaggio siano del tipo corretto. Se si verifica una mancata corrispondenza del tipo, l'espressione in questione si risolve in FALSE.

È responsabilità dell'utente assicurarsi che la stringa di selezione e le proprietà del messaggio utilizzino tipi compatibili.

I criteri di selezione continuano ad essere applicati per conto dei sottoscrittori durevoli inattivi, in modo che vengano conservati solo i messaggi che corrispondono alla stringa di selezione originariamente fornita.

Le stringhe di selezione non sono modificabili quando una sottoscrizione durevole viene ripresa con alter (MQSO_ALTER). Se viene presentata una stringa di selezione diversa quando un sottoscrittore durevole riprende l'attività, allora MQRC_SELECTOR_NOT_ALTERABLE viene restituito all'applicazione.

Le applicazioni ricevono un codice di ritorno di MQRC_NO_MSG_AVAILABLE se non è presente alcun messaggio su una coda che soddisfa i criteri di selezione.

Se un'applicazione ha specificato una stringa di selezione contenente valori di proprietà, solo i messaggi che contengono proprietà corrispondenti sono idonei per la selezione. Ad esempio, un sottoscrittore specifica una stringa di selezione "a = 3" e viene pubblicato un messaggio che non contiene proprietà o proprietà in cui 'a' non esiste o non è uguale a 3. Il sottoscrittore non riceve il messaggio nella coda di destinazione.

Prestazioni di messaggistica

La selezione dei messaggi da una coda richiede IBM MQ per ispezionare in modo sequenziale ogni messaggio sulla coda. I messaggi vengono ispezionati fino a quando non viene trovato un messaggio che corrisponde ai criteri di selezione o non ci sono più messaggi da esaminare. Pertanto, le prestazioni della messaggistica subiscono un peggioramento se la selezione dei messaggi viene utilizzata su code profonde.

Per ottimizzare la selezione dei messaggi su code profonde quando la selezione è basata su JMSCorrelationID o JMSMessageID, utilizzare una stringa di selezione del formato:

- JMSCorrelationID = 'ID:*id_correlazione*'
- JMSMessageID= 'ID:*id_messaggio*'

dove:

- *correlation_id* è una stringa contenente un identificativo di correlazione IBM MQ standard.
- *message_id* è una stringa contenente un identificativo di messaggio IBM MQ standard.

Nota: Il selettore deve fare riferimento solo a una delle proprietà. L'utilizzo di un selettore che ha uno di questi formati offre un miglioramento significativo delle prestazioni quando si seleziona JMSCorrelationID e offre un miglioramento delle prestazioni marginale per JMSMessageID. Per ulteriori informazioni, consultare "Selettori di messaggi in JMS" a pagina 147.

Utilizzo di selettori complessi

I selettori possono contenere molti componenti, ad esempio:

```
a e b o c e d o e e f o g e h o i e j... o y e z
```

L'utilizzo di tali selettori complessi può avere gravi implicazioni sulle prestazioni e requisiti di risorse eccessivi. Come tale, IBM MQ proteggerà il sistema non riuscendo a elaborare selettori eccessivamente complessi che potrebbero causare una carenza di risorse di sistema. La protezione può verificarsi su stringhe di selezione che contengono più di 100 test oppure quando IBM MQ rileva che si sta avvicinando il limite sulla dimensione dello stack del sistema operativo. È necessario provare a fondo e testare l'utilizzo delle stringhe di selezione con molti componenti, sulle piattaforme appropriate, per garantire che i limiti di protezione non vengano raggiunti.

Le prestazioni e complessità dei selettori possono essere migliorate semplificandoli utilizzando ulteriori parentesi per combinare i componenti. Ad esempio:

```
( a e b o c e d) o ( e e f o g e h) o ( i e j) ...
```

Concetti correlati

"Regole e limitazioni della stringa di selezione" a pagina 41
Familiarizzare con queste regole su come vengono interpretate le stringhe di selezione e le limitazioni di caratteri per evitare potenziali problemi quando si utilizzano i selettori.

Sintassi del selettore messaggi

Un selettore di messaggi IBM MQ è una stringa con sintassi basata su un sottoinsieme della sintassi dell'espressione condizionale SQL92 .

L'ordine in cui viene valutato un selettore di messaggi è da sinistra a destra all'interno di un livello di precedenza. È possibile utilizzare le parentesi per modificare questo ordine. I valori letterali selettori predefiniti e i nomi degli operatori vengono scritti qui in maiuscolo; tuttavia, non sono sensibili al maiuscolo / minuscolo.

Se il selettore viene fornito tramite l'API, IBM MQ verifica la correttezza sintattica di un selettore di messaggi nel momento in cui viene presentato. Se la sintassi della stringa di selezione non è corretta o se il nome di una proprietà non è valido e non è disponibile un provider di selezione dei messaggi estesi, `MQRC_SELECTION_NOT_AVAILABLE` viene restituito all'applicazione. Se la sintassi della stringa di selezione non è corretta o il nome di una proprietà non è valido quando viene ripresa una sottoscrizione, viene restituito un `MQRC_SELECTOR_SYNTAX_ERROR` all'applicazione. Se la convalida del nome della proprietà è stata disabilitata quando la proprietà è stata impostata (impostando `MQCMHO_NONE` invece di `MQCMHO_VALIDATE`) e un'applicazione successivamente inserisce un messaggio con un nome proprietà non valido, questo messaggio non viene mai selezionato.

Non viene restituito alcun errore nel momento in cui il selettore viene presentato se IBM MQ determina che un selettore di sottoscrizione definito amministrativamente utilizza la sintassi del messaggio esteso, come indicato dal parametro **DISPLAY SUB SELTYPE** con il valore EXTENDED. In questo caso, la

verifica della sintassi della stringa di selezione viene rimandata fino all'ora di pubblicazione (consultare MQRC_SELECTION_NOT_AVAILABLE).

Un selettore può contenere:

- Valori letterali:
  - I valori letterali stringa sono racchiusi tra virgolette singole. Due virgolette singole consecutive rappresentano una virgoletta singola. Gli esempi sono 'literal' e 'literal' '. Come i letterali della stringa Java , questi utilizzano la codifica dei caratteri Unicode. Non è possibile utilizzare le virgolette doppie per racchiudere un letterale stringa. Qualsiasi sequenza di byte può essere utilizzata tra virgolette singole.
  - Una stringa di byte è una o più coppie di caratteri esadecimali racchiusi tra virgolette doppie e preceduti da 0x. Gli esempi sono "0x2F1C" o "0XD43A". La lunghezza di una stringa di byte deve essere almeno un byte. Se una stringa di byte del selettore corrisponde a una proprietà del messaggio di tipo MQTYPE_BYTE_STRING, non viene eseguita alcuna azione speciale sullo zero iniziale o finale. I byte vengono trattati come un altro carattere. Anche l'endianità non è considerata. La lunghezza delle stringhe di byte del selettore e della proprietà deve essere uguale e la sequenza di byte deve essere la stessa.

    Esempi di selezioni di stringhe di byte (si supponga che *myBytes* = 0AFC23) che corrispondono sono:

    - "myBytes = "0x0AFC23"" = TRUE

    Le seguenti selezioni di stringa non corrispondono:

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (perché il numero di byte non è multiplo di due)
    - "myBytes = "0x0AFC2300"" = FALSE (perché lo zero finale è significativo nel confronto)
    - "myBytes = "0x000AFC23"" = FALSE (perché lo zero iniziale è significativo nel confronto)
    - "myBytes = "0x23FC0A"" = FALSE (perché l'endianità non viene considerata)
  - I numeri esadecimali iniziano con uno zero, seguiti da un xmaiuscolo o minuscolo. Il resto della costante letterale contiene uno o più caratteri esadecimali validi. Esempi sono 0xA, 0xAF, 0X2020.
  - Uno zero iniziale seguito da una o più cifre nell'intervallo 0 - 7 viene sempre interpretato come l'inizio di un numero ottale. Non è possibile rappresentare un numero decimale con prefisso zero come questo, ad esempio, 09 restituisce un errore di sintassi perché 9 non è una cifra ottale valida. Esempi di numeri ottali sono 0177, 0713.
  - Una costante letterale numerica esatta è un valore numerico senza un punto decimale, ad esempio 57, -957e +62. Una costante letterale numerica esatta può avere una L maiuscola o minuscola finale; ciò non influisce sul modo in cui il numero viene memorizzato o interpretato. IBM MQ supporta i numeri esatti nell'intervallo -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.
  - Un valore letterale numerico approssimativo è un valore numerico in notazione scientifica, ad esempio 7E3 o -57.9E2, oppure un valore numerico con un decimale, ad esempio 7., -95.7o +6.2. IBM MQ supporta i numeri compresi tra -1.797693134862315E+308 e 1.797693134862315E+308.

    Il significando deve seguire un segno facoltativo (+ o -). Il significando deve essere un numero intero o una frazione. Una parte frazionaria del significando non deve necessariamente avere una cifra iniziale.

    Un E maiuscolo o minuscolo indica l'inizio di un esponente facoltativo. L'esponente ha una radice decimale e la parte numerica dell'esponente può essere preceduta da un segno facoltativo.

    Le costanti letterali numeriche approssimate possono terminare con un carattere F o D (non sensibile al maiuscolo / minuscolo). Questa sintassi esiste per supportare il metodo cross - language di tagging di numeri di precisione singoli o doppi. Questi caratteri sono facoltativi e non influiscono sul modo in cui una costante letterale numerica approssimativa viene memorizzata o elaborata. Questi numeri vengono sempre memorizzati ed elaborati utilizzando la doppia precisione.
  - I letterali booleani TRUE e FALSE.

Nota: Le rappresentazioni IEEE-754 non finite come `NaN`, `+Infinity`, `-Infinity` non sono supportate nelle stringhe di selezione. Non è quindi possibile utilizzare questi valori come operandi in un'espressione. Lo zero negativo viene trattato come lo zero positivo per le operazioni matematiche.

- Identificativi:

  Un identificativo è una sequenza di caratteri a lunghezza variabile che deve iniziare con un carattere di inizio identificativo valido, seguito da zero o più caratteri di parte identificativo validi. Le regole per i nomi degli identificativi sono le stesse per i nomi delle proprietà dei messaggi, consultare "Nomi proprietà" a pagina 28 e "Limitazioni nome proprietà" a pagina 29 per ulteriori informazioni.

  Nota: La selezione può essere eseguita sul payload del messaggio solo se è disponibile un provider di selezione del messaggio esteso.

  Gli identificatori sono riferimenti di campi di intestazione o riferimenti di proprietà. Il tipo di valore di una proprietà in un selettore di messaggi deve corrispondere al tipo utilizzato per impostare la proprietà, anche se la promozione numerica viene eseguita dove possibile. Se si verifica una mancata corrispondenza del tipo, il risultato dell'espressione è FALSE. Se si fa riferimento a una proprietà che non esiste in un messaggio, il suo valore è `NULL`.

  Le conversioni di tipo che si applicano ai metodi get per le proprietà non si applicano quando una proprietà viene utilizzata in un'espressione del selettore messaggi. Ad esempio, se si imposta una proprietà come valore stringa e si utilizza un selettore per eseguire la query come valore numerico, l'espressione restituisce FALSE.

  I nomi campo e proprietà JMS associati ai nomi proprietà o ai nomi campo `MQMD` sono anch'essi identificatori validi in una stringa di selezione. IBM MQ associa i nomi campo e proprietà JMS riconosciuti ai valori delle proprietà del messaggio. Per ulteriori informazioni, fare riferimento a "Selettori di messaggi in JMS" a pagina 147. Ad esempio, la stringa di selezione `"JMSPriority >="` viene selezionata nella proprietà `Pri` della cartella `jms` del messaggio corrente.

- Eccedenza / insufficienza:

  Per i numeri decimali e approssimativi, le seguenti condizioni non sono definite:

  - Specifica di un numero non compreso nell'intervallo definito
  - Specifica di un'espressione aritmetica che causerebbe un overflow o un underflow

  Non vengono eseguite verifiche per queste condizioni.

- Spazio vuoto:

  Definito come spazio, avanzamento pagina, nuova riga, ritorno a capo, tabulazione orizzontale o tabulazione verticale. I seguenti caratteri Unicode sono riconosciuti come spazi vuoti:

  - `\u0009 to \u000D`
  - `\u0020`
  - `\u001C`
  - `\u001D`
  - `\u001E`
  - `\u001F`
  - `\u1680`
  - `\u180E`
  - Da `\u2000` a `\u200A`
  - `\u2028`
  - `\u2029`
  - `\u202F`
  - `\u205F`
  - `\u3000`

- Espressioni:

  - Un selettore è un'espressione condizionale. Un selettore che assume il valore di true corrisponde; un selettore che assume il valore di false o unknown non corrisponde.
  - Le espressioni aritmetiche sono costituite da se stesse, da operazioni aritmetiche, da identificativi (il valore dell'identificativo viene trattato come una costante letterale numerica) e da costanti letterali numeriche.
  - Le espressioni condizionali sono composte da se stesse, operazioni di confronto e operazioni logiche.
- La parentesi standard (), per impostare l'ordine in cui vengono valutate le espressioni, è supportata.
- Operatori logici in ordine di precedenza: `NOT`, `AND`, `OR`.
- Operatori di confronto: =, >, >=, <, <=, <> (non uguale).
  - Due stringhe di byte sono uguali solo se le stringhe sono della stessa lunghezza e la sequenza di byte è uguale.
  - È possibile confrontare solo i valori dello stesso tipo. Un'eccezione è che è valido per confrontare i valori numerici esatti e i valori numerici approssimativi (la conversione del tipo richiesta è definita dalle regole della promozione numerica Java). Se si tenta di confrontare tipi diversi, il selettore è sempre false.
  - La stringa e il confronto booleano sono limitati a = e <>. Due stringhe sono uguali solo se contengono la stessa sequenza di caratteri.
- Operatori aritmetici in ordine di precedenza:
  - +, - unario.
  - * moltiplicazione e / divisione.
  - + addizione e - sottrazione.
  - Le operazioni aritmetiche su un valore NULL non sono supportate. Se vengono tentati, il selettore completo è sempre false.
  - Le operazioni aritmetiche devono utilizzare la promozione numerica Java .
- Operatore di confronto arithmetic-expr1 [`NOT`] `BETWEEN` arithmetic-expr2 e arithmetic-expr3 :
  - `Age BETWEEN 15 and 19` è equivalente a `age>= 15 AND age <= 19`.
  - `Age NOT BETWEEN 15 and 19` è equivalente a `age < 15 OR age > 19`.
  - Se una qualsiasi delle espressioni di un'operazione `BETWEEN` è NULL, il valore dell'operazione è false. Se una delle espressioni di un'operazione `NOT BETWEEN` è `NULL`, il valore dell'operazione è true.
- operatore di confronto identificativo `[NOT] IN (string-literal1, string-literal2,...)` dove l'identificativo ha un valore Stringa o `NULL` .
  - `Country IN ('UK', 'US', 'France')` è true per `'UK'` e false per `'Peru'`. Equivale all'espressione `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` è false per `'UK'` e true per `'Peru'`. Equivale all'espressione `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Se l'identificativo di un'operazione `IN` o `NOT IN` è NULL, il valore dell'operazione è sconosciuto.
- `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` operatore di confronto, dove `identifier` ha un valore stringa. *pattern - value* è una stringa letterale, dove _ sta per qualsiasi carattere singolo e % sta per qualsiasi sequenza di caratteri (inclusa la sequenza vuota). Tutti gli altri personaggi rappresentano se stessi. Il *carattere di escape* facoltativo è un valore letterale stringa di caratteri singolo utilizzato per eseguire l'escape del significato speciale di _ e % in *valore - modello*. L'operatore `LIKE` deve essere utilizzato solo per confrontare due valori stringa.
  - `phone LIKE '12%3'` è true per 123 e 12993 e false per 1234.
  - `word LIKE 'l_se'` è true per la perdita e false per la perdita.
  - `underscored LIKE '\_%' ESCAPE '\'` è true per `_foo` e false per `bar`.
  - `phone NOT LIKE '12%3'` è false per 123 e 12993 e true per 1234.

– Se l'identificativo di un'operazione LIKE o NOT LIKE è NULL, il valore dell'operazione è sconosciuto.

Nota: L'operatore LIKE deve essere utilizzato per confrontare due valori stringa. Il valore di Root.MQMD.CorrelId è una schiera di byte a 24 byte, non una stringa di caratteri. La stringa del selettore Root.MQMD.CorrelId LIKE 'ABC%' viene accettata dal parser come sintatticamente valida, ma viene valutata come false. Quando si confronta un array di byte con una stringa di caratteri, LIKE non può essere utilizzato.

- test dell'operatore di confronto identifier IS NULL per un valore del campo di intestazione NULL o per un valore della proprietà mancante.
- L'operatore di confronto identifier IS NOT NULL verifica l'esistenza di un valore di campo di intestazione non null o di un valore di proprietà.
- Valori null

  La valutazione delle espressioni del selettore che contengono valori NULL è definita dalla semantica SQL 92 NULL , in sintesi:

  – SQL considera un valore NULL come sconosciuto.
  – Il confronto o l'aritmetica con un valore sconosciuto produce sempre un valore sconosciuto.
  – Gli operatori IS NULL e IS NOT NULL convertono un valore sconosciuto in valori TRUE e FALSE .

  Gli operatori booleani utilizzano la logica a tre valori (T=TRUE, F=FALSE, U=UNKNOWN)

Tabella 1. Valore del risultato dell'operatore booleano quando la logica è A AND B

Operatore A	**Operatore B**	**Risultato (A E B)**
T	F	F
T	U	U
T	T	T
F	T	F
F	U	F
F	F	F
U	T	U
U	U	U
U	F	F

Tabella 2. Valore del risultato dell'operatore booleano quando la logica è A OR B

Operatore A	**Operatore B**	**Risultato (A OR B)**
T	F	T
T	U	T
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

Tabella 3. Valore del risultato dell'operatore booleano quando la logica è NOT A	
Operatore A	**Risultato (NOT A)**
T	F
F	T
U	U

Il seguente selettore di messaggi seleziona i messaggi con un tipo di messaggio di auto, un colore di blu e un peso maggiore di 2500 libbre:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

Sebbene SQL supporti il confronto decimale fisso e l'aritmetica, i selettori dei messaggi non lo fanno. Questo è il motivo per cui le costanti letterali numeriche esatte sono limitate a quelle senza un decimale. È anche il motivo per cui ci sono numeri con un decimale come rappresentazione alternativa per un valore numerico approssimativo.

I commenti SQL non sono supportati.

Concetti correlati

“Proprietà dei messaggi” a pagina 27
Utilizzare le proprietà del messaggio per consentire a un'applicazione di selezionare i messaggi da elaborare o per richiamare le informazioni su un messaggio senza accedere alle intestazioni MQMD o MQRFH2 . Inoltre, facilitano la comunicazione tra applicazioni IBM MQ e JMS .

Riferimenti correlati

MsgHandle

MQBUFMH - Converti buffer in handle del messaggio

Regole e limitazioni della stringa di selezione

Familiarizzare con queste regole su come vengono interpretate le stringhe di selezione e le limitazioni di caratteri per evitare potenziali problemi quando si utilizzano i selettori.

- La selezione del messaggio per la messaggistica di pubblicazione / sottoscrizione si verifica sul messaggio come inviato dal publisher. Consultare la sezione Stringhe di selezione.
- L'equivalenza viene verificata utilizzando un singolo carattere uguale; ad esempio, a `=` b è corretto, mentre a `==` b è errato.
- Un operatore utilizzato da molti linguaggi di programmazione per rappresentare 'non uguale a' è `!=`. Questa rappresentazione non è un sinonimo valido per <> ; ad esempio, a `<>` b è valido, mentre a `!=` b non è valido.
- Le virgolette singole vengono riconosciute solo se il ' Viene utilizzato il carattereU+0027. Allo stesso modo, le virgolette doppie, valide solo quando utilizzate per racchiudere stringhe di byte, devono utilizzare il carattere " (U+0022).
- I simboli &, &&, | e || non sono sinonimi per congiunzione / disgiunzione logica; ad esempio, a `&&` b deve essere specificato come a `AND` b.
- I caratteri jolly * e ? non sono sinonimi per % e _.
- I selettori contenenti espressioni composte come `20 < b < 30` non sono validi. Il programma di analisi valuta gli operatori che hanno la stessa precedenza da sinistra a destra. L'esempio diventa quindi `(20 < b) < 30`, che non ha senso. Invece, l'espressione deve essere scritta come `(b > 20) AND (b < 30)`.
- Le stringhe di byte devono essere racchiuse tra virgolette doppie; se vengono utilizzate virgolette singole, la stringa di byte viene considerata una stringa letterale. Il numero di caratteri (non il numero che i caratteri rappresentano) che seguono `0x` deve essere un multiplo di due.

- La parola chiave IS non è un sinonimo del carattere uguale. Pertanto, le stringhe di selezione a `IS 3` e b `IS 'red'` non sono valide. La parola chiave IS esiste solo per supportare i casi `IS NULL` e `IS NOT NULL` .

Concetti correlati

"Comportamento selezione" a pagina 35
Panoramica del comportamento di selezione IBM MQ .

Riferimenti correlati

Stringhe di selezione

Considerazioni su UTF-8 e Unicode quando si utilizzano i selettori di messaggi

I caratteri, non racchiusi tra virgolette singole, che costituiscono le parole chiave riservate di una stringa di selezione devono essere immessi in Basic Latin Unicode (che va dal carattere U+0000 a U+0007F). Non è valido utilizzare altre rappresentazioni di punti di codice di caratteri alfanumerici. Ad esempio, il numero 1 deve essere espresso come U+0031 in Unicode, non è valido per utilizzare l'equivalente cifra a larghezza intera U+FF11 o l'equivalente arabo U+0661.

I nomi delle proprietà del messaggio possono essere specificati utilizzando qualsiasi sequenza valida di caratteri Unicode. I nomi delle proprietà dei messaggi contenuti nelle stringhe di selezione codificati in UTF-8 verranno convalidati anche se contengono caratteri multi - byte. La convalida di UTF-8 a più byte è rigorosa ed è necessario assicurarsi che per i nomi delle proprietà del messaggio siano utilizzate sequenze UTF-8 valide. I caratteri oltre il piano Unicode Basic Multilingual Plane (quelli superiori a U + FFFF), rappresentati in UTF-16 da punti di codice surrogati (da X'D800'a X'DFFF'), o quattro byte in UTF-8, non sono supportati nei nomi delle proprietà dei messaggi.

Non viene eseguita alcuna elaborazione supplementare sui valori o sui nomi delle proprietà durante il confronto per l'uguaglianza. Ciò significa, ad esempio, che non avviene alcuna pre / de - composizione e alle legature non viene dato alcun significato speciale. Ad esempio, il carattere umlaut precomposto U+00FC non è considerato equivalente a U+0075 + U+0308 e la sequenza di caratteri ff non è considerata equivalente a Unicode U+FB00 (LATIN SMALL LIGATURE FF)

I dati della proprietà racchiusi tra virgolette singole possono essere rappresentati da qualsiasi sequenza di byte e non vengono convalidati.

Selezione del contenuto di un messaggio

È possibile effettuare la sottoscrizione in base a una selezione del contenuto del payload del messaggio (noto anche come filtro del contenuto), ma la decisione su quali messaggi devono essere consegnati a tale sottoscrizione non può essere eseguita direttamente da IBM MQ; invece, è necessario un provider di selezione dei messaggi esteso, ad esempio IBM Integration Bus, per elaborare i messaggi.

Quando un'applicazione pubblica una stringa di argomenti, dove uno o più sottoscrittori hanno una stringa di selezione selezionata sul contenuto del messaggio, IBM MQ richiederà che il provider di selezione dei messaggi estesi analizzi la pubblicazione e informi IBM MQ se la pubblicazione corrisponde ai criteri di selezione specificati da ciascun sottoscrittore con un filtro del contenuto.

Se il provider di selezione del messaggio esteso determina che la pubblicazione corrisponde alla stringa di selezione del sottoscrittore, il messaggio continuerà ad essere consegnato al sottoscrittore.

Se il provider di selezione dei messaggi estesi determina che la pubblicazione non corrisponde, il messaggio non viene consegnato al sottoscrittore. Ciò potrebbe causare l'esito negativo della chiamata MQPUT o MQPUT1 con codice motivo MQRC_PUBLICATION_FAILURE. Se il provider di selezione dei messaggi estesi non è in grado di analizzare la pubblicazione, viene restituito il codice motivo MQRC_CONTENT_ERROR e la chiamata MQPUT o MQPUT1 ha esito negativo.

Se il provider di selezione dei messaggi estesi non è disponibile o non è in grado di determinare se il sottoscrittore (subscriber) deve ricevere la pubblicazione, viene restituito il codice motivo MQRC_SELECTION_NOT_AVAILABLE e la chiamata MQPUT o MQPUT1 ha esito negativo.

Quando una sottoscrizione viene creata con un filtro del contenuto e il provider di selezione dei messaggi estesi non è disponibile, la chiamata MQSUB ha esito negativo con codice di errore MQRC_SELECTION_NOT_AVAILABLE. Se viene ripresa una sottoscrizione con un filtro del contenuto e

il provider di selezione dei messaggi estesi non è disponibile, la chiamata MQSUB restituisce un avviso di MQRC_SELECTION_NOT_AVAILABLE, ma la sottoscrizione può essere ripresa.

Riferimenti correlati

Stringhe di selezione

Utilizzo asincrono dei messaggi IBM MQ

L'utilizzo asincrono utilizza una serie di estensioni MQI (Message Queue Interface), MQI richiama MQCB e MQCTL, che consentono a un'applicazione MQI di essere scritta per utilizzare i messaggi da una serie di code. I messaggi vengono consegnati all'applicazione richiamando una 'unità di codice', identificata dall'applicazione che trasmette il messaggio o un token che rappresenta il messaggio.

Negli ambienti applicativi più semplici, l'unità di codice è definita da un puntatore di funzione, tuttavia in altri ambienti l'unità di codice può essere definita da un nome di programma o di modulo.

Nell'utilizzo asincrono dei messaggi, vengono utilizzati i termini seguenti:

consumatore di messaggi
: Un costrutto di programmazione che consente di definire un programma o una funzione da richiamare con un messaggio quando ne diventa disponibile uno che corrisponde al requisito delle applicazioni.

Gestore eventi
: Un costrutto di programmazione che consente di definire un programma o una funzione da richiamare quando si verifica un evento asincrono, come la sospensione del gestore code.

Callback
: Un termine generico utilizzato per fare riferimento a una routine Message Consumer o Event Handler.

Il consumo asincrono può semplificare la progettazione e l'implementazione di nuove applicazioni, in particolare quelle che elaborano più code di input o sottoscrizioni. Tuttavia, se si sta utilizzando più di una coda di input e si stanno elaborando i messaggi in sequenza di priorità, la sequenza di priorità viene osservata indipendentemente all'interno di ciascuna coda: è possibile che si ricevano messaggi a bassa priorità da una coda prima dei messaggi ad alta priorità da un'altra. L'ordine dei messaggi tra più code non è garantito. Notare inoltre che se si utilizzano le uscite API, potrebbe essere necessario modificarle per includere le chiamate MQCB e MQCTL.

Le seguenti illustrazioni forniscono un esempio di come utilizzare questa funzione.

Figura 5 a pagina 44 mostra un'applicazione a più thread che utilizza i messaggi da due code. L'esempio mostra tutti i messaggi consegnati a una funzione singola.

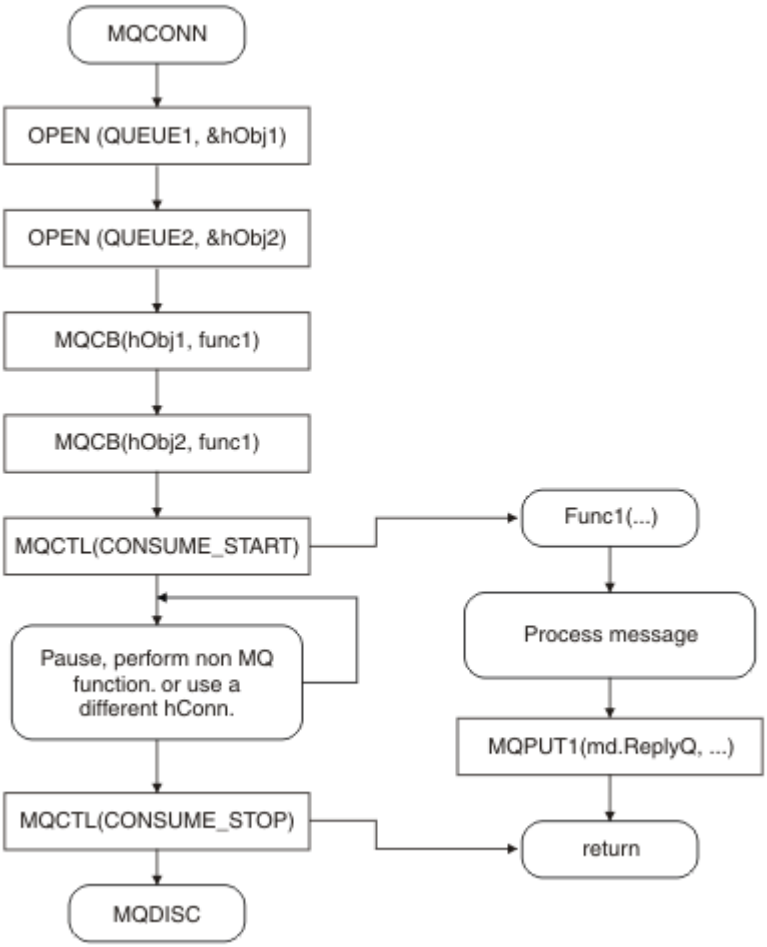


Figura 5. Applicazione basata sui messaggi standard che utilizza due code

z/OS Su z/OS, il thread di controllo principale deve emettere una chiamata MQDISC prima di terminare. Ciò consente ai sottoprocessi di callback di terminare e rilasciare le risorse di sistema.

Figura 6 a pagina 45 Questo flusso di esempio mostra un'applicazione a thread singolo che utilizza messaggi da due code. L'esempio mostra tutti i messaggi consegnati a una funzione singola.

La differenza rispetto al caso asincrono è che il controllo non ritorna all'emittente di MQCTL fino a quando tutti i consumer non si sono disattivati; vale a dire che un consumer ha emesso una richiesta MQCTL STOP o il gestore code è inattivo.

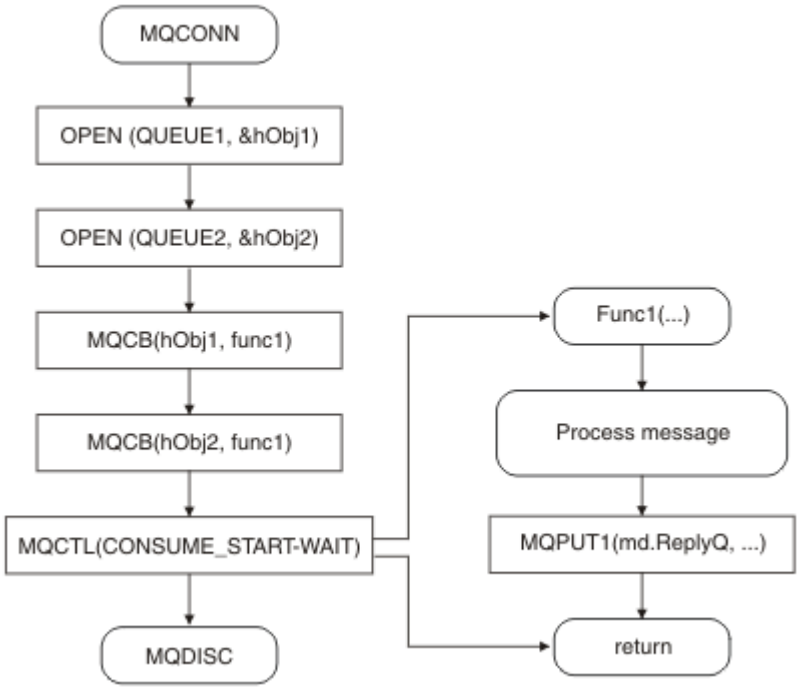


Figura 6. L'applicazione basata sui messaggi a thread singolo utilizza due code

Gruppi di messaggi

I messaggi possono verificarsi all'interno dei gruppi per consentire l'ordinamento dei messaggi.

I gruppi di messaggi consentono di contrassegnare più messaggi come correlati tra loro e un ordine logico da applicare al gruppo (consultare "Ordinamento logico e fisico" a pagina 774). Su Multiplatforms, la segmentazione dei messaggi consente la suddivisione di messaggi di grandi dimensioni in segmenti più piccoli. Non è possibile utilizzare messaggi raggruppati o segmentati durante l'inserimento in un argomento.

La gerarchia all'interno di un gruppo è la seguente:

Gruppo
Questo è il livello più alto nella gerarchia ed è identificato da *GroupId*. Consiste in uno o più messaggi che contengono lo stesso *GroupId*. Questi messaggi possono essere memorizzati ovunque nella coda.

Nota: Il termine *messaggio* viene utilizzato qui per indicare un elemento su una coda, come ad esempio viene restituito da un singolo MQGET che non specifica MQGMO_COMPLETE_MSG.

Figura 7 a pagina 45 mostra un gruppo di messaggi logici:

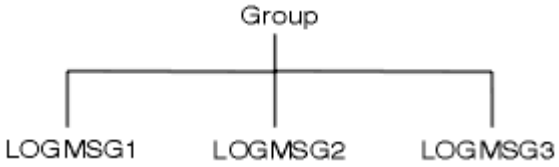


Figura 7. Gruppo di messaggi logici

Aprendo una coda e specificando MQOO_BIND_ON_GROUP, si forzano tutti i messaggi in un gruppo inviati a questa coda ad essere inviati alla stessa istanza della coda. Per ulteriori informazioni sull'opzione BIND_ON_GROUP, consultare Gestione delle affinità dei messaggi.

Messaggio logico

I messaggi logici all'interno di un gruppo vengono identificati dai campi *GroupId* e *MsgSeqNumber* . Il *MsgSeqNumber* inizia da 1 per il primo messaggio all'interno di un gruppo e se un messaggio non si trova in un gruppo, il valore del campo è 1.

Utilizzare messaggi logici all'interno di un gruppo per:

- Verificare l'ordine (se ciò non è garantito nelle circostanze in cui il messaggio viene trasmesso).
- Consentire alle applicazioni di raggruppare messaggi simili (ad esempio, quelli che devono essere elaborati dalla stessa istanza del server).

Ogni messaggio all'interno di un gruppo è costituito da un messaggio fisico, a meno che non sia suddiviso in segmenti. Ogni messaggio è logicamente un messaggio separato e solo i campi *GroupId* e *MsgSeqNumber* in MQMD devono avere una relazione con altri messaggi nel gruppo. Gli altri campi in MQMD sono indipendenti; alcuni potrebbero essere identici per tutti i messaggi nel gruppo, mentre altri potrebbero essere diversi. Ad esempio, i messaggi in un gruppo possono avere nomi di formato, CCSID e codifiche differenti.

Segmento

I segmenti vengono utilizzati per gestire i messaggi che sono troppo grandi per l'applicazione di inserimento o di richiamo o per il gestore code (inclusi i gestori code che intervengono attraverso i quali il messaggio passa). Per ulteriori informazioni, consultare “Segmentazione del messaggio” a pagina 793.

Un singolo messaggio viene suddiviso in messaggi più piccoli denominati *segmenti*. Un segmento di un messaggi è identificato dai campi *GroupId*, *MsgSeqNumber*e *Offset* . Il campo *Offset* inizia da zero per il primo segmento all'interno di un messaggio.

Ogni segmento è costituito da un messaggio fisico che potrebbe appartenere a un gruppo (Figura 8 a pagina 46 mostra un esempio di messaggi all'interno di un gruppo). Un segmento è logicamente parte di un singolo messaggio, quindi solo i campi *MsgId*, *Offset*e *MsgFlags* in MQMD devono essere diversi tra segmenti separati dello stesso messaggio. Se un segmento non riesce ad arrivare, viene restituito il codice motivo MQRC_INCOMPLETE_GROUP o MQRC_INCOMPLETE_MSG come appropriato.

Figura 8 a pagina 46 mostra un gruppo di messaggi logici, alcuni dei quali sono segmentati:

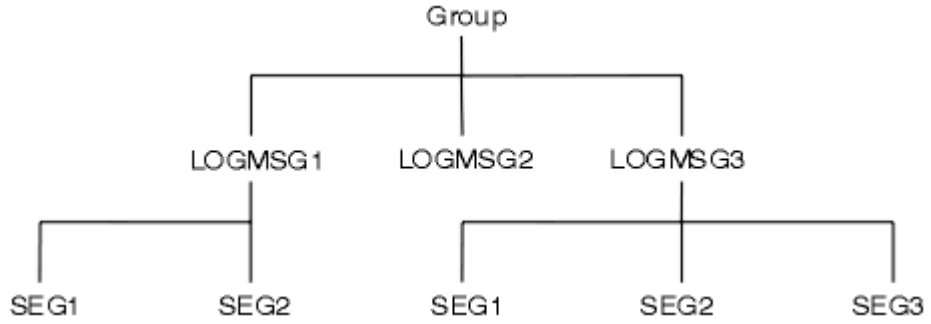


Figura 8. Messaggi segmentati

z/OS La segmentazione non è supportata su IBM MQ for z/OS.

Non è possibile utilizzare messaggi segmentati o raggruppati con Pubblicazione / Sottoscrizione.

Concetti correlati

“Segmentazione del messaggio” a pagina 793
Utilizzare queste informazioni per informazioni sulla segmentazione dei messaggi. Questa funzione non è supportata su IBM MQ for z/OS o dalle applicazioni che utilizzano IBM MQ classes for JMS.

Riferimenti correlati

“Ordinamento logico e fisico” a pagina 774
I messaggi sulle code possono verificarsi (all'interno di ogni livello di priorità) in ordine *fisico* o *logico* .

Persistenza messaggio

I messaggi persistenti vengono scritti nei log e nei file di dati della coda. Se un gestore code viene riavviato dopo un errore, recupera questi messaggi persistenti come necessario dai dati registrati. I messaggi non persistenti vengono eliminati se un gestore code si arresta, se l'arresto è dovuto a un comando dell'operatore o a un malfunzionamento di una parte del sistema.

z/OS I messaggi non persistenti memorizzati in una CF (Coupling Facility) su z/OS sono un'eccezione. Persistono fino a quando il CF rimane disponibile.

Quando si crea un messaggio, se si inizializza il descrittore del messaggio (MQMD) utilizzando i valori predefiniti, la persistenza per il messaggio viene ricavata dall'attributo **DefPersistence** della coda specificata nel comando MQOPEN. In alternativa, è possibile impostare la persistenza del messaggio utilizzando il campo di *Persistence* della struttura MQMD per definire il messaggio come persistente o non persistente.

Le prestazioni dell'applicazione vengono influenzate quando si utilizzano i messaggi persistenti; l'estensione dell'effetto dipende dalle caratteristiche delle prestazioni del sottosistema I/O della macchina e dal modo in cui si utilizzano le opzioni del punto di sincronizzazione su ciascuna piattaforma:

- Un messaggio persistente, al di fuori dell'unità di lavoro corrente, viene scritto su disco su ogni operazione put e get. Consultare “Commit e backout delle unità di lavoro” a pagina 854.
- z/OS ALW Per tutte le piattaforme tranne IBM i, un messaggio persistente all'interno dell'unità di lavoro corrente viene registrato solo quando viene eseguito il commit dell'unità di lavoro e l'unità di lavoro può contenere molte operazioni di coda.

I messaggi non persistenti possono essere utilizzati per la messaggistica rapida. Consultare Sicurezza dei messaggi per ulteriori informazioni sui messaggi rapidi.

Nota: Una combinazione di scrittura di messaggi persistenti all'interno di un'unità di lavoro e scrittura di messaggi persistenti all'esterno di un'unità o di un lavoro, può causare problemi di prestazioni potenzialmente gravi per le applicazioni. Ciò è particolarmente vero quando la stessa coda di destinazione viene utilizzata per entrambe le operazioni.

Messaggi che non vengono recapitati

Quando un gestore code non è in grado di inserire un messaggio su una coda, esistono diverse opzioni.

Sarai in grado di:

- Tentare nuovamente di inserire il messaggio nella coda.
- Richiedere che il messaggio venga restituito al mittente.
- Inserire il messaggio nella coda di messaggi non recapitabili.

Per ulteriori informazioni, consultare“Gestione degli errori del programma procedurale” a pagina 1040.

Messaggi di cui è stato eseguito il backout

Quando si elaborano i messaggi da una coda sotto il controllo di un'unità di lavoro, l'unità di lavoro può essere composta da uno o più messaggi. Se si verifica un backout, i messaggi che sono stati richiamati dalla coda vengono reintegrati nella coda e possono essere elaborati di nuovo in un'altra unità di lavoro. Se l'elaborazione di un particolare messaggio sta causando il problema, viene eseguito nuovamente il backout dell'unità di lavoro. Ciò può causare un loop di elaborazione. I messaggi inseriti in una coda vengono rimossi dalla coda.

Un'applicazione può rilevare i messaggi rilevati in tale loop verificando il campo *BackoutCount* di MQMD. L'applicazione può correggere la situazione o inviare un'avvertenza a un operatore.

Multi Il conteggio di backout sopravvive sempre ai riavvii del gestore code. Qualsiasi modifica all'attributo **HardenGetBackout** viene ignorata.

z/OS Per le code condivise, il conteggio di backout sopravvive sempre ai riavvii del gestore code. Per tutte le altre configurazioni su z/OS, per garantire che il numero di backout per le code private sopravviva ai riavvii del gestore code, impostare l'attributo *HardenGetBackout* su MQQA_BACKOUT_HARDENED; altrimenti, se il gestore code deve essere riavviato, non conserva un conteggio di backout accurato per ciascun messaggio. L'impostazione dell'attributo in questo modo aggiunge il costo dell'elaborazione supplementare.

Per ulteriori informazioni sul commit e il backout dei messaggi, consultare "Commit e backout delle unità di lavoro" a pagina 854.

Coda di risposta e gestore code

Vi sono occasioni in cui è possibile ricevere messaggi in risposta a un messaggio inviato:

- Un messaggio di risposta in risposta ad un messaggio di richiesta
- Un messaggio di report su un evento o una scadenza imprevisti
- Un messaggio di report su un evento COA (Conferma di arrivo) o COD (Conferma di consegna)
- Un messaggio di report su un evento PAN (Positive Action Notification) o NAN (Negative Action Notification)

Utilizzando la struttura MQMD, specificare il nome della coda a cui si desidera inviare i messaggi di risposta e di report nel campo *ReplyToQ* . Specificare il nome del gestore code proprietario della coda di risposta nel campo *ReplyToQMgr* .

Se si lascia vuoto il campo *ReplyToQMgr* , il gestore code imposta il contenuto dei seguenti campi nel descrittore del messaggio sulla coda:

ReplyToQ
Se *ReplyToQ* è una definizione locale di una coda remota, il campo *ReplyToQ* è impostato sul nome della coda remota; altrimenti, questo campo non viene modificato.

ReplyToQMgr
Se *ReplyToQ* è una definizione locale di una coda remota, il campo *ReplyToQMgr* è impostato sul nome del gestore code che possiede la coda remota; altrimenti, il campo *ReplyToQMgr* è impostato sul nome del gestore code a cui è connessa l'applicazione.

Nota: È possibile richiedere che un gestore code compia più di un tentativo di recapito di un messaggio ed è possibile richiedere che il messaggio venga eliminato in caso di errore. Se il messaggio, dopo non essere stato consegnato, non deve essere eliminato, il gestore code remoto inserisce il messaggio nella relativa coda di messaggi non recapitabili (messaggio non recapitato) (consultare "Utilizzo della coda dei messaggi non recapitabili (messaggi non recapitati)" a pagina 1044).

Contesto messaggio

Le informazioni relative al *contesto del messaggio* consentono all'applicazione che richiama il messaggio di individuare il creatore del messaggio.

L'applicazione di richiamo potrebbe voler:

- Verificare che l'applicazione mittente disponga del livello di autorizzazione corretto
- Eseguire alcune funzioni di contabilità in modo che possa addebitare l'applicazione di invio per qualsiasi lavoro che deve eseguire
- Conserva una traccia di verifica di tutti i messaggi che ha utilizzato

Quando si utilizza la chiamata MQPUT o MQPUT1 per inserire un messaggio in una coda, è possibile specificare che il gestore code deve aggiungere alcune informazioni di contesto predefinite al descrittore del messaggio. Le applicazioni che dispongono del livello di autorizzazione appropriato possono

aggiungere ulteriori informazioni di contesto. Per ulteriori informazioni su come specificare le informazioni di contesto, consultare “Controllo delle informazioni di contesto del messaggio” a pagina 760.

Il contesto utente viene utilizzato dal gestore code durante la generazione dei seguenti tipi di messaggi di report:

- Conferma alla consegna
- Scadenza

Quando vengono generati questi messaggi di report, il contesto utente viene controllato per l'autorizzazione + put e + passid sulla destinazione del report. Se il contesto utente non dispone di autorizzazione sufficiente, il messaggio di report viene inserito nella coda di messaggi non recapitabili, se ne è stato definito uno. Se non è presente una coda di messaggi non instradabili, il messaggio di report viene eliminato.

Tutte le informazioni di contesto vengono memorizzate nei campi di contesto del descrittore del messaggio. Il tipo di informazioni ricade nelle informazioni di identità, origine e contesto utente.

contesto di identità

Le informazioni *Contesto identità* identificano l'utente dell'applicazione che per primo ha inserito il messaggio su una coda. Le applicazioni opportunamente autorizzate possono impostare i seguenti campi:

- Il gestore code riempie il campo *UserIdentifier* con un nome che identifica l'utente; il modo in cui il gestore code può eseguire questa operazione dipende dall'ambiente in cui è in esecuzione l'applicazione.
- Il gestore code riempie il campo *AccountingToken* con un token o un numero determinato dall'applicazione che ha inserito il messaggio.
- Le applicazioni possono utilizzare il campo *ApplIdentityData* per qualsiasi informazione aggiuntiva che desiderano includere sull'utente (ad esempio, una password codificata).

Un SID (system security identifier) Windows viene memorizzato nel campo *AccountingToken* quando un messaggio viene creato in IBM MQ for Windows. Il SID può essere utilizzato per integrare il campo *UserIdentifier* e stabilire le credenziali di un utente.

Per informazioni su come il gestore code compila i campi *UserIdentifier* e *AccountingToken* , consultare le descrizioni di questi campi in UserIdentifier e AccountingToken.

Le applicazioni che inoltrano i messaggi da un gestore code a un altro devono anche trasmettere le informazioni di contesto dell'identità in modo che altre applicazioni conoscano l'identità del mittente del messaggio.

Contesto di origine

Le informazioni sul *contesto di origine* descrivono l'applicazione che inserisce il messaggio nella coda in cui il messaggio è attualmente memorizzato. Il descrittore del messaggio contiene i seguenti campi per le informazioni sul contesto di origine:

- *PutApplType* definisce il tipo di applicazione che inserisce il messaggio (ad esempio, una transazione CICS).
- *PutApplName* definisce il nome dell'applicazione che inserisce il messaggio (ad esempio, il nome di un lavoro o di una transazione).
- *PutDate* definisce la data in cui il messaggio è stato inserito sulla coda.
- *PutTime* definisce l'ora in cui il messaggio è stato inserito sulla coda.
- *ApplOriginData* definisce le informazioni aggiuntive che un'applicazione desidera includere sull'origine del messaggio. Ad esempio, potrebbe essere impostato da applicazioni autorizzate in modo appropriato per indicare se i dati di identità sono attendibili.

Le informazioni sul contesto di origine vengono generalmente fornite dal gestore code. GMT (Greenwich Mean Time) viene utilizzato per i campi *PutDate* e *PutTime* . Consultare le descrizioni di questi campi in PutDate e PutTime.

Un'applicazione con autorizzazione sufficiente può fornire il proprio contesto. Ciò consente di conservare le informazioni di account quando un singolo utente ha un ID utente differente su ciascuno dei sistemi che elaborano un messaggio originato.

Oggetti IBM MQ

Queste informazioni forniscono dettagli sugli oggetti IBM MQ che includono: gestori code, gruppi di condivisione code, code, oggetti argomento di gestione, elenchi nomi, definizioni di processo, oggetti informazioni di autenticazione, canali, classi di memoria, listener e servizi.

I gestori code definiscono le proprietà (note come attributi) di questi oggetti. I valori di questi attributi influenzano il modo in cui IBM MQ elabora tali oggetti. Dalle proprie applicazioni, si utilizza MQI (Message Queue Interface) per controllare questi oggetti. Gli oggetti vengono identificati da un *descrittore oggetto* (MQOD) quando vengono indirizzati da un programma.

Quando si utilizzano i comandi IBM MQ per definire, modificare o eliminare oggetti, ad esempio, il gestore code verifica di disporre del livello di autorizzazione richiesto per eseguire queste operazioni. Allo stesso modo, quando un'applicazione utilizza la chiamata MQOPEN per aprire un oggetto, il gestore code controlla che l'applicazione disponga del livello di autorizzazione richiesto prima di consentire l'accesso a tale oggetto. Le verifiche vengono effettuate sul nome dell'oggetto da aprire.

Concetti correlati

“Controllo delle informazioni di contesto del messaggio” a pagina 760
Quando si utilizza la chiamata MQPUT o MQPUT1 per inserire un messaggio in una coda, è possibile specificare che il gestore code deve aggiungere alcune informazioni di contesto predefinite al descrittore del messaggio. Le applicazioni che dispongono del livello di autorizzazione appropriato possono aggiungere ulteriori informazioni di contesto. È possibile utilizzare il campo opzioni nella struttura MQPMO per controllare le informazioni di contesto.

Riferimenti correlati

“Opzioni MQOPEN relative al contesto del messaggio” a pagina 750
Se si desidera poter associare le informazioni di contesto a un messaggio quando le si inserisce in una coda, è necessario utilizzare una delle opzioni di contesto del messaggio quando si apre la coda.

Windows Preparazione ed esecuzione di applicazioni di Microsoft Transaction Server

Per preparare un'applicazione MTS da eseguire come un'applicazione IBM MQ MQI client , seguire queste istruzioni come appropriato per il proprio ambiente.

Per informazioni generali su come sviluppare applicazioni Microsoft Transaction Server (MTS) che accedono a risorse IBM MQ , consultare la sezione su MTS nel Centro assistenza IBM MQ .

Per preparare un'applicazione MTS da eseguire come un'applicazione IBM MQ MQI client , effettuare una delle seguenti operazioni per ogni componente dell'applicazione:

- Se il componente utilizza i bind in linguaggio C per MQI, seguire le istruzioni in “Preparazione dei programmi C in Windows” a pagina 1018 ma collegare il componente alla libreria mqicxa.lib invece di mqic.lib.
- Se il componente utilizza le classi C++ di IBM MQ , seguire le istruzioni in “Creazione di programmi C++ su Windows” a pagina 551 ma collegare il componente alla libreria imqx23vn.lib invece di imqc23vn.lib.
- Se il componente utilizza i bind del linguaggio Visual Basic per MQI, seguire le istruzioni in “Preparazione dei programmi Visual Basic in Windows” a pagina 1022 , ma quando si definisce il progetto Visual Basic, immettere `MqType=3` nel campo **Argomenti di compilazione condizionale** .

Considerazioni sulla progettazione per applicazioni IBM MQ

Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

Quando si progetta un'applicazione IBM MQ , considerare le seguenti domande e opzioni:

Tipo di applicazione

Qual è lo scopo della tua applicazione? Consultare i seguenti link per informazioni sui diversi tipi di applicazione che è possibile sviluppare:

- Server
- Client
- Pubblicazione/sottoscrizione
- Servizi Web
- Uscite utente, uscite API e servizi installabili

Inoltre, è anche possibile scrivere le applicazioni per automatizzare la gestione di IBM MQ.
Per ulteriori informazioni, consultare The IBM MQ Administration Interface (MQAI) e Automating administration tasks.

Linguaggio programmazione

IBM MQ supporta diversi linguaggi di programmazione per la scrittura di applicazioni. Per ulteriori informazioni, consultare “Sviluppo di applicazioni per IBM MQ” a pagina 5.

Applicazioni per più di una piattaforma

L'applicazione verrà eseguita su più di una piattaforma? Hai una strategia per passare a una piattaforma diversa da quella che usi oggi? Se la risposta a una di queste domande è sì, assicurarsi di codificare i programmi per l'indipendenza della piattaforma.

Ad esempio, se si utilizza C, codice nello standard ANSI C. Utilizzare una funzione della libreria C standard piuttosto che una funzione specifica della piattaforma equivalente, anche se la funzione specifica della piattaforma è più veloce o più efficiente. L'eccezione è quando l'efficienza nel codice è fondamentale, quando è necessario codificare per entrambe le situazioni utilizzando #ifdef. Ad esempio:

```
#ifdef _AIX
    AIX specific code
#else
    generic code
#endif
```

Tipi di code

Si desidera creare una coda ogni volta che ne è necessaria una o si desidera utilizzare code già impostate? Si desidera eliminare una coda quando si è terminato di utilizzarla o verrà utilizzata di nuovo? Si desidera utilizzare le code alias per l'indipendenza dell'applicazione? Per vedere quali tipi di code sono supportati, fare riferimento a Code.

z/OS **Utilizzo di code condivise, gruppi di condivisione code e cluster di gruppi di condivisione code (solo IBM MQ for z/OS)**

È possibile trarre vantaggio dall'aumento di disponibilità, scalabilità e bilanciamento del carico di lavoro che sono possibili quando si utilizzano code condivise con gruppi di condivisione code. Per ulteriori informazioni, consultare Code condivise e gruppi di condivisione code .

Si potrebbe anche voler stimare i flussi di messaggi medi e di picco e considerare l'utilizzo dei cluster di gruppi di condivisione code per diffondere il carico di lavoro. Per ulteriori informazioni, consultare Code condivise e gruppi di condivisione code .

Usando i cluster del gestore code

È possibile trarre vantaggio dall'amministrazione del sistema semplificata e da una maggiore disponibilità, scalabilità e bilanciamento del carico di lavoro possibili quando si utilizzano i cluster.

Tipi di messaggi

È possibile utilizzare i datagrammi per i messaggi semplici, ma richiedere i messaggi (per i quali si prevedono risposte) per altre situazioni. È possibile assegnare diverse priorità ad alcuni messaggi. Per ulteriori informazioni sulla progettazione dei messaggi, consultare “Tecniche di progettazione per i messaggi” a pagina 59.

Utilizzo della messaggistica di pubblicazione / sottoscrizione o point - to - point

Utilizzando la messaggistica di pubblicazione / sottoscrizione, un'applicazione di invio invia le informazioni che desidera condividere in un messaggio IBM MQ ad una destinazione standard gestita dalla sottoscrizione IBM MQ publish? e consente a IBM MQ di gestire la distribuzione di tali informazioni. L'applicazione di destinazione non deve sapere nulla sulla fonte delle informazioni che riceve, registra solo un interesse per uno o più argomenti e riceve tali informazioni quando sono disponibili. Per ulteriori informazioni sulla messaggistica di pubblicazione / sottoscrizione, consultare Messaggistica di pubblicazione / sottoscrizione.

Utilizzando la messaggistica point - to - point, un'applicazione di invio invia un messaggio a una coda specifica, da dove sa che un'applicazione di ricezione lo richiamerà. Un'applicazione ricevente riceve i messaggi da una coda specifica e agisce sul contenuto. Un'applicazione spesso funziona sia come mittente che come destinatario, inviando una query a un'altra applicazione e ricevendo una risposta.

Controllo dei programmi IBM MQ

È possibile che si desideri avviare alcuni programmi automaticamente o far sì che i programmi attendano l'arrivo di un determinato messaggio su una coda (utilizzando la funzione IBM MQ *triggering* , consultare “Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866). In alternativa, è possibile avviare un'altra istanza di un'applicazione quando i messaggi su una coda non vengono elaborati abbastanza velocemente (utilizzando la funzione IBM MQ Eventi di strumentazione come descritto in Eventi di strumentazione).

Esecuzione della tua applicazione su un client IBM MQ

L'MQI completa è supportato nell'ambiente del client e quasi tutte le applicazioni IBM MQ scritte in un linguaggio procedurale possono essere ricollegate per essere eseguite su un IBM MQ MQI client. Collegare l'applicazione su IBM MQ MQI client alla libreria MQIC, piuttosto che alla libreria MQI.

z/OS Get (segnale) on z/OS non è supportato.

Nota: Un'applicazione in esecuzione su un client IBM MQ può connettersi a più di un gestore code contemporaneamente oppure utilizzare un nome gestore code con un asterisco (*) su una chiamata MQCONN o MQCONNX. Modificare l'applicazione se si desidera creare un collegamento alle librerie del gestore code invece che alle librerie del client, in quanto questa funzione non sarà disponibile.

Per ulteriori informazioni, consultare “Esecuzione di applicazioni nell'ambiente di IBM MQ MQI client” a pagina 923.

Prestazioni delle applicazioni

Le decisioni di progettazione possono influenzare le prestazioni dell'applicazione, per suggerimenti per migliorare le prestazioni delle applicazioni IBM MQ , consultare “Considerazioni sulla progettazione dell'applicazione e sulle prestazioni” a pagina 61 IBM i e “Considerazioni sulla progettazione e sulle prestazioni per le applicazioni IBM i” a pagina 65 .

Tecniche IBM MQ avanzate

Per applicazioni più avanzate, è possibile utilizzare alcune tecniche IBM MQ avanzate, come la correlazione delle risposte e la generazione e l'invio di informazioni di contesto IBM MQ . Per ulteriori informazioni, consultare “Tecniche di progettazione per applicazioni avanzate” a pagina 63.

Proteggere i dati e mantenerne l'integrità

È possibile utilizzare le informazioni di contesto passate con un messaggio per verificare che il messaggio sia stato inviato da un'origine accettabile. È possibile utilizzare le funzioni di sincronizzazione fornite da IBM MQ o dal sistema operativo per garantire che i dati rimangano congruenti con altre risorse (consultare “Commit e backout delle unità di lavoro” a pagina 854 per ulteriori dettagli). È possibile utilizzare la funzione *persistenza* dei messaggi IBM MQ per assicurare la consegna di messaggi importanti.

Test delle applicazioni IBM MQ

L'ambiente di sviluppo dell'applicazione per programmi IBM MQ non è diverso da quello per qualsiasi altra applicazione, quindi è possibile utilizzare gli stessi strumenti di sviluppo e le funzioni di traccia IBM MQ .

Quando si verificano le applicazioni CICS con IBM MQ for z/OS, è possibile utilizzare CEDF (CICS Execution Diagnostic Facility). CEDF esegue il trap dell'entrata e dell'uscita di ogni chiamata MQI e delle chiamate a tutti i servizi CICS . Inoltre, nell'ambiente CICS , è possibile scrivere un programma di uscita API - crossing per fornire informazioni diagnostiche prima e dopo ogni chiamata MQI. Per informazioni su come svolgere questa procedura, consultare “Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890.

Quando si verificano le applicazioni IBM i , è possibile utilizzare il Debugger standard. Per iniziare, utilizzare il comando STRDBG.

Gestione di eccezioni ed errori

È necessario considerare come elaborare i messaggi che non possono essere consegnati e come risolvere le situazioni di errore riportate dal gestore code. Per alcuni report, è necessario impostare le opzioni di report su MQPUT.

Concetti correlati

IBM MQ - Sommario tecnico

“Considerazioni sulla progettazione e sulle prestazioni per le applicazioni z/OS” a pagina 67
La progettazione delle applicazioni è uno dei principali fattori che influiscono sulle prestazioni. Utilizzare questo argomento per comprendere alcuni dei fattori di progettazione coinvolti nelle prestazioni.

“Sviluppo di applicazioni per IBM MQ” a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

“Scrittura di un'applicazione procedurale per l'accodamento” a pagina 721
Utilizzare queste informazioni per informazioni sulla scrittura delle applicazioni di accodamento, la connessione e la disconnessione da un gestore code, la pubblicazione / sottoscrizione e l'apertura e la chiusura di oggetti.

“Scrittura di applicazioni procedurali client” a pagina 915
Cosa devi sapere per scrivere le applicazioni client su IBM MQ utilizzando un linguaggio procedurale.

“Sviluppo di applicazioni .NET” a pagina 556
IBM MQ classes for .NET consentire alle applicazioni .NET di connettersi a IBM MQ come IBM MQ MQI client o di connettersi direttamente a un server IBM MQ .

“Sviluppo di applicazioni C + +” a pagina 528
IBM MQ fornisce classi C+ + equivalenti a oggetti IBM MQ e alcune classi aggiuntive equivalenti a tipi di dati array. Fornisce un certo numero di funzioni non disponibili tramite MQI.

“Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

“Utilizzo di IBM MQ classes for Java” a pagina 349
Utilizzare IBM MQ in un ambiente Java . IBM MQ classes for Java consenti a un'applicazione Java di connettersi a IBM MQ come client IBM MQ o di connettersi direttamente a un gestore code IBM MQ .

Specifica del nome dell'applicazione nei linguaggi di programmazione supportati

Prima di IBM MQ 9.2.0, era già possibile specificare un nome applicazione su applicazioni client Java o JMS . Da IBM MQ 9.2.0 questa funzione è estesa ad altri linguaggi di programmazione su IBM MQ for Multiplatforms.

Come viene utilizzato il nome dell'applicazione

Il nome dell'applicazione viene emesso da:

- runmqsc VISUALIZZA APPLTAG CONN
- runmqsc VISUALIZZA QSTATUS TYPE (HANDLE) APPLTAG
- runmqsc VISUALIZZA RAPPLTAG CHSTATUS
- MQMD.PutApplName
- Traccia attività applicazione

Il nome dell'applicazione viene utilizzato anche durante la configurazione della traccia dell'attività dell'applicazione. Il nome applicazione predefinito per le applicazioni nonJava è il nome troncato dell'eseguibile, tranne su Windows e IBM i.

Windows Su Windows, il nome predefinito è il nome completo dell'eseguibile, troncato a 28 caratteri a sinistra.

IBM i Su IBM i, il nome predefinito è il nome lavoro.

Per applicazioni Java è il nome della classe preceduto dal nome del pacchetto troncato a sinistra a 28 caratteri.

Per ulteriori informazioni, vedi NomePutAppl.

Da IBM MQ 9.2.0, le applicazioni su IBM MQ for Multiplatforms possono impostare i propri nomi applicazione sia amministrativamente che utilizzando vari metodi di programmazione. Ciò consente alle applicazioni di fornire un nome più significativo indipendente dalla piattaforma, quando si configura la traccia dell'attività dell'applicazione o quando si esegue l'output da vari comandi **runmqsc** .

Da IBM MQ 9.2.0, puoi ribilanciare le applicazioni in un cluster uniforme. Per raggiungere questo obiettivo vengono utilizzati nomi di applicazioni significativi.

Caratteri supportati

Consultare “Caratteri nome applicazione consigliati” a pagina 55 per ulteriori informazioni su come specificare il nome dell'applicazione.

Linguaggi di programmazione

Consultare “Connessioni linguaggio di programmazione” a pagina 57 per ulteriori informazioni su come le applicazioni che si risolvono nelle librerie IBM MQ in C e in altri linguaggi di programmazione possono fornire il nome dell'applicazione.

Applicazioni .NET gestite

Consultare “Applicazioni .NET gestite” a pagina 58 per informazioni su come le applicazioni .NET gestite possono fornire il nome dell'applicazione.

Applicazioni XMS

Consultare “XMS applicazioni” a pagina 58 per informazioni su come le applicazioni XMS possono fornire il nome dell'applicazione.

Applicazioni di bind Java e JMS

ALW

Consultare “Applicazioni di bind Java e JMS” a pagina 59 per informazioni su come le applicazioni Java e JMS possono fornire il nome dell'applicazione.

Concetti correlati

Traccia attività applicazione

Informazioni sui cluster uniformi

Riferimenti correlati

MQCNO

MQCNO su IBM i

Utilizzo del nome applicazione nei linguaggi di programmazione supportati

Utilizzare queste informazioni per informazioni su come viene selezionato il nome dell'applicazione nelle diverse lingue supportate da IBM MQ .

Caratteri nome applicazione consigliati

I nomi applicazione devono essere nella serie di caratteri fornita dall'attributo **CodedCharSetId** del campo gestore code. Per ulteriori informazioni su questo attributo, consultare Attributi per il gestore code.

Tuttavia, se l'applicazione è in esecuzione come IBM MQ MQI client, il nome dell'applicazione deve essere nella serie di caratteri e nella codifica del client.

Per garantire una transizione fluida del nome dell'applicazione tra i gestori code e per consentire il monitoraggio delle risorse dell'applicazione tramite gli argomenti di monitoraggio delle risorse, i nomi applicazione devono contenere solo caratteri stampabili a byte singolo.

Note:

- Si consiglia inoltre di evitare l'uso di caratteri barra e e commerciale nei nomi delle applicazioni.
- V 9.3.0 È necessario evitare di utilizzare il carattere e commerciale nei nomi delle applicazioni. Le metriche STATAPP dell'argomento di sistema per nomi applicazione contenenti una e commerciale non verranno prodotte.

Ciò limita il nome a:

- Caratteri alfanumerici: `A-Z, a-ze 0-9`

  Nota: Non utilizzare i caratteri minuscoli `a-z` nei nomi applicazione sui sistemi che utilizzano EBCDIC Katakana.
- Il carattere spazio
- Caratteri stampabili non modificabili in EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`
- V 9.3.0 Il carattere /. Quando si esegue la sottoscrizione alla traccia dell'attività o alle metriche dell'argomento di sistema STATAPP per un'applicazione il cui nome contiene una barra, è necessario sostituire qualsiasi carattere barra con una e commerciale. Ad esempio, per ricevere le metriche STATAPP per un'applicazione denominata "DEPT1/APPS/STOCKQUOTE" è necessario sottoscrivere la stringa di argomenti "$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE". Le applicazioni di esempio amqsact e amqsrua convertiranno automaticamente i caratteri barra in e commerciali durante la creazione delle relative sottoscrizioni.

Come si impostano i caratteri

La seguente tabella riepiloga i mezzi con cui viene scelto il nome dell'applicazione nelle varie lingue supportate da IBM MQ . Il mezzo con cui viene scelto il nome è in ordine di precedenza, il primo più alto.

	Bind C e client	**Bind e client di Java**	**Bind e client di JMS**	**Client . NET gestito**	**Client e bind .NET non gestiti**	**Client XMS gestito**	**Client e bindXMS non gestiti**
Sovrascrittura proprietà di connessione		Java sovrascrittura proprietà di connessione		.Sovrascrittura della proprietà di connessione NET	.Sovrascrittura della proprietà di connessione NET		
proprietà Sovrascritta		Java proprietà sovrascritta		.Proprietà sovrascritta NET	.Proprietà sovrascritta NET		
Ambiente MQ		Java Ambiente MQ		.Ambiente MQ NET	.Ambiente MQ NET		
Proprietà factory di connessione			Proprietà factory di connessione			Proprietà factory di connessione	Proprietà factory di connessione
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	Opzioni di connessione						
Variabile di ambiente	variabili di ambiente				variabili di ambiente		variabili di ambiente
mqclient.ini (Applicabile solo alle connessioni client)	Connessioni client				Connessioni client		Connessioni client
Java nome classe		Java nome classe	Java nome classe				
Nome predefinito	Nome predefinito			.Nome predefinito NET	.Nome predefinito NET	.Nome predefinito NET	.Nome predefinito NET

Nota: I collegamenti C e la colonna client si applicano anche ai seguenti linguaggi di programmazione:

- COBOL
- Assembler
- Visual Basic
- RPG

Connessioni linguaggio di programmazione

Le applicazioni che si risolvono nelle librerie IBM MQ in C e in altri linguaggi di programmazione, possono fornire il nome dell'applicazione nei seguenti modi.

I metodi di connessione sono elencati in ordine di precedenza, a partire dal più alto.

Multi **Opzioni di connessione**

- ALW MQCNO

  Nota: z/OS Quando ci si connette a un gestore code IBM MQ for z/OS , è possibile impostare il nome dell'applicazione solo utilizzando le connessioni in modalità client o utilizzando le applicazioni IBM MQ classes for JMS o IBM MQ classes for Java .

- IBM i MQCNO su IBM i

ALW **variabili di ambiente**

Se non è stato ancora selezionato un nome applicazione, è possibile utilizzare la variabile di ambiente **MQAPPLNAME** per identificare la connessione al gestore code. Ad esempio:

```
export MQAPPLNAME=ExampleApplName
```

Si noti che vengono utilizzati solo i primi 28 caratteri e questi caratteri non devono essere tutti spazi o valori null.

Nota: L'attributo è valido solo per i linguaggi di programmazione supportati, .NETnon gestiti e le connessioni XMS non gestite.

ALW **File di configurazione client**

Se non è stato ancora selezionato un nome applicazione e la connessione è una connessione client, è possibile specificare le seguenti informazioni nel file di configurazione del client (ad esempio, `mqclient.ini`) per identificare la connessione al gestore code.

```
Connection:
    ApplName=ExampleApplName
```

Note:

1. Vengono utilizzati solo i primi 28 caratteri e questi caratteri non devono essere tutti spazi o valori null.
2. L'attributo si applica solo alle connessioni client sui linguaggi di programmazione supportati, .NETnon gestiti e XMS non gestiti.

Per ulteriori informazioni, consultare il file di configurazione di IBM MQ MQI client , `mqclient.ini`.

Nome predefinito

Se non è stato ancora scelto il nome dell'applicazione, il nome predefinito continua ad essere utilizzato, che contiene la quantità di percorso e di nome eseguibile visualizzata dal sistema operativo. Per ulteriori informazioni, vedi NomePutAppl.

Applicazioni .NET gestite

Le applicazioni .NET gestite possono fornire il nome dell'applicazione nei modi seguenti.

I metodi di connessione sono elencati in ordine di precedenza, a partire dal più alto.

Sovrascrittura proprietà connessione

È possibile fornire un file di sovrascrittura dei dettagli di connessione alle applicazioni nel seguente modo:

```
<appSettings>
     <add key="overrideConnectionDetails" value="true" />
     <add key="overrideConnectionDetailsFile" value="<location>" />
</appSettings>
```

Il file specificato da `overrideConnectionDetailsFile` contiene un elenco di proprietà con prefisso `mqj`. Le applicazioni devono definire la proprietà **`mqj.APPNAME`** in cui il valore della proprietà **`mqj.APPNAME`** specifica il nome utilizzato per identificare la connessione al gestore code.

Vengono utilizzati solo i primi 28 caratteri del nome. Ad esempio:

```
mqj.APPNAME=ExampleApplName
```

Proprietà sovrascritta

Una costante **`MQC.APPNAME_PROPERTY`** è stata definita con il valore *APPNAME*. Ora è possibile passare questa proprietà al costruttore **`MQQueueManager`** , utilizzando solo i primi 28 caratteri del nome. Ad esempio:

```
Hashtable properties = new Hashtable();
properties.Add( MQC.APPNAME_PROPERTY, "ExampleApplName" );
MQQueueManager qMgr = new MQQueueManager("qmgrname", properties);
```

Per ulteriori informazioni, consultare “Operazioni gestite e non gestite in .NET” a pagina 653.

Ambiente MQ

La proprietà *AppName* viene aggiunta alla classe **`MQEnvironment`**e vengono utilizzati solo i primi 28 caratteri. Ad esempio:

```
MQEnvironment.AppName = "ExampleApplName";
```

Nome predefinito

Se non è stato fornito il nome dell'applicazione con uno dei mezzi descritti nel testo precedente, il nome dell'applicazione viene automaticamente impostato in modo da essere il nome eseguibile (e la maggior parte del percorso che si adatterà).

XMS applicazioni

I metodi di connessione sono elencati in ordine di precedenza, a partire dal più alto.

Proprietà factory di connessione

Le applicazioni XMS possono fornire il nome dell'applicazione sul factory di connessione utilizzando la proprietà **`XMSC.WMQ_APPLICATIONNAME`** ("*XMSC_WMQ_APPNAME*") in modo simile a JMS. È possibile specificare fino a 28 caratteri.

Per ulteriori informazioni, consultare “XMS .NET creazione di oggetti gestiti” a pagina 662 e “Proprietà di un messaggio XMS” a pagina 670.

JMSAdmin

Nella strumentazione amministrativa la proprietà è nota come "**APPLICATIONNAME**" o "**APPNAME**", in breve.

Applicazioni di bind Java e JMS

I metodi di connessione sono elencati in ordine di precedenza, a partire dal più alto.

ALW Le applicazioni client Java e JMS possono già specificare un nome applicazione, che è stato esteso su IBM MQ for Multiplatforms alle applicazioni di bind, utilizzando il campo MQCNO **ApplName** .

Sovrascrittura proprietà connessione

La proprietà **Application name** è stata aggiunta all'elenco di proprietà di connessione che è possibile sovrascrivere. Per ulteriori informazioni, consultare Utilizzo della sovrascrittura della proprietà di connessione IBM MQ.

Attenzione: Le proprietà di connessione e il modo in cui utilizzare il file di sovrascrittura della proprietà di connessione sono uguali per IBM MQ classes for Java e .NET.

Proprietà sovrascritta

Una costante **MQC.APPNAME_PROPERTY** è stata definita con il valore *APPNAME*. Ora è possibile passare questa proprietà al costruttore **MQQueueManager** , utilizzando solo i primi 28 caratteri del nome. Per ulteriori informazioni, consultare Utilizzo della sovrascrittura della proprietà di connessione in IBM MQ classes for Java.

Ambiente MQ

La proprietà *AppName* viene aggiunta alla classe **MQEnvironment** e vengono utilizzati solo i primi 28 caratteri.

Per ulteriori informazioni, consultare “Configurazione dell'ambiente di IBM MQ per IBM MQ classes for Java” a pagina 377.

Java nome classe

Se non è stato fornito il nome dell'applicazione mediante uno dei mezzi nel testo precedente, il nome dell'applicazione deriva dal nome della classe principale.

Per ulteriori informazioni, consultare “Configurazione dell'ambiente di IBM MQ per IBM MQ classes for Java” a pagina 377.

Attenzione: IBM i Su IBM i non è possibile eseguire la query del nome della classe principale, quindi viene utilizzato IBM MQ client per Java .

Concetti correlati

“Configurazione dell'ambiente di IBM MQ per IBM MQ classes for Java” a pagina 377
Perché un'applicazione si connetta a un gestore code in modalità client, l'applicazione deve specificare il nome canale, il nome host e il numero di porta.

Riferimenti correlati

MQCNO
MQCNO su IBM i

Tecniche di progettazione per i messaggi

Considerazioni che consentono di progettare i messaggi, incluse considerazioni per i selettori e le proprietà dei messaggi.

Cose da considerare nella fase di progettazione

Creare un messaggio quando si utilizza una chiamata MQI per inserire il messaggio su una coda. Come input per la chiamata, si forniscono alcune informazioni di controllo in un *descrittore di messaggi* (MQMD) e i dati che si desidera inviare ad un altro programma. Ma nella fase di progettazione, è necessario considerare quanto segue, perché influenzano il modo in cui si creano i propri messaggi:

Tipo di messaggio da utilizzare

Si sta progettando una semplice applicazione in cui è possibile inviare un messaggio, quindi non intraprendere ulteriori azioni? O stai chiedendo una risposta a una domanda? Se si sta facendo una domanda, è possibile includere nel descrittore del messaggio il nome della coda su cui si desidera ricevere la risposta.

Si desidera che i messaggi di richiesta e risposta siano sincroni? Ciò implica che si imposta un periodo di timeout per la risposta per rispondere alla richiesta, e se non si riceve la risposta entro tale periodo, viene trattato come un errore.

Oppure preferisci lavorare in modo asincrono, in modo che i tuoi processi non dipendano dal verificarsi di eventi specifici, come i segnali di temporizzazione comuni?

Un'altra considerazione è se hai tutti i tuoi messaggi all'interno di un'unità di lavoro.

Assegnazione di priorità differenti ai messaggi

È possibile assegnare un valore di priorità a ciascun messaggio e definire la coda in modo che conservi i messaggi in ordine di priorità. Se si esegue questa operazione, quando un altro programma richiama un messaggio dalla coda, riceve sempre il messaggio con la priorità più alta. Se la coda non conserva i messaggi in ordine di priorità, un programma che richiama i messaggi dalla coda li richiamerà nell'ordine in cui sono stati aggiunti alla coda.

I programmi possono anche selezionare un messaggio utilizzando l'identificativo assegnato dal gestore code quando il messaggio è stato inserito nella coda. In alternativa, è possibile generare i propri identificativi per ciascuno dei messaggi.

Effetto del riavvio del gestore code sui messaggi

Il gestore code conserva tutti i messaggi persistenti, ripristinandoli quando necessario dai file di log IBM MQ , quando vengono riavviati. I messaggi non persistenti e le code dinamiche temporanee non vengono conservati. I messaggi che non si desidera eliminare devono essere definiti come persistenti quando vengono creati. Quando si scrive un'applicazione per IBM MQ for Windows o IBM MQ su sistemi AIX and Linux , accertarsi di sapere come è stato impostato il sistema rispetto all'assegnazione del file di log per ridurre il rischio di progettare un'applicazione che verrà eseguita nei limiti del file di log.

z/OS Poiché i messaggi sulle code condivise (disponibili solo su IBM MQ for z/OS) sono conservati nella CF (coupling facility), i messaggi non persistenti vengono conservati durante i riavvii di un gestore code finché la CF rimane disponibile. Se la CF ha esito negativo, i messaggi non persistenti vengono persi.

Fornire informazioni personali al destinatario dei messaggi

Di solito, il gestore code imposta l'ID utente, ma le applicazioni debitamente autorizzate possono anche impostare questo campo, in modo che sia possibile includere il proprio ID utente e altre informazioni che il programma di ricezione può utilizzare per scopi di account o di sicurezza.

Quantità di code di ricezione

Multi Se un messaggio deve essere inserito in più code, è possibile pubblicarlo in un argomento o in un elenco di distribuzione.

z/OS Se è necessario inserire un messaggio in diverse code, è possibile pubblicarlo in un argomento.

Selettori e proprietà dei messaggi

Ai messaggi possono essere associati metadati insieme al payload del messaggio principale. Queste proprietà del messaggio possono essere utili per fornire ulteriori dati.

Ci sono due aspetti di questi dati aggiuntivi che è importante conoscere:

- Le proprietà non sono soggette a protezione AMS (Advanced Message Security). Se si desidera utilizzare AMS per proteggere i dati, inserirlo nel payload e non nelle proprietà del messaggio.
- Le proprietà possono essere utilizzate per eseguire la selezione dei messaggi.

È importante notare che l'utilizzo dei selettori viola la convenzione del messaggio standard del primo in uscita. Poiché il gestore code è ottimizzato per questo carico di lavoro, la fornitura di selettori complessi non è consigliata per motivi di prestazioni. Il gestore code non memorizza gli indici delle proprietà del messaggio, quindi la ricerca di un messaggio deve essere una ricerca lineare. Più profonda è la coda, più complesso è il selettore e minore è la probabilità che il selettore che corrisponde a un messaggio influisca negativamente sulle prestazioni.

Se è richiesta una selezione complessa, si consiglia di filtrare i messaggi utilizzando qualsiasi applicazione o motore di elaborazione, ad esempio IBM Integration Bus, su destinazioni differenti. In alternativa, l'utilizzo di una gerarchia di argomenti potrebbe essere utile.

Nota: IBM MQ classes for Java non supportano l'utilizzo dei selettori, se si desidera utilizzare i selettori questi devono essere eseguiti tramite l'API JMS .

Considerazioni sulla progettazione dell'applicazione e sulle prestazioni

Esistono diversi modi in cui la scarsa progettazione del programma può influire sulle prestazioni. Questi possono essere difficili da rilevare perché il programma può sembrare di eseguire bene se stesso, ma influenzano le prestazioni di altre attività. In questo argomento vengono illustrati diversi problemi specifici dei programmi che effettuano chiamate IBM MQ .

Ecco alcune idee per aiutarti a progettare applicazioni efficienti:

- Progetta la tua applicazione in modo che l'elaborazione proceda in parallelo con il tempo di pensiero di un utente:
  - Visualizzare un pannello e consentire all'utente di iniziare a digitare mentre l'applicazione è ancora in fase di inizializzazione.
  - Ottenere i dati necessari in parallelo da server differenti.
- Mantenere le connessioni e le code aperte se si intende riutilizzarle invece di aprirle e chiuderle ripetutamente, collegarle e disconnetterle.
- Tuttavia, un'applicazione server che sta inserendo solo un messaggio deve utilizzare MQPUT1.
- I gestori code sono ottimizzati per i messaggi di dimensione compresa tra 4 KB e 100 KB. I messaggi molto grandi sono inefficienti; è probabilmente meglio inviare 100 messaggi da 1 MB ciascuno piuttosto che un singolo messaggio da 100 MB. Anche i messaggi molto piccoli sono inefficienti. Il gestore code esegue la stessa quantità di lavoro per un messaggio a byte singolo come per un messaggio da 4 KB.
- Conservare i messaggi all'interno di un'unità di lavoro in modo che sia possibile eseguirne il commit o il backout contemporaneamente.
- Utilizzare l'opzione non persistente per i messaggi che non devono essere ripristinabili.
- Se è necessario inviare un messaggio a un numero di code di destinazione, utilizzare un elenco di distribuzione.

Effetto della lunghezza del messaggio

La quantità di dati in un messaggio può influire sulle prestazioni dell'applicazione che elabora il messaggio. Per ottenere le migliori prestazioni dalla tua applicazione, invia solo i dati essenziali in un messaggio. Ad esempio, in una richiesta di addebito di un conto bancario, le uniche informazioni che

potrebbero dover essere inoltrate dal client all'applicazione server sono il numero di conto e l'importo dell'addebito.

Effetto della persistenza del messaggio

I messaggi persistenti vengono generalmente registrati. La registrazione dei messaggi riduce le prestazioni della tua applicazione, quindi utilizza i messaggi persistenti solo per i dati essenziali. Se i dati in un messaggio possono essere eliminati se il gestore code si arresta o ha esito negativo, utilizzare un messaggio non persistente.

z/OS Le operazioni MQPUT e MQGET per i messaggi persistenti verranno bloccate quando lo spazio del log di ripristino non è sufficiente per registrare le operazioni. Tale condizione è indicata nel log dei lavori del gestore code dai messaggi CSQJ110E e CSQJ111A. Assicurarsi che i processi di controllo siano attivi in modo che tali condizioni siano gestite ed evitate.

Ricerca di un particolare messaggio

La chiamata MQGET di solito richiama il primo messaggio da una coda. Se si utilizzano gli identificativi di correlazione e messaggio (*MsgId* e *CorrelId*) nel descrittore del messaggio per specificare un particolare messaggio, il gestore code deve ricercare la coda fino a quando non trova quel messaggio. L'utilizzo della chiamata MQGET in questo modo influisce sulle prestazioni dell'applicazione.

Code che contengono messaggi di lunghezza diversa

Se l'applicazione non può utilizzare messaggi di lunghezza fissa, ingrandire e ridurre i buffer in modo dinamico per adattarli alla dimensione tipica del messaggio. Se l'applicazione emette una chiamata MQGET che non riesce perché il buffer è troppo piccolo, viene restituita la dimensione dei dati del messaggio. Aggiungi codice alla tua applicazione in modo che il buffer venga ridimensionato di conseguenza e la chiamata MQGET venga reimmessa.

Nota: Se non si imposta esplicitamente l'attributo **MaxMsgLength** , il valore predefinito è 4 MB, che potrebbe essere molto inefficiente se viene utilizzato per influenzare la dimensione del buffer dell'applicazione.

Frequenza dei punti di sincronizzazione

I programmi che emettono un numero molto elevato di chiamate MQPUT o MQGET all'interno del punto di sincronizzazione, senza eseguirne il commit, possono causare problemi di prestazioni. Le code interessate possono riempirsi di messaggi attualmente inaccessibili, mentre altre attività potrebbero essere in attesa di ricevere tali messaggi. Ciò ha implicazioni in termini di memoria, e in termini di thread che sono collegati con le attività che stanno tentando di ottenere messaggi.

Utilizzo della chiamata MQPUT1

Utilizzare la chiamata MQPUT1 solo se si dispone di un singolo messaggio da inserire in una coda. Se si desidera inserire più di un messaggio, utilizzare la chiamata MQOPEN, seguita da una serie di chiamate MQPUT e da una singola chiamata MQCLOSE.

Numero di thread in uso

Windows Per IBM MQ for Windows, un'applicazione potrebbe richiedere un numero elevato di thread. A ciascun processo del gestore code viene assegnato un numero massimo consentito di thread dell'applicazione.

Le applicazioni potrebbero utilizzare troppi thread. Considerare se l'applicazione prende in considerazione questa possibilità e se intraprende azioni per arrestare o notificare questo tipo di ricorrenza.

Inserisci messaggi persistenti nel punto di sincronizzazione

I messaggi persistenti devono essere inseriti e ricevuti nel punto di sincronizzazione. Ciò è dovuto al fatto che quando si richiama un messaggio persistente al di fuori del punto di sincronizzazione, se il richiamo ha esito negativo, non è possibile per l'applicazione sapere se il messaggio è stato ricevuto dalla coda o meno e se, se il messaggio è stato ricevuto, è stato perso. Quando si richiamano i messaggi persistenti nel punto di sincronizzazione, se qualcosa non riesce, viene eseguito il rollback della transazione e il messaggio persistente non viene perso perché si trova ancora nella coda.

Allo stesso modo, quando si inserono messaggi persistenti, inserirli nel punto di sincronizzazione. Un altro motivo per inserire e richiamare i messaggi persistenti nel punto di sincronizzazione è che il codice del messaggio persistente in IBM MQ è fortemente ottimizzato per il punto di sincronizzazione. Quindi, l'inserimento e il richiamo di messaggi persistenti nel punto di sincronizzazione è più rapido rispetto all'inserimento e al richiamo di messaggi persistenti al di fuori del punto di sincronizzazione.

Se l'applicazione inserisce i messaggi persistenti al di fuori del punto di sincronizzazione, il gestore code verifica se è possibile creare un punto di sincronizzazione implicito per conto dell'applicazione. Se il gestore code è in grado di eseguire tale operazione, include l'inserimento all'interno del punto di sincronizzazione e ne esegue automaticamente il commit. Consultare “Punto di sincronizzazione implicito su Multiplatforms” a pagina 863 per una descrizione più dettagliata.

Tuttavia, è più rapido inserire e ottenere messaggi non persistenti al di fuori del punto di sincronizzazione, poiché il codice non persistente in IBM MQ è ottimizzato per essere esterno al punto di sincronizzazione. L'inserimento e il richiamo dei messaggi persistenti avviene alla velocità del disco poiché il messaggio persistente è persistente su disco. Tuttavia, l'inserimento e il richiamo di messaggi non persistenti avviene alla velocità della CPU perché non è coinvolta la scrittura su disco, nemmeno quando si utilizza il punto di sincronizzazione.

Se un'applicazione sta ricevendo messaggi e non sa in anticipo se sono persistenti o meno, è possibile utilizzare l'opzione GMO MQGMO_SYNCPOINT_IF_PERSISTENT.

Tecniche di progettazione per applicazioni avanzate

Quando si progettano applicazioni più avanzate, è possibile considerare alcune tecniche quali l'attesa di messaggi, la correlazione delle risposte, l'impostazione e l'utilizzo delle informazioni di contesto, l'avvio automatico delle applicazioni, la creazione di report e la rimozione di affinità di messaggi quando si utilizza il clustering.

Per una applicazione IBM MQ semplice, è necessario decidere quali oggetti IBM MQ utilizzare nell'applicazione e quali tipi di messaggi si desidera utilizzare. Per un'applicazione più avanzata, è possibile utilizzare alcune delle tecniche introdotte nelle seguenti sezioni.

In attesa di messaggi

Un programma che serve una coda può attendere i messaggi per:

- Attesa fino all'arrivo di un messaggio o fino alla scadenza di un intervallo di tempo specificato (consultare “In attesa di messaggi” a pagina 798).
- z/OS Solo su IBM MQ for z/OS , impostare un segnale in modo che il programma venga informato quando arriva un messaggio. Per ulteriori informazioni, consultare “Segnaletica” a pagina 798.
- Stabilire un'uscita di callback da guidare quando arriva un messaggio; consultare “Utilizzo asincrono dei messaggi IBM MQ” a pagina 43.
- Esecuzione di chiamate periodiche sulla coda per verificare se un messaggio è arrivato (*polling*). Ciò non è in genere consigliabile perché può avere implicazioni sulle prestazioni.

Correlazione delle risposte

Nelle applicazioni IBM MQ , quando un programma riceve un messaggio che lo richiede per eseguire alcune operazioni, il programma in genere invia uno o più messaggi di risposta al richiedente.

Per aiutare il richiedente ad associare queste risposte alla sua richiesta originale, un'applicazione può impostare un *identificativo di correlazione* nel descrittore di ogni messaggio. I programmi quindi copiano l'identificativo del messaggio di richiesta nel campo identificativo di correlazione dei relativi messaggi di risposta.

Impostazione e utilizzo delle informazioni di contesto

Le *informazioni di contesto* sono utilizzate per associare i messaggi all'utente che li ha generati e per identificare l'applicazione che ha generato il messaggio. Tali informazioni sono utili per la sicurezza, la contabilità, il controllo e la determinazione dei problemi.

Quando si crea un messaggio, è possibile specificare un'opzione che richiede che il gestore code associ le informazioni di contesto predefinite al proprio messaggio.

Per ulteriori informazioni sull'utilizzo e l'impostazione delle informazioni di contesto, consultare "Contesto messaggio" a pagina 48.

Avvio automatico dei programmi IBM MQ

Utilizzare IBM MQ *trigger* per avviare un programma automaticamente quando i messaggi arrivano su una coda.

È possibile impostare condizioni di trigger su una coda in modo che un programma inizi ad elaborare tale coda:

- Ogni volta che un messaggio arriva sulla coda
- Quando arriva il primo messaggio sulla coda
- Quando il numero di messaggi sulla coda raggiunge un numero predefinito

Per ulteriori informazioni sull'attivazione, consultare "Avvio delle applicazioni IBM MQ utilizzando i trigger" a pagina 866. L'attivazione è solo un modo per avviare un programma automaticamente. Ad esempio, è possibile avviare un programma automaticamente su un timer utilizzando funzioni nonIBM MQ .

Multi Su Multiplatforms, IBM MQ può definire gli oggetti di servizio per avviare i programmi IBM MQ all'avvio del gestore code; consultare Oggetti di servizio.

Generazione di report IBM MQ

È possibile richiedere i seguenti report all'interno di un'applicazione:

- Report di eccezioni
- Report di scadenza
- Report COA (Confirm - on - arrival)
- Report COD (Confirm - on - delivery)
- Report PAN (positive action notification)
- Report NAN (negative action notification)

Tali regole sono descritte in "Messaggi di report" a pagina 21.

Cluster e affinità di messaggi

Prima di iniziare ad utilizzare i cluster con più definizioni per la stessa coda, esaminare le applicazioni per verificare se vi sono dei cluster che richiedono uno scambio di messaggi correlati.

All'interno di un cluster, un messaggio può essere instradato a qualsiasi gestore code che ospita un'istanza della coda appropriata. Pertanto, la logica delle applicazioni con affinità di messaggi può essere alterata.

Ad esempio, si potrebbero avere due applicazioni che si basano su una serie di messaggi che scorrono tra di loro sotto forma di domande e risposte. Potrebbe essere importante che tutte le domande vengano inviate allo stesso gestore code e che tutte le risposte vengano inviate nuovamente all'altro gestore code. In questa situazione, è importante che la routine di gestione del carico di lavoro non invii i messaggi ad alcun gestore code che ospita un'istanza della coda appropriata.

Laddove possibile, rimuovere le affinità. La rimozione delle affinità dei messaggi migliora la disponibilità e scalabilità delle applicazioni.

Per ulteriori informazioni, consultare Gestione delle affinità dei messaggi.

Considerazioni sulla progettazione e sulle prestazioni per le applicazioni IBM i

Utilizzare queste informazioni per comprendere il modo in cui la progettazione dell'applicazione, i thread e l'archiviazione possono influire sulle prestazioni.

Queste informazioni sono suddivise in due sezioni:

- “Considerazioni sulla progettazione dell'applicazione” a pagina 65
- “Problemi di prestazioni specifici” a pagina 66

Considerazioni sulla progettazione dell'applicazione

Esistono diversi modi in cui la scarsa progettazione del programma può influire sulle prestazioni. Questi problemi possono essere difficili da rilevare perché il programma può sembrare funzionare bene, mentre influisce sulle prestazioni di altre attività. Diversi problemi specifici dei programmi che effettuano chiamate IBM MQ for IBM i sono illustrati nelle seguenti sezioni.

Per ulteriori informazioni sulla progettazione dell'applicazione, consultare “Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51.

Effetto della lunghezza del messaggio
Sebbene IBM MQ for IBM i consenta ai messaggi di contenere fino a 100 MB di dati, la quantità di dati in un messaggio influenza le prestazioni dell'applicazione che elabora il messaggio. Per ottenere le migliori prestazioni dalla tua applicazione, invia solo i dati essenziali in un messaggio; ad esempio, in una richiesta di addebito di un conto bancario, le sole informazioni che potrebbero dover essere trasmesse dal client all'applicazione server sono il numero di conto e l'importo dell'addebito.

Effetto della persistenza del messaggio
I messaggi persistenti vengono registrati su giornale. La registrazione su giornale dei messaggi riduce le prestazioni dell'applicazione, quindi utilizzare i messaggi persistenti solo per i dati essenziali. Se i dati in un messaggio possono essere eliminati se il gestore code si arresta o ha esito negativo, utilizzare un messaggio non persistente.

Ricerca di un particolare messaggio
La chiamata MQGET di solito richiama il primo messaggio da una coda. Se si utilizzano gli identificativi di correlazione e messaggio (*MsgId* e *CorrelId*) nel descrittore del messaggio per specificare un determinato messaggio, il gestore code deve ricercare la coda fino a quando non trova tale messaggio. L'utilizzo della chiamata MQGET in questo modo influisce sulle prestazioni dell'applicazione.

Code che contengono messaggi di lunghezza diversa
Se i messaggi su una coda hanno lunghezze diverse, per determinare la dimensione di un messaggio, l'applicazione può utilizzare la chiamata MQGET con il campo *BufferLength* impostato su zero in modo che, anche se la chiamata ha esito negativo, restituisca la dimensione dei dati del messaggio. L'applicazione può quindi ripetere la chiamata, specificando l'identificativo del messaggio misurato nella prima chiamata e un buffer della dimensione corretta. Tuttavia, se ci sono altre applicazioni che servono la stessa coda, si potrebbe scoprire che le prestazioni dell'applicazione sono ridotte perché la sua seconda chiamata MQGET impiega il tempo per ricercare un messaggio che un'altra applicazione ha richiamato nel tempo tra le due chiamate.

Se l'applicazione non può utilizzare messaggi di lunghezza fissa, un'altra soluzione a questo problema consiste nell'utilizzare la chiamata MQINQ per trovare la dimensione massima dei messaggi che la

coda può accettare, quindi utilizzare questo valore nella chiamata MQGET. La dimensione massima dei messaggi per una coda è memorizzata nell'attributo **MaxMsgLen** della coda. Questo metodo potrebbe utilizzare grandi quantità di memoria, tuttavia, perché il valore di questo attributo della coda può essere il massimo consentito da IBM MQ for IBM i, che potrebbe essere maggiore di 2 GB.

Frequenza dei punti di sincronizzazione
I programmi che emettono numerose chiamate MQPUT all'interno del punto di sincronizzazione, senza eseguirne il commit, possono causare problemi di prestazione. Le code interessate possono riempirsi di messaggi attualmente inutilizzabili, mentre altre attività potrebbero essere in attesa di ricevere questi messaggi. Questo problema ha implicazioni in termini di memoria e in termini di thread collegati alle attività che tentano di ottenere i messaggi.

Utilizzo della chiamata MQPUT1
Utilizzare la chiamata MQPUT1 solo se si dispone di un singolo messaggio da inserire in una coda. Se si desidera inserire più di un messaggio, utilizzare la chiamata MQOPEN, seguita da una serie di chiamate MQPUT e da una singola chiamata MQCLOSE.

Numero di thread in uso
Un'applicazione potrebbe richiedere molti thread. A ciascun elaborazione del gestore code viene assegnato un numero di thread massimo consentito. Se alcune applicazioni sono problematiche, potrebbe essere dovuto alla loro progettazione utilizzando troppi thread. Considerare se l'applicazione prende in considerazione questa possibilità e se intraprende azioni per arrestare o notificare questo tipo di ricorrenza. Il numero massimo di thread consentiti da IBM i è 4.095. Tuttavia, il valore predefinito è 64. IBM MQ rende disponibili fino a 63 thread per i propri processi.

Problemi di prestazioni specifici

Questa sezione spiega i problemi di storage e le scarse prestazioni.

Problemi di memoria
Se si riceve il messaggio di sistema `CPF0907. Serious storage condition may exist` è possibile che si stia riempiendo lo spazio associato ai gestori code IBM MQ for IBM i .

L'applicazione o IBM MQ for IBM i è in esecuzione lentamente?
Se l'applicazione è in esecuzione lentamente, potrebbe indicare che si trova in un loop o in attesa di una risorsa non disponibile. Questa esecuzione lenta potrebbe anche essere causata da un problema di prestazioni. Forse è perché il sistema sta funzionando vicino ai limiti della sua capacità. Questo tipo di problema è probabilmente peggiore nei periodi di picco di carico del sistema, in genere a metà mattina e metà pomeriggio. (Se la rete si estende su più di un fuso orario, il carico di sistema di picco potrebbe sembrare che si verifichi in un altro momento.)

Se si rileva che la riduzione delle prestazioni non dipende dal caricamento del sistema, ma accade a volte quando il sistema è leggermente caricato, è probabile che la colpa sia di un programma applicativo mal progettato. Questo problema potrebbe manifestarsi come un problema che si verifica solo quando si accede a determinate code.

QTOTJOB e QADLTOTJ sono valori di sistema che vale la pena analizzare.

I seguenti sintomi potrebbero indicare che IBM MQ for IBM i è in esecuzione lentamente:

- Se il sistema è lento a rispondere ai comandi MQSC.
- Se visualizzazioni ripetute della profondità della coda indicano che la coda viene elaborata lentamente per un'applicazione con cui si prevede una grande quantità di attività della coda.
- La traccia IBM MQ è in esecuzione?

Linux Considerazioni sulla progettazione per applicazioni Linux on Power Systems - Little Endian

Poiché Linux on Power Systems - Little Endian supporta solo applicazioni a 64 bit, non è disponibile alcun supporto in IBM MQ per le applicazioni a 32 bit.

Concetti correlati
“Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51

Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

z/OS Considerazioni sulla progettazione e sulle prestazioni per le applicazioni z/OS

La progettazione delle applicazioni è uno dei principali fattori che influiscono sulle prestazioni. Utilizzare questo argomento per comprendere alcuni dei fattori di progettazione coinvolti nelle prestazioni.

Esistono diversi modi in cui la scarsa progettazione del programma può influire sulle prestazioni. Questi problemi possono essere difficili da rilevare perché il programma può sembrare funzionare bene, mentre influisce sulle prestazioni di altre attività. Nelle seguenti sezioni vengono illustrati diversi problemi specifici dei programmi che effettuano chiamate MQI.

Per ulteriori informazioni sulla progettazione dell'applicazione, consultare “Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51.

Effetto della lunghezza del messaggio

Sebbene IBM MQ for z/OS consenta ai messaggi di contenere fino a 100 MB di dati, la quantità di dati in un messaggio influenza le prestazioni dell'applicazione che elabora il messaggio. Per ottenere le migliori prestazioni dalla tua applicazione, invia solo i dati essenziali in un messaggio. Ad esempio, in una richiesta di addebito di un conto bancario, le uniche informazioni che potrebbero dover essere inoltrate dal client all'applicazione server sono il numero di conto e l'importo da addebitare.

Effetto della persistenza del messaggio

Vengono registrati i messaggi persistenti. La registrazione dei messaggi riduce le prestazioni della tua applicazione, quindi utilizza i messaggi persistenti solo per i dati essenziali. Se i dati in un messaggio possono essere eliminati se il gestore code si arresta o ha esito negativo, utilizzare un messaggio non persistente.

I dati per i messaggi persistenti vengono scritti nei buffer di log. Questi buffer vengono scritti nei dataset di log quando:

- Si verifica un commit
- Un messaggio è stato ricevuto o inserito fuori dal punto di sincronizzazione
- I buffer WRTHRSH sono pieni

L'elaborazione di molti messaggi in un'unità di lavoro può causare meno input / output che se i messaggi fossero elaborati uno per ogni unità di lavoro o fuori dal punto di sincronizzazione.

Ricerca di un particolare messaggio

La chiamata MQGET generalmente richiama il primo messaggio da una coda. Se si utilizzano il messaggio e gli identificatori di correlazione (**MsgId** e **CorrelId**) nel descrittore di messaggi per specificare uno specifico messaggio, il gestore code ricerca la coda finché non trova tale messaggio. L'uso di MQGET in questo modo influisce sulle prestazioni della tua applicazione perché, per trovare un particolare messaggio, IBM MQ potrebbe dover eseguire la scansione dell'intera coda.

È possibile utilizzare l'attributo della coda **IndexType** per specificare che si desidera che il gestore code mantenga un indice che può essere utilizzato per aumentare la velocità delle operazioni MQGET sulla coda. Tuttavia, vi è una piccola riduzione delle prestazioni per la gestione di un indice, quindi generarne uno solo se è necessario utilizzarlo. È possibile scegliere di creare un indice di identificatori di messaggi o di identificatori di correlazione oppure è possibile scegliere di non creare un indice per le code in cui i messaggi vengono richiamati in modo sequenziale. Provare ad avere molti valori chiave diversi, non lotti con lo stesso valore. Ad esempio Balance1, Balance2e Balance3, non tre con Balance. Per

le code condivise, è necessario disporre del **IndexType**corretto. Per i dettagli dell'attributo della coda **IndexType** , vedere IndexType.

Per evitare di influenzare il tempo di riavvio del gestore code utilizzando code indicizzate, utilizzare il parametro QINDXBLD (NOWAIT) nella macro CSQ6SYSP . Ciò consente il completamento del riavvio del gestore code senza attendere il termine della generazione dell'indice della coda.

Per una descrizione completa dell'attributi **IndexType** e di altri attributi oggetto, consultare Attributi degli oggetti.

Code che contengono messaggi di lunghezza diversa

Richiamare un messaggio, utilizzando una dimensione buffer corrispondente alla dimensione prevista del messaggio. Se si riceve il codice di ritorno che indica che il messaggio è troppo lungo, ottenere un buffer più grande. Quando il richiamo non riesce in questo modo, la lunghezza dei dati restituita è la dimensione dei dati del messaggio non convertiti. Se si specifica MQGMO_CONVERT sulla chiamata MQGET e i dati si espandono durante la conversione, è possibile che non rientrino nel buffer, nel qual caso è necessario aumentare ulteriormente la dimensione del buffer.

Se si immette MQGET con una lunghezza del buffer pari a zero, viene restituita la dimensione del messaggio e l'applicazione può ottenere un buffer di questa dimensione ed emettere nuovamente il comando get. Se si dispone di più applicazioni che elaborano la coda, un'altra applicazione potrebbe aver già elaborato il messaggio quando l'applicazione originale ha riemesso il richiamo. Se di tanto in tanto si dispone di messaggi di grandi dimensioni, potrebbe essere necessario ottenere un buffer di grandi dimensioni solo per questi messaggi e rilasciarlo dopo che il messaggio è stato elaborato. Ciò consente di ridurre i problemi di memoria virtuale se tutte le applicazioni dispongono di buffer di grandi dimensioni.

Se l'applicazione non può utilizzare messaggi di lunghezza fissa, un'altra soluzione a questo problema consiste nell'utilizzare la chiamata MQINQ per trovare la dimensione massima dei messaggi che la coda può accettare, quindi utilizzare questo valore nella chiamata MQGET . La dimensione massima dei messaggi per una coda è memorizzata nell'attributo **MaxMsgL** della coda. Tuttavia, questo metodo potrebbe utilizzare grandi quantità di memoria, poiché il valore di **MaxMsgL** potrebbe essere pari a 100 MB, il massimo consentito da IBM MQ for z/OS.

Nota: È possibile ridurre il parametro **MaxMsgL** dopo che i messaggi di grandi dimensioni sono stati inseriti nella coda. Ad esempio, è possibile inserire un messaggio da 100 MB, quindi impostare **MaxMsgL** su 50 byte. Ciò significa che è ancora possibile ottenere messaggi più grandi di quelli previsti dall'applicazione.

Frequenza dei punti di sincronizzazione

I programmi che emettono molte chiamate MQPUT nel punto di sincronizzazione, senza eseguirne il commit, possono causare problemi di prestazioni. Le code interessate possono riempirsi di messaggi attualmente inutilizzabili, mentre altre attività potrebbero essere in attesa di ricevere questi messaggi. Ciò ha implicazioni in termini di memoria, e in termini di thread collegati con le attività che stanno tentando di ottenere messaggi.

Come regola se si dispone di più applicazioni che elaborano una coda, generalmente si ottengono le migliori prestazioni quando si dispone di

- 100 messaggi brevi (meno di 1 KB) oppure
- Un messaggio per messaggi più grandi (100 KB)

per ogni punto di sincronizzazione. Se c'è una sola applicazione che elabora la coda, è necessario avere più messaggi per ogni unità di lavoro.

È possibile limitare il numero di messaggi che un'attività può ricevere o inserire all'interno di una singola unità di recupero con l'attributo del gestore code **MAXUMSGS** . Per informazioni su questo attributo, consultare il comando **ALTER QMGR** in Comandi MQSC.

Vantaggi della chiamata MQPUT1

Utilizzare la chiamata MQPUT1 solo se si dispone di un singolo messaggio da inserire in una coda. Se si desidera inserire più di un messaggio, utilizzare la chiamata MQOPEN , seguita da una serie di chiamate MQPUT e da una singola chiamata MQCLOSE .

Il numero di messaggi che un gestore code può contenere

Code locali

Il numero di messaggi locali che un gestore code può contenere è fondamentalmente la dimensione delle serie di pagine. È possibile avere fino a 100 serie di pagine (anche se è consigliata la serie di pagine 0 e la serie di pagine 1 sono per le code e gli oggetti correlati al sistema). È possibile utilizzare una serie di pagine con formato esteso e aumentare la capacità di una serie di pagine.

Code condivise

La capacità delle code condivise dipende dalla dimensione della CF (coupling facility). IBM MQ utilizza le strutture di elenco CF in cui le unità di memoria fondamentali sono voci ed elementi. Ogni messaggio viene memorizzato come una voce e più elementi contenenti l'MQMD associato e altri dati del messaggio. Il numero di elementi utilizzati da un singolo messaggio dipende dalla dimensione del messaggio e, per CFLEVEL (5), dalle regole di offload in vigore al momento di MQPUT. Sono necessari meno elementi quando i dati del messaggio sono scaricati su Db2 o SMDS. L'accesso ai dati del messaggio è più lento quando è stato eseguito l'offload del messaggio. Consultare Performance Supportpac MP1H per un ulteriore confronto delle prestazioni e del sovraccarico della CPU associato all'offload del messaggio.

Cosa influenza le prestazioni

Le prestazioni possono indicare la velocità di elaborazione dei messaggi e la quantità di CPU necessaria per messaggio.

Ciò che influisce sulla velocità di elaborazione dei messaggi

Per i messaggi persistenti, l'impatto maggiore è la velocità dei dataset di log. La velocità dei dataset di log dipende dalla DASD su cui si trovano. Pertanto, è necessario prestare attenzione a inserire la serie di dati di log su volumi utilizzati bassi per ridurre il conflitto. Lo striping dei log di MQ migliora le prestazioni del log quando sono presenti più pagine scritte per I/O. La connessione a fibre ottiche ad alte prestazioni Z (zHPF) ha anche prestazioni significative per il tempo di risposta I/O quando il sottosistema I/O è occupato.

Quando è presente una richiesta di ricezione e inserimento di un messaggio, l'accesso alla coda viene bloccato durante la richiesta per preservare l'integrità della coda. Per scopi di pianificazione, considerare la coda bloccata per l'intera richiesta. Quindi, se il tempo per un put è di 100 microsecondi e hai più di 10.000 richieste al secondo, potresti riscontrare dei ritardi. Si potrebbe ottenere di più di questo in pratica, ma è una buona regola generale. È possibile utilizzare code differenti per migliorare le prestazioni.

I possibili motivi possono essere:

- utilizzare una coda di risposta comune utilizzata da ogni transazione CICS
- ogni transazione CICS riceve una risposta univoca alla coda
- una risposta a una coda per una regione CICS e tutte le transazioni nella regione CICS utilizzano questa coda.

La risposta dipende dal numero di richieste al secondo e dal tempo di risposta delle richieste.

Se i messaggi devono essere letti da una serie di pagine, saranno più lenti rispetto a quando i messaggi si trovano nel pool di buffer. Se si dispone di più messaggi che si adattano a un pool di

buffer, questi verranno trasmessi al disco. Quindi è necessario assicurarsi che il pool di buffer sia abbastanza grande per i messaggi di breve durata. Se si dispone di messaggi che vengono elaborati molte ore dopo, è probabile che si riversino su disco, quindi è consigliabile prevedere un richiamo per questi messaggi più lento rispetto a quando si trovavano nel pool di buffer.

Per una coda condivisa, la velocità dei messaggi dipende dalla velocità della funzione di accoppiamento. Una CF all'interno del processore fisico è probabilmente più veloce di una CF esterna. Il tempo di risposta CF dipende da quanto è occupato il CF. Ad esempio sui sistemi Hursley, quando il CF era occupato al 17%, il tempo di risposta era di 14 microsecondi. Quando la CF era occupata al 95%, il tempo di risposta era di 45 microsecondi.

Se le richieste MQ utilizzano molta CPU, ciò può influire sulla velocità di elaborazione dei messaggi. Poiché se la partizione logica (LPAR) è vincolata per la CPU, le applicazioni verranno ritardate in attesa di CPU.

Quantità di CPU per messaggio

In generale, i messaggi più grandi utilizzano più CPU, quindi evita messaggi di grandi dimensioni (x MB), se possibile.

Quando si ricevono messaggi specifici dalle code, la coda deve essere indicizzata in modo che il gestore code possa andare direttamente al messaggio (in modo da evitare potenzialmente un'intera scansione della coda). Se la coda non è indicizzata, viene eseguita la scansione della coda dall'inizio della ricerca del messaggio. Se ci sono 1000 messaggi nella coda, potrebbe essere necessario eseguire la scansione di tutti i 1000 messaggi. Il risultato è un sacco di utilizzo di CPU non necessario.

I canali che utilizzano TLS hanno un costo aggiuntivo dovuto alla crittografia del messaggio.

In MQ V7 è possibile selezionare i messaggi mediante una stringa del selettore in aggiunta a **CORRELID** o **MSGID**. Ogni messaggio deve essere cercato, quindi se ci sono molti messaggi nella coda questo è costoso.

È più efficiente per un'applicazione eseguire OPEN PUT PUT CLOSE rispetto a PUT1 PUT1.

Attivazione in CICS

Quando la frequenza di arrivo dei messaggi per una coda attivata è bassa, è efficiente utilizzare prima il trigger. Quando la frequenza di arrivo dei messaggi è superiore a 10 messaggi al secondo, è più efficiente attivare la prima transazione, quindi fare in modo che la transazione elabori un messaggio e ottenga il messaggio successivo e così via. Se un messaggio non è arrivato in un breve periodo (ad esempio, tra 0.1 e 1 secondo), la transazione termina. Con un livello di prestazione elevato, è possibile che siano necessarie più transazioni in esecuzione per elaborare i messaggi e impedire un accumulo di messaggi. Per ogni messaggio di trigger prodotto, ciò richiede un put e un get di un messaggio di trigger, che in realtà raddoppia il costo del messaggio.

Quante connessioni o utenti simultanei sono supportati

Ogni connessione utilizza la memoria virtuale all'interno del gestore code, in modo che più utenti simultanei utilizzano più memoria. Se è necessario un pool di buffer molto grande e un numero elevato di utenti, è possibile che l'archiviazione virtuale sia limitata e che sia necessario ridurre la dimensione dei pool di buffer.

Se si sta utilizzando la sicurezza, il gestore code memorizza nella cache le informazioni all'interno del gestore code per un lungo periodo. Viene influenzata la quantità di memoria virtuale utilizzata all'interno del gestore code.

CHINIT può supportare fino a circa 10.000 connessioni. Ciò è limitato dalla memoria virtuale. Se una connessione utilizza più spazio di archiviazione, ad esempio utilizzando TLS, l'archiviazione per connessione aumenta, il che significa che **CHINIT** può supportare meno connessioni. Se si stanno elaborando messaggi di grandi dimensioni, questi richiederanno più memoria per i buffer in **CHINIT**, in modo che **CHINIT** possa supportare meno messaggi.

Le connessioni a un gestore code remoto sono più efficienti delle connessioni client. Ad esempio, ogni richiesta del client MQ richiede due flussi di rete (uno per la richiesta e uno per la risposta). Con un canale a un gestore code remoto, potrebbero esserci 50 invii sulla rete prima che ritorni una risposta.

Se si sta considerando una rete client di grandi dimensioni, potrebbe essere più efficiente utilizzare un gestore code concentratore su una casella distribuita e avere un canale in entrata e in uscita dal concentratore.

Altre cose che influenzano le prestazioni

La dimensione del dataset di log deve essere almeno 1000 cilindri. Se i log sono più piccoli, l'attività del punto di controllo potrebbe essere troppo frequente. Su un sistema occupato, un punto di controllo in genere dovrebbe essere ogni 15 minuti o più lungo, con velocità di trasmissione molto elevate potrebbe essere inferiore a questo. Quando si verifica un checkpoint, i pool di buffer vengono sottoposti a scansione e i messaggi 'vecchi' e le pagine modificate vengono scritti sul disco. Se i punti di controllo sono troppo frequenti, ciò può influire sulle prestazioni. Il valore di LOGLOAD può influire anche sulla frequenza del punto di controllo. Se il gestore code termina in modo anomalo, al riavvio potrebbe dover leggere di nuovo su 3 punti di controllo. L'intervallo di checkpoint migliore è un bilanciamento tra l'attività quando viene eseguito un checkpoint e la quantità di dati di log che potrebbe essere necessario leggere al riavvio del gestore code.

Si è verificato un sovraccarico significativo durante l'avvio di un canale. Di solito è meglio avviare un canale e lasciarlo connesso, piuttosto che avvii e arresti frequenti del canale.

Informazioni correlate

MP1K: IBM MQ for z/OS 9.0 Report delle prestazioni

z/OS Applicazioni bridge IMS e IMS su IBM MQ for z/OS

Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

- Per utilizzare i syncpoint e le chiamate MQI nelle applicazioni IMS , consultare “Scrittura di applicazioni IMS utilizzando IBM MQ” a pagina 72.
- Per scrivere le applicazioni che utilizzano il bridge IBM MQ - IMS , consultare “Scrittura delle applicazioni bridge IMS” a pagina 76.

Utilizzare i seguenti link per ulteriori informazioni sulle applicazioni bridge IMS e IMS su IBM MQ for z/OS:

- “Scrittura di applicazioni IMS utilizzando IBM MQ” a pagina 72
- “Scrittura delle applicazioni bridge IMS” a pagina 76

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866

Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

z/OS Scrittura di applicazioni IMS utilizzando IBM MQ

Ci sono ulteriori considerazioni quando si utilizza IBM MQ nelle applicazioni IMS , tra cui quali chiamate API di MQ possono essere utilizzate e il meccanismo utilizzato per il punto di sincronizzazione.

Utilizzare i seguenti link per ulteriori informazioni sulla scrittura di applicazioni IMS su IBM MQ for z/OS:

- “Punti di sincronizzazione nelle applicazioni IMS” a pagina 72
- “Chiamate MQI nelle applicazioni IMS” a pagina 73

Limitazioni

Esistono delle limitazioni su cui le chiamate API IBM MQ possono essere utilizzate da un'applicazione utilizzando l'adattatore IMS .

Le seguenti chiamate API IBM MQ non sono supportate in una applicazione che utilizza l'adattatore IMS :

- MQCB
- MQCB_FUNZIONE
- MQCTL

Concetti correlati

“Scrittura delle applicazioni bridge IMS” a pagina 76
Questo argomento contiene informazioni sulla scrittura di applicazioni per utilizzare il bridge IBM MQ - IMS .

z/OS *Punti di sincronizzazione nelle applicazioni IMS*

In un'applicazione IMS , si stabilisce un punto di sincronizzazione utilizzando chiamate IMS come GU (get unique) a IOPCB e CHKP (checkpoint).

Per annullare tutte le modifiche dal punto di controllo precedente, è possibile utilizzare la chiamata IMS ROLB (rollback). Per ulteriori informazioni, vedi ROLB call nella documentazione di IMS .

Il gestore code è un partecipante in un protocollo di commit a due fasi; il gestore del punto di sincronizzazione IMS è il coordinatore.

Tutte le maniglie aperte vengono chiuse dall'adattatore IMS in un punto di sincronizzazione (tranne in un ambiente BMP batch o non basato su messaggi). Ciò è dovuto al fatto che un utente differente potrebbe avviare la successiva unità di lavoro e il controllo di sicurezza di IBM MQ viene eseguito quando vengono effettuate le chiamate MQCONN, MQCONNX e MQOPEN, non quando vengono effettuate le chiamate MQPUT o MQGET.

Tuttavia, in un ambiente WFI (Wait - for - Input) o PWFI (pseudo Wait - for - Input) IMS non notifica IBM MQ di chiudere gli handle fino a quando non arriva il messaggio successivo o non viene restituito un codice di stato QC all'applicazione. Se l'applicazione è in attesa nella regione IMS e uno di questi handle appartiene a code attivate, l'attivazione non si verificherà perché le code sono aperte. Per questo motivo, le applicazioni in esecuzione in un ambiente WFI o PWFI devono esplicitamente MQCLOSE che la coda gestisce prima di eseguire il GU all'IOPCB per il messaggio successivo.

Se un'applicazione IMS (BMP o MPP) emette la chiamata MQDISC, le code aperte vengono chiuse ma non viene utilizzato alcun punto di sincronizzazione implicito. Se l'applicazione termina normalmente, tutte le code aperte vengono chiuse e si verifica un commit implicito. Se l'applicazione termina in modo anomalo, tutte le code aperte vengono chiuse e si verifica un backout implicito.

z/OS *Chiamate MQI nelle applicazioni IMS*

Utilizzare queste informazioni per informazioni sull'utilizzo delle chiamate MQI sulle applicazioni Server e sulle applicazioni di interrogazione.

Questa sezione riguarda l'utilizzo delle chiamate MQI nei seguenti tipi di applicazioni IMS :

- "Applicazioni del server" a pagina 73
- "Applicazioni di richiesta" a pagina 75

Applicazioni del server

Di seguito viene riportato uno schema del modello di applicazione server MQI:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

Il programma di esempio CSQ4ICB3 mostra l'implementazione, in C/370, di un BMP che utilizza questo modello. Il programma stabilisce prima la comunicazione con IMS e poi con IBM MQ:

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

L'inizializzazione di IMS determina se il programma è stato richiamato come un BMP orientato ai messaggi o batch e controlla la connessione del gestore code IBM MQ e gli handle di coda di conseguenza:

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
```

```
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

L'inizializzazione IBM MQ si connette al gestore code e apre le code. In un BMP basato sui messaggi, questo viene richiamato dopo ogni punto di sincronizzazione IMS ; in un BMP orientato al batch, questo viene richiamato solo durante l'avvio del programma:

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

L'implementazione del modello server in un MPP è influenzata dal fatto che l'MPP elabora una singola unità di lavoro per chiamata. Ciò si verifica perché, quando viene utilizzato un punto di sincronizzazione (GU), gli handle di connessione e di coda vengono chiusi e viene consegnato il successivo messaggio IMS . Questa limitazione può essere parzialmente superata da uno dei seguenti:

- **Elaborazione di molti messaggi in una singola unità di lavoro**

  Ciò comporta:

  – Lettura di un messaggio
  – Elaborazione degli aggiornamenti richiesti
  – Inserimento della risposta

  in un loop fino a quando non sono stati elaborati tutti i messaggi o fino a quando non è stato elaborato un numero massimo impostato di messaggi, in cui viene utilizzato un punto di sincronizzazione.

  Solo alcuni tipi di applicazione (ad esempio, un semplice aggiornamento del database o una richiesta di informazioni) possono essere avvicinati in questo modo. Sebbene i messaggi di risposta MQI possano essere inseriti con l'autorizzazione del creatore del messaggio MQI gestito, le implicazioni di sicurezza di qualsiasi aggiornamento delle risorse IMS devono essere affrontate con attenzione.
- **Elaborazione di un messaggio per richiamo di MPP e garanzia della pianificazione multipla di MPP per elaborare tutti i messaggi disponibili.**

  Utilizzare il programma di controllo trigger IBM MQ IMS (CSQQTRMN) per pianificare la transazione MPP quando ci sono messaggi sulla coda IBM MQ e nessuna applicazione che la supporta.

Se il controllo dei trigger avvia MPP, il nome del gestore code e il nome della coda vengono passati al programma, come mostrato nel seguente estratto di codice COBOL:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

Il modello server, che si prevede sia un'attività di lunga durata, è meglio supportato in una regione di elaborazione batch, anche se BMP non può essere attivato utilizzando CSQQTRMN.

Applicazioni di richiesta

Un'applicazione tipica IBM MQ che avvia un'interrogazione o un aggiornamento funziona nel modo seguente:

- Raccogliere dati dall'utente
- Inserire uno o più messaggi IBM MQ
- Richiamare i messaggi di risposta (potrebbe essere necessario attenderli)
- Fornire una risposta all'utente

Poiché i messaggi inseriti nelle code IBM MQ non diventano disponibili per altre applicazioni IBM MQ fino a quando non ne viene eseguito il commit, devono essere inseriti fuori dal punto di sincronizzazione oppure l'applicazione IMS deve essere suddivisa in due transazioni.

Se l'interrogazione implica l'inserimento di un singolo messaggio, è possibile utilizzare l'opzione *nessun punto di sincronizzazione* ; tuttavia, se l'interrogazione è più complessa o gli aggiornamenti delle risorse sono coinvolti, è possibile che si verifichino problemi di congruenza se si verifica un errore e non si utilizza il punto di sincronizzazione.

Per ovviare a questo problema, è possibile suddividere le transazioni IMS MPP utilizzando chiamate MQI utilizzando un commutatore di messaggi da programma a programma; per informazioni, consultare *IMS ISC (Intersystem Communication)* . Ciò consente a un programma di interrogazione di essere implementato in un MPP:

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
```

```
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS Scrittura delle applicazioni bridge IMS

Questo argomento contiene informazioni sulla scrittura di applicazioni per utilizzare il bridge IBM MQ - IMS .

Per informazioni sul bridge IBM MQ - IMS , vedere Il bridge IMS.

Utilizzare i seguenti link per ulteriori informazioni sulla scrittura di applicazioni bridge IMS su IBM MQ for z/OS:

- “Come il bridge IMS gestisce i messaggi” a pagina 76
- “Scrittura di programmi di transazione IMS tramite IBM MQ” a pagina 913

Concetti correlati

“Scrittura di applicazioni IMS utilizzando IBM MQ” a pagina 72
Ci sono ulteriori considerazioni quando si utilizza IBM MQ nelle applicazioni IMS , tra cui quali chiamate API di MQ possono essere utilizzate e il meccanismo utilizzato per il punto di sincronizzazione.

z/OS *Come il bridge IMS gestisce i messaggi*

Quando si utilizza il bridge IBM MQ - IMS per inviare i messaggi a un'applicazione IMS , è necessario creare i messaggi in un formato speciale.

È inoltre necessario inserire i messaggi nelle code IBM MQ che sono state definite con una classe di memoria che specifica il gruppo XCF e il nome membro del sistema IMS di destinazione. Queste sono note come code bridge MQ-IMS o semplicemente code **bridge** .

Il bridge IBM MQ-IMS richiede l'accesso di immissione esclusivo (MQOO_INPUT_EXCLUSIVE) alla coda bridge se è definita con QSGDISP (QMGR) o se è definita con QSGDISP (SHARED) insieme all'opzione NOSHARE.

Un utente non ha bisogno di collegarsi a IMS prima di inviare messaggi a una applicazione IMS . L'ID utente nel campo *UserIdentifier* della struttura MQMD viene utilizzato per il controllo di sicurezza. Il livello di controllo viene determinato quando IBM MQ si connette a IMSed è descritto in Application access control for IMS bridge. Ciò consente l'implementazione di uno pseudo accesso.

Il bridge di IBM MQ - IMS accetta i seguenti tipi di messaggio:

- Messaggi contenenti i dati della transazione IMS e una struttura MQIIH (descritta in MQIIH):

  ```
  MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
  ```

  Nota:

  1. Le parentesi quadre, [], rappresentano segmenti multipli facoltativi.
  2. Impostare il campo *Format* della struttura di MQMD su MQFMT_IMS per utilizzare la struttura MQIIH.
- Messaggi contenenti dati di transazione IMS ma nessuna struttura MQIIH:

  ```
  LLZZ<trancode><data> \
  [LLZZ<data>][LLZZ<data>]
  ```

IBM MQ convalida i dati del messaggio per garantire che la somma dei byte LL più la lunghezza di MQIIH (se presente) sia uguale alla lunghezza del messaggio.

Quando il bridge IBM MQ - IMS richiama i messaggi dalle code bridge, li elabora nel modo seguente:

- Se il messaggio contiene una struttura MQIIH, il bridge verifica MQIIH (vedere MQIIH), crea le intestazioni OTMA e invia il messaggio a IMS. Il codice di transazione è specificato nel messaggio di input. Se si tratta di un LTERM, IMS risponde con un messaggio DFS1288E . Se il codice della transazione rappresenta un comando, IMS esegue il comando; altrimenti il messaggio viene accodato in IMS per la transazione.
- Se il messaggio contiene dati di transazione IMS , ma non una struttura MQIIH, il bridge IMS fa i seguenti presupposti:
  - Il codice transazione è espresso in byte da 5 a 12 dei dati utente
  - La transazione è in modalità non conversazionale
  - La transazione è in modalità di commit 0 (commit - e - invio)
  - Il *Format* in MQMD viene utilizzato come *MFSMapName* (in input)
  - La modalità di protezione è MQISS_CHECK

  Anche il messaggio di risposta viene creato senza una struttura MQIIH, prendendo *Format* per MQMD dall' *MFSMapName* dell'output IMS .

Il bridge IBM MQ - IMS utilizza uno o due Tpipe per ogni coda IBM MQ :

- Un Tpipe sincronizzato viene utilizzato per tutti i messaggi che utilizzano la modalità Commit 0 (COMMIT_THEN_SEND) (questi vengono visualizzati con SYN nel campo di stato del comando TPIPE xxxx del client TMEMBER IMS /DIS)
- Un Tpipe non sincronizzato viene utilizzato per tutti i messaggi che utilizzano la modalità Commit 1 (SEND_THEN_COMMIT)

I Tpipe vengono creati da IBM MQ quando vengono utilizzati per la prima volta. Esiste un Tpipe non sincronizzato fino a quando IMS non viene riavviato. I Tpipe sincronizzati esistono fino a quando IMS non viene avviato a freddo. Non è possibile eliminare questi Tpipe.

Per ulteriori informazioni su come il bridge IBM MQ - IMS gestisce i messaggi, consultare i seguenti argomenti:

- “Associazione di messaggi IBM MQ a tipi di transazioni IMS” a pagina 77
- “Se il messaggio non può essere inserito nella coda IMS” a pagina 78
- “Codici di feedback del bridge IMS” a pagina 78
- “I campi MQMD nei messaggi dal bridge IMS .” a pagina 79
- “I campi MQIIH nei messaggi dal bridge IMS” a pagina 80
- “Messaggi di risposta da IMS” a pagina 81
- “Utilizzo di PCB di risposta alternativi in transazioni IMS” a pagina 81
- “Invio di messaggi non richiesti da IMS” a pagina 81
- “Segmentazione del messaggio” a pagina 82
- “Conversione dei dati per i messaggi da e verso il bridge IMS” a pagina 82

Concetti correlati

“Scrittura di programmi di transazione IMS tramite IBM MQ” a pagina 913
La codifica richiesta per gestire le transazioni IMS tramite IBM MQ dipende dal formato del messaggio richiesto dalla transazione IMS e dall'intervallo di risposte che può restituire. Tuttavia, ci sono diversi punti da considerare quando la tua applicazione gestisce le informazioni di formattazione dello schermo IMS .

z/OS *Associazione di messaggi IBM MQ a tipi di transazioni IMS*

Una tabella che descrive l'associazione dei messaggi IBM MQ ai tipi di transazione IMS .

Tabella 4. Modalità di associazione dei messaggi IBM MQ ai tipi di transazione IMS		
IBM MQTipo messaggio	**Commit - then - send (modalità 0) - utilizza Tpipe IMS sincronizzati**	**Send - then - commit (modalità 1) - utilizza Tpipe IMS non sincronizzati**
Messaggi IBM MQ persistenti	• Transazioni con funzioni complete recuperabili • Le transazioni non recuperabili vengono rifiutate da IMS	• Transazioni Fastpath • Transazioni conversazionali • Transazioni con funzioni complete
Messaggi IBM MQ non persistenti	• Transazioni con funzioni complete non recuperabili • Le transazioni ripristinabili sono consentite con IMS V8 e APAR PQ61404 e tutte le versioni successive di IMS	• Transazioni Fastpath • Transazioni conversazionali • Transazioni con funzioni complete

Nota: I comandi IMS non possono utilizzare messaggi IBM MQ persistenti con modalità di commit 0. Per ulteriori informazioni, consultare Modalità commit (commitMode) .

z/OS *Se il messaggio non può essere inserito nella coda IMS*

Informazioni sulle azioni da intraprendere se il messaggio non può essere inserito nella coda IMS .

Se il messaggio non può essere inserito nella coda IMS , la seguente azione viene eseguita da IBM MQ:

- Se un messaggio non può essere inserito in IMS perché non è valido, viene inserito nella coda di messaggi non recapitabili e un messaggio viene inviato alla console di sistema.
- Se il messaggio è valido, ma è rifiutato da IMS, IBM MQ invia un messaggio di errore alla console di sistema, il messaggio include il codice di condizione IMS e il messaggio IBM MQ viene inserito nella coda dei messaggi non instradabili. Se il codice di rilevamento IMS è 001A, IMS invia un messaggio IBM MQ contenente il motivo dell'esito negativo alla coda di risposta.

  Nota: Nelle circostanze elencate in precedenza, se IBM MQ non è in grado di inserire il messaggio nella coda di messaggi non instradabili per qualsiasi motivo, il messaggio viene restituito alla coda IBM MQ di origine. Un messaggio di errore viene inviato alla console di sistema e non vengono inviati ulteriori messaggi da tale coda.

  Per inviare di nuovo i messaggi, eseguire ***uno*** dei seguenti:

  – Arrestare e riavviare i Tpipe in IMS corrispondenti alla coda
  – Modificare la coda in GET (DISABLED) e di nuovo in GET (ENABLED)
  – Arrestare e riavviare IMS o OTMA
  – Arrestare e riavviare il sottosistema IBM MQ
- Se il messaggio viene rifiutato da IMS per un messaggio diverso da un errore, il messaggio IBM MQ viene restituito alla coda di origine, IBM MQ arresta l'elaborazione della coda e viene inviato un messaggio di errore alla console di sistema.

  Se è richiesto un messaggio di report di eccezioni, il bridge lo inserisce nella coda di risposta con l'autorizzazione del creatore. Se il messaggio non può essere inserito nella coda, il messaggio di prospetto viene inserito nella coda di messaggi non recapitabili con l'autorizzazione del bridge. Se non è possibile inserirlo nella DLQ, viene eliminato.

z/OS *Codici di feedback del bridge IMS*

I codici di rilevamento IMS vengono generalmente emessi in formato esadecimale nei messaggi della console IBM MQ come CSQ2001I (ad esempio, codice di rilevamento 0x001F). I codici di feedback IBM

MQ come visualizzati nell'intestazione dei messaggi non instradabili inseriti nella coda dei messaggi non instradabili sono numeri decimali.

I codici di feedback del bridge IMS sono compresi tra 301 e 399 o tra 600 e 855 per il codice di rilevamento NACK 0x001A. Vengono associati dai codici di rilevamento IMS-OTMA come segue:

1. Il codice di rilevamento IMS-OTMA viene convertito da un numero esadecimale a un numero decimale.
2. 300 viene aggiunto al numero risultante dal calcolo in 1, fornendo il codice IBM MQ *Feedback* .
3. Il codice di rilevamento IMS-OTMA 0x001A, decimale 26 è un caso speciale. Viene generato un codice *Feedback* compreso tra 600 e 855.
   a. Il codice motivo IMS-OTMA viene convertito da un numero esadecimale a un numero decimale.
   b. 600 viene aggiunto al numero risultante dal calcolo in a, fornendo il codice IBM MQ *Feedback* .

Per informazioni sui codici di rilevamento IMS-OTMA, consultare Sense code OTMA per i messaggi NAK.

z/OS *I campi MQMD nei messaggi dal bridge IMS .*
Informazioni sui campi MQMD nei messaggi dal bridge IMS .

L'MQMD del messaggio di origine viene trasmesso da IMS nella sezione Dati utente delle intestazioni OTMA. Se il messaggio ha origine in IMS, viene creato da IMS Destination Resolution Exit. L'MQMD di un messaggio ricevuto da IMS viene creato come segue:

StrucID
"MG"

Versione
MQMD_VERSION_1

Prospetto
MQRO_NONE

MsgType
MQMT_REPLY

Scadenza
Se MQIIH_PASS_EXPIRATION è impostato nel campo Indicatori di MQIIH, questo campo contiene il tempo di scadenza rimanente, altrimenti è impostato su MQEI_UNLIMITED

Feedback
MQFB_NONE

Codifica
MQENC.Native (la codificazione del sistema z/OS)

CodedCharSetId
MQCCSI_Q_MGR (ilSetID CodedChardel sistema z/OS)

Formato
MQFMT_IMS se MQMD.Format del messaggio di input è MQFMT_IMS, altrimenti IOPCB.MODNAME

Priorità
MQMD.Priority del messaggio di input

Persistenza
Dipende dalla modalità di commit: MQMD.Persistence del messaggio di input se CM-1; persistenza corrisponde alla recuperabilità del messaggio IMS se CM-0

MsgId
MQMD.MsgId se MQRO_PASS_MSG_ID, altrimenti New MsgId (impostazione predefinita)

CorrelId
MQMD.CorrelId dal messaggio di input se MQRO_PASS_CORREL_ID, altrimenti MQMD.MsgId dal messaggio di input (impostazione predefinita)

BackoutCount
0

ReplyToQ
Spazi

ReplyToQMgr
Spazi (impostati sul nome qmgr locale dal gestore code durante MQPUT)

UserIdentifier
MQMD.UserIdentifier del messaggio di input

AccountingToken
MQMD.AccountingToken del messaggio di input

ApplIdentityData
MQMD.ApplIdentityData del messaggio di input

PutApplType
MQAT_XCF se non si verifica alcun errore, altrimenti MQAT_BRIDGE

PutApplName
<XCFgroupName> <XCFmemberName> se non si verifica alcun errore, altrimenti il nome QMGR

PutDate
Data in cui il messaggio è stato inserito

PutTime
Ora in cui il messaggio è stato inserito

ApplOriginData
Spazi

z/OS *I campi MQIIH nei messaggi dal bridge IMS*

Informazioni sui campi MQIIH nei messaggi dal bridge IMS .

L'MQIIH di un messaggio ricevuto da IMS viene creato come segue:

StrucId
"IIH"

Versione
1

StrucLength
84

Codifica
MQEN_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Formato
MQIIH.ReplyToFormat del messaggio di input se MQIIH.ReplyToFormat non è vuoto, altrimenti IOPCB.MODNAME

Indicatori
0

LTermOverride
Nome LTERM (Tpipe) dall'intestazione OTMA

MFSMapName
Nome associazione dall'intestazione OTMA

ReplyToFormat
Spazi

Programma di autenticazione
MQIIH.Authenticator del messaggio di input se il messaggio di risposta viene inserito in una coda bridge MQ-IMS , altrimenti spazi vuoti.

TranInstanceId
ID conversazione / Token server dall'intestazione OTMA se in conversazione. Nelle versioni di IMS precedenti a V14, questo campo è sempre null se non è in conversazione. Da IMS V14 in poi, questo campo può essere impostato da IMS anche se non è in conversazione.

TranState
"C" se nella conversazione, altrimenti vuoto

CommitMode
Modalità di commit dall'intestazione OTMA ("0" o "1")

SecurityScope
Spazio

Riservato
Spazio

z/OS *Messaggi di risposta da IMS*

Quando una transazione IMS ISRTs al suo IOPCB, il messaggio viene reinstradato all'LTERM o TPIPE di origine.

Questi sono visualizzati in IBM MQ come messaggi di risposta. I messaggi di risposta da IMS vengono inseriti nella coda di risposta specificata nel messaggio originale. Se il messaggio non può essere inserito nella coda di risposta, viene inserito nella coda di messaggi non recapitabili utilizzando l'autorità del bridge. Se il messaggio non può essere inserito nella coda di messaggi non recapitabili, viene inviato un riconoscimento negativo a IMS per indicare che non è possibile ricevere il messaggio. La responsabilità del messaggio viene quindi restituita a IMS. Se si utilizza la modalità di commit 0, i messaggi da tale Tpipe non vengono inviati al bridge e rimangono nella coda IMS ; in altre parole, non vengono inviati ulteriori messaggi fino al riavvio. Se si sta utilizzando la modalità di commit 1, l'altro lavoro può continuare.

Se la risposta ha una struttura MQIIH, il suo tipo di formato è MQFMT_IMS; in caso contrario, il suo tipo di formato è specificato dal nome MOD IMS utilizzato durante l'inserimento del messaggio.

z/OS *Utilizzo di PCB di risposta alternativi in transazioni IMS*

Quando una transazione IMS utilizza PCB a risposta alternativa (ISRT a ALTPCB o emette una chiamata CHNG a un PCB modificabile), viene richiamata l'exit di pre - instradamento (DFSYPRX0) per stabilire se il messaggio deve essere reinstradato.

Se il messaggio deve essere reinstradato, l'uscita di risoluzione della destinazione (DFSYDRU0) viene richiamata per confermare la destinazione e preparare le informazioni di intestazione. Consultare Utilizzo delle uscite OTMA in IMS e L'uscita di pre - instradamento DFSYPRX0 per informazioni su questi programmi di uscita.

A meno che non venga eseguita un'azione nelle uscite, tutti gli output dalle IMS transazioni avviate da un gestore code IBM MQ , sia nell'IOPCB che in un ALTPCB, verranno restituiti allo stesso gestore code.

z/OS *Invio di messaggi non richiesti da IMS*

Per inviare i messaggi da IMS a una coda IBM MQ , è necessario richiamare una transazione IMS ISRT a un ALTPCB.

È necessario scrivere uscite di pre - instradamento e di risoluzione della destinazione per instradare i messaggi non richiesti da IMS e creare i dati utente OTMA, in modo che l'MQMD del messaggio possa essere creato correttamente. Consultare L'uscita di preinstradamento DFSYPRX0 e L'uscita utente di risoluzione di destinazione per informazioni su questi programmi di uscita.

Nota: Il bridge IBM MQ - IMS non sa se un messaggio ricevuto è una risposta o un messaggio non richiesto. Gestisce il messaggio nello stesso modo in ogni singolo caso, creando MQMD e MQIIH della risposta in base all'OTMA UserData arrivato con il messaggio

I messaggi non richiesti possono creare nuovi Tpipe. Ad esempio, se una transazione IMS esistente passa a un nuovo LTERM (ad esempio PRINT01), ma l'implementazione richiede che l'output venga distribuito tramite OTMA, viene creato un nuovo Tpipe (denominato PRINT01 in questo esempio). Per impostazione predefinita, questo è un Tpipe non sincronizzato. Se l'implementazione richiede che il

messaggio sia ripristinabile, impostare l'indicatore di output dell'uscita di risoluzione di destinazione. Per ulteriori informazioni, consultare il manuale *IMS Customization Guide* .

z/OS *Segmentazione del messaggio*

È possibile definire le transazioni IMS come se prevedesse un input a segmento singolo o multiplo.

L'applicazione IBM MQ di origine deve creare l'input utente che segue la struttura MQIIH come uno o più segmenti LLZZ - data. Tutti i segmenti di un messaggio IMS devono essere contenuti in un singolo messaggio IBM MQ inviato con un singolo MQPUT.

La lunghezza massima di un segmento di dati LLZZ è definita da IMS/OTMA (32767 byte). La lunghezza totale del messaggio IBM MQ è la somma dei byte LL, più la lunghezza della struttura MQIIH.

Tutti i segmenti della risposta sono contenuti in un unico messaggio IBM MQ .

Esiste un'ulteriore limitazione alla limitazione di 32 KB sui messaggi con formato MQFMT_IMS_VAR_STRING. Quando i dati in un messaggio CCSID misto ASCII vengono convertiti in un messaggio CCSID misto EBCDIC, viene aggiunto un byte di fine stringa o un byte di inizio stringa ogni volta che si verifica una transizione tra i caratteri SBCS e DBCS. La limitazione di 32 KB si applica alla dimensione massima del messaggio. Ciò significa che, poiché il campo LL nel messaggio non può superare i 32 KB, il messaggio non deve superare i 32 KB inclusi tutti i caratteri di inizio e fine stringa. L'applicazione che crea il messaggio deve consentirlo.

z/OS *Conversione dei dati per i messaggi da e verso il bridge IMS*

La conversione dei dati viene eseguita dalla funzione di accodamento distribuito (che può richiamare le uscite necessarie) o dall'agente di accodamento all'interno del gruppo (che non supporta l'utilizzo delle uscite) quando inserisce un messaggio in una coda di destinazione che ha le informazioni XCF definite per la relativa classe di memoria. La conversione dei dati non si verifica quando un messaggio viene consegnato a una coda mediante pubblicazione / sottoscrizione.

Tutte le uscite necessarie devono essere disponibili per la funzione di accodamento distribuito nel dataset a cui fa riferimento l'istruzione CSQXLIB DD. Ciò significa che è possibile inviare messaggi a una applicazione IMS utilizzando il bridge IBM MQ - IMS da qualsiasi piattaforma IBM MQ .

Se si verificano errori di conversione, il messaggio viene inserito nella coda non convertita; ciò fa sì che venga considerato come un errore dal bridge IBM MQ - IMS , poiché il bridge non è in grado di riconoscere il formato dell'intestazione. Se si verifica un errore di conversione, viene inviato un messaggio di errore alla console z/OS .

Consultare “Scrittura delle uscite di conversione dati” a pagina 985 per informazioni dettagliate sulla conversione dei dati in generale.

Invio di messaggi al bridge IBM MQ - IMS

Per garantire che la conversione venga eseguita correttamente, è necessario indicare al gestore code il formato del messaggio.

Se il messaggio ha una struttura MQIIH, il *Format* in MQMD deve essere impostato sul formato integrato MQFMT_IMS e il *Format* in MQIIH deve essere impostato sul nome del formato che descrive i propri dati del messaggio. Se non è presente alcun MQIIH, impostare il *Format* in MQMD sul proprio nome formato.

Se i dati (diversi da LLZZs) sono tutti dati carattere (MQCHAR), utilizzare come nome formato (in MQIIH o MQMD, a seconda dei casi) il formato integrato MQFMT_IMS_VAR_STRING. Altrimenti, utilizzare il proprio nome formato, nel cui caso è necessario fornire anche un'uscita di conversione dati per il formato. L'uscita deve gestire la conversione degli LLZZs nel tuo messaggio, oltre ai dati stessi (ma non deve gestire alcun MQIIH all'inizio del messaggio).

Se l'applicazione utilizza *MFSMapName*, è possibile utilizzare i messaggi con MQFMT_IMS e definire il nome della mappa passato alla transazione IMS nel campo MFSMapName di MQIIH.

Ricezione di messaggi dal bridge IBM MQ - IMS

Se una struttura MQIIH è presente sul messaggio originale che si sta inviando a IMS, ne è presente anche uno sul messaggio di replica.

Per assicurarsi che la propria risposta sia convertita correttamente:

- Se si dispone di una struttura MQIIH sul messaggio originale, specificare il formato che si desidera per il messaggio di risposta nel campo MQIIH *ReplytoFormat* del messaggio originale. Questo valore viene inserito nel campo MQIIH *Format* del messaggio di risposta. Ciò è particolarmente utile se tutti i dati di output sono nel formato LLZZ < dati carattere>.
- Se non si dispone di una struttura MQIIH sul messaggio originale, specificare il formato che si desidera per il messaggio di risposta come nome MFS MOD nell'ISRT dell'applicazione IMS per IOPCB.

Sviluppo di applicazioni JMS/Jakarta Messaging e Java

IBM MQ fornisce tre interfacce di linguaggio Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSe IBM MQ classes for Java.

Informazioni su questa attività

V 9.3.0 JM 3.0 V 9.3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging è un provider Jakarta Messaging che implementa le interfacce Jakarta Messaging per IBM MQ come sistema di messaggistica. Jakarta Connectors Architecture fornisce una modalità standard di connessione delle applicazioni in esecuzione in un ambiente Jakarta EE a un EIS (Enterprise Information System) come IBM MQ o Db2.

Per ulteriori informazioni, consultare “Perché dovrei utilizzare IBM MQ classes for Jakarta Messaging?” a pagina 85 e “Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88.

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS è un provider JMS che implementa le interfacce JMS per IBM MQ come sistema di messaggistica. Java Platform, Enterprise Edition Connector Architecture (JCA) fornisce un modo standard per collegare le applicazioni in esecuzione in un ambiente Java EE a un EIS (Enterprise Information System) come IBM MQ o Db2.

Per ulteriori informazioni, consultare “Perché dovrei utilizzare IBM MQ classes for JMS?” a pagina 86 e “Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88.

IBM MQ classes for Java

IBM MQ classes for Java consente di utilizzare IBM MQ in un ambiente Java . IBM MQ classes for Java consenti a un'applicazione Java di connettersi a IBM MQ come client IBM MQ o di connettersi direttamente a un gestore code IBM MQ .

IBM MQ classes for Java incapsula MQI (Message Queue Interface), l'API IBM MQ nativa, e utilizza lo stesso modello di oggetto di altre interfacce orientate agli oggetti, mentre IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging implementano le interfacce di messaggistica Java da Oracle e Java Community Process rispettivamente.

Per ulteriori informazioni, consultare “Perché dovrei utilizzare IBM MQ classes for Java?” a pagina 351 e “Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88.

Nota:

Stabilized IBM non apporterà ulteriori miglioramenti a IBM MQ classes for Java e saranno funzionalmente stabilizzati al livello fornito in IBM MQ 8.0. Le applicazioni esistenti che utilizzano IBM MQ classes for Java continuano ad essere completamente supportate, ma le nuove funzioni non verranno aggiunte e le richieste di miglioramenti verranno rifiutate. Completamente supportato significa che i difetti verranno corretti insieme a tutte le modifiche richieste dalle modifiche ai requisiti di sistema IBM MQ .

Le IBM MQ classes for Java non sono supportate in IMS.

Le IBM MQ classes for Java non sono supportate in WebSphere Liberty. Non devono essere utilizzati con la funzione di messaggistica IBM MQ Liberty o con il supporto JCA generico. Per ulteriori informazioni, consultare Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

Utilizzo di IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, è supportato Jakarta Messaging 3.0 per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API JMS 2.0 che l'API Jakarta Messaging 3.0 nella stessa applicazione.

Nota: Per Jakarta Messaging 3.0, il controllo della specifica JMS passa da Oracle a Java Community Process. Tuttavia, Oracle conserva il controllo del nome "javax", che viene utilizzato in altre tecnologie Java che non sono state spostate nel processo della comunità Java . Quindi, mentre Jakarta Messaging 3.0 è funzionalmente equivalente a JMS 2.0 ci sono alcune differenze nella denominazione:

- Il nome ufficiale per la versione 3.0 è Jakarta Messaging invece di Java Message Service.
- I nomi di package e costanti hanno come prefisso `jakarta` anziché `javax`. Ad esempio, in JMS 2.0 la connessione iniziale a un fornitore di messaggistica è un oggetto `javax.jms.Connection` e in Jakarta Messaging 3.0 è un oggetto `jakarta.jms.Connection` .

JMS 2.0 I package javax.jms definiscono le interfacce JMS e un fornitore JMS implementa tali interfacce per un prodotto di messaggistica specifico. IBM MQ classes for JMS è un provider JMS che implementa le interfacce JMS per IBM MQ.

JM 3.0 I pacchetti jakarta.jms definiscono le interfacce Jakarta Messaging e un provider Jakarta Messaging implementa tali interfacce per un prodotto di messaggistica specifico. IBM MQ classes for Jakarta Messaging è un provider Jakarta Messaging che implementa le interfacce Jakarta Messaging per IBM MQ.

Le specifiche JMS e Jakarta Messaging prevedono che gli oggetti gestiti siano ConnectionFactory e Destination. Un amministratore crea e gestisce oggetti gestiti in un repository centrale e un'applicazione JMS o Jakarta Messaging richiama tali oggetti utilizzando Java Naming Directory Interface (JNDI).

JMS 2.0 Per JMS 2.0, un amministratore può utilizzare lo strumento di amministrazione IBM MQ JMS **JMSAdmin**o IBM MQ Explorer, per creare e gestire gli oggetti gestiti in un repository centrale.

JM 3.0 Per Jakarta Messaging 3.0, non è possibile gestire JNDI utilizzando IBM MQ Explorer. La gestione JNDI è supportata dalla variante Jakarta Messaging 3.0 di **JMSAdmin**, che è **JMS30Admin**.

Poiché JMS e Jakarta Messaging condividono molto in comune, ulteriori riferimenti a JMS in questo argomento possono essere considerati come riferimenti ad entrambi. Eventuali differenze vengono evidenziate come necessario.

IBM MQ classes for JMS fornisce anche due serie di estensioni all'API JMS . Il focus principale di queste estensioni riguarda la creazione e la configurazione di factory di connessione e destinazioni in modo dinamico al runtime, ma le estensioni forniscono anche funzioni che non sono direttamente correlate alla messaggistica, come la funzione per la determinazione dei problemi.

Le estensioni IBM MQ JMS

IBM MQ classes for JMS contiene le estensioni implementate in oggetti quali MQConnectionFactory, MQQueue e MQTopic. Questi oggetti hanno proprietà e metodi specifici di IBM MQ. Gli oggetti possono essere gestiti oppure un'applicazione può creare gli oggetti in modo dinamico al runtime. Queste estensioni sono definite estensioni IBM MQ JMS .

Le estensioni IBM JMS

IBM MQ classes for JMS fornisce inoltre una serie più generica di estensioni all'API JMS , che non è specifica di IBM MQ come sistema di messaggistica o di Java come linguaggio di programmazione utilizzato. Queste estensioni sono definite estensioni IBM JMS e hanno i seguenti obiettivi generali:

- Per fornire un livello maggiore di coerenza tra i provider IBM JMS .
- Per semplificare la scrittura di un'applicazione bridge tra due sistemi di messaggistica IBM .
- Per rendere più semplice la porta di un'applicazione da un provider IBM JMS a un altro.

Le estensioni forniscono una funzione simile a quella fornita in IBM MQ Message Service Client (XMS) for C/C++ e IBM MQ Message Service Client (XMS) for .NET.

Concetti correlati

Interfacce di lingua IBM MQ Java

Attività correlate

“Scrittura di applicazioni IBM MQ classes for JMS/Jakarta Messaging” a pagina 142

Dopo una breve introduzione al modello JMS , questa sezione fornisce una guida dettagliata su come scrivere applicazioni IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging .

V 9.3.0 JM 3.0 V 9.3.0 Perché dovrei utilizzare IBM MQ classes for Jakarta Messaging?

L'utilizzo di IBM MQ classes for Jakarta Messaging ha una serie di vantaggi, tra cui la possibilità di riutilizzare tutte le competenze Jakarta Messaging esistenti nella tua organizzazione e le applicazioni che sono più indipendenti dal provider Jakarta Messaging e dalla configurazione IBM MQ sottostante.

Riepilogo dei vantaggi dell'utilizzo di IBM MQ classes for Jakarta Messaging

L'uso di IBM MQ classes for Jakarta Messaging consente di riutilizzare le competenze Jakarta Messaging esistenti e di fornire l'indipendenza dell'applicazione.

- Puoi riutilizzare le competenze Jakarta Messaging .

  IBM MQ classes for Jakarta Messaging è un provider Jakarta Messaging che implementa le interfacce Jakarta Messaging per IBM MQ come sistema di messaggistica. Se la tua organizzazione è nuova a IBM MQ, ma ha già le competenze di sviluppo dell'applicazione Jakarta Messaging (o JMS), potresti trovare più semplice utilizzare l'API Jakarta Messaging familiare per accedere alle risorse IBM MQ piuttosto che una delle altre API fornite con IBM MQ.
- Jakarta Messaging è parte integrante di Jakarta EE.

  Jakarta Messaging è l'API naturale da utilizzare per la messaggistica sulla piattaforma Jakarta EE . Ogni server delle applicazioni compatibile con Jakarta EE deve contenere un fornitore Jakarta Messaging . È possibile utilizzare Jakarta Messaging in client delle applicazioni, servlet, JSP (Java Server Pages), EJB (enterprise Java beans) e MDB (message driven beans). Si noti in particolare che le applicazioni Jakarta EE utilizzano MDB per elaborare i messaggi in modo asincrono e tutti i messaggi vengono consegnati agli MDB come messaggi Jakarta Messaging .
- I factory di connessione e le destinazioni possono essere memorizzati come oggetti gestiti Jakarta Messaging in un repository centrale piuttosto che essere codificati in un'applicazione.

  Un amministratore può creare e gestire gli oggetti gestiti Jakarta Messaging in un repository centrale e le applicazioni IBM MQ classes for Jakarta Messaging possono recuperare tali oggetti utilizzando Java Naming Directory Interface (JNDI). Le Jakarta Messaging factory di connessione e le destinazioni incapsulano IBM MQle informazioni specifiche come i nomi dei gestori code, i nomi dei canali, le opzioni di connessione, i nomi delle code e i nomi degli argomenti. Se le factory di connessione e le destinazioni sono memorizzate come oggetti gestiti, queste informazioni non sono codificate in modo permanente in un'applicazione. Questa disposizione fornisce quindi all'applicazione un grado di indipendenza dalla configurazione IBM MQ sottostante.
- Jakarta Messaging è un'API standard del settore che può fornire la portabilità dell'applicazione.

Un'applicazione Jakarta Messaging può utilizzare JNDI per richiamare le factory di connessione e le destinazioni memorizzate come oggetti gestiti e utilizzare solo le interfacce definite nel package `jakarta.jms` (Jakarta Messaging 3.0) per eseguire operazioni di messaggistica. L'applicazione è quindi completamente indipendente da qualsiasi provider Jakarta Messaging , ad esempio IBM MQ classes for Jakarta Messaging, e può essere trasferita da un provider Jakarta Messaging a un altro senza alcuna modifica all'applicazione.

Se JNDI non è disponibile in un determinato ambiente dell'applicazione, un'applicazione IBM MQ classes for Jakarta Messaging può utilizzare le estensioni all'API Jakarta Messaging per creare e configurare le factory di connessione e le destinazioni in modo dinamico in fase di runtime. L'applicazione è quindi completamente autonoma, ma è collegata a IBM MQ classes for Jakarta Messaging come provider Jakarta Messaging .

- Le applicazioni bridge potrebbero essere più semplici da scrivere utilizzando Jakarta Messaging.

  Un'applicazione bridge è un'applicazione che riceve i messaggi da un sistema di messaggistica e li invia a un altro sistema di messaggistica. La scrittura di un'applicazione bridge può essere complicata utilizzando le API specifiche del prodotto e i formati del messaggio. Invece, è possibile scrivere un'applicazione bridge utilizzando due provider Jakarta Messaging , uno per ogni sistema di messaggistica. L'applicazione utilizza quindi solo un'API, l'API Jakarta Messaging ed elabora solo i messaggi Jakarta Messaging .

Ambienti distribuibili

Per fornire l'integrazione con un server delle applicazioni Jakarta EE , gli standard Jakarta EE richiedono che i fornitori di messaggistica forniscano un adattatore risorse. Seguendo la specifica Jakarta Connectors Architecture , IBM MQ fornisce un adattatore di risorse che utilizza Jakarta Messaging per fornire funzioni di messaggistica in qualsiasi ambiente Jakarta EE certificato. Per ulteriori informazioni, consultare “Liberty e l'adattatore risorse IBM MQ” a pagina 443.

Nota: WebSphere Application Server traditional attualmente non supporta Jakarta EE.

Al di fuori dell'ambiente Jakarta EE , vengono forniti i file OSGi e JAR, semplificando l'ottenimento di IBM MQ classes for Jakarta Messaging. Questi file JAR sono più facilmente distribuibili in modo autonomo o all'interno di framework di gestione software come Maven. Per ulteriori informazioni, fare riferimento a “Ottenimento di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging separatamente” a pagina 129.

Concetti correlati

Classi IBM MQ per Jakarta Messaging: una panoramica

“Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88

IBM MQ fornisce tre interfacce di linguaggio Java .

JMS 2.0 Perché dovrei utilizzare IBM MQ classes for JMS?

L'utilizzo di IBM MQ classes for JMS ha una serie di vantaggi, tra cui la possibilità di riutilizzare tutte le competenze JMS esistenti nella tua organizzazione e le applicazioni che sono più indipendenti dal provider JMS e dalla configurazione IBM MQ sottostante.

Riepilogo dei vantaggi dell'utilizzo di IBM MQ classes for JMS

L'uso di IBM MQ classes for JMS consente di riutilizzare le competenze JMS esistenti e di fornire l'indipendenza dell'applicazione.

V 9.3.0 V 9.3.0

Nota: JMS 2.0 è stato sostituito da Jakarta Messaging. IBM MQ classes for JMS continua a supportare lo standard JMS 2.0 , ma i futuri miglioramenti alla messaggistica Java emergeranno solo in Jakarta Messaging, quindi in IBM MQ classes for Jakarta Messaging. IBM MQ classes for JMS sono consigliati solo per la manutenzione e l'estensione di applicazioni JMS 2.0 esistenti. IBM MQ classes for Jakarta Messaging dovrebbe essere la tecnologia preferita per il nuovo sviluppo.

- Puoi riutilizzare le competenze JMS .

  IBM MQ classes for JMS è un provider JMS che implementa le interfacce JMS per IBM MQ come sistema di messaggistica. Se la tua organizzazione non ha dimestichezza con IBM MQ, ma ha già competenze di sviluppo dell'applicazione JMS , potresti trovare più semplice utilizzare l'API familiare JMS per accedere alle risorse IBM MQ piuttosto che una delle altre API fornite con IBM MQ.
- JMS è una parte integrale di Java Platform, Enterprise Edition (Java EE).

  JMS è l'API naturale da utilizzare per la messaggistica sulla piattaforma Java EE . Ogni server delle applicazioni compatibile con Java EE deve contenere un fornitore JMS . È possibile utilizzare JMS in client delle applicazioni, servlet, JSP (Java Server Pages), EJB (enterprise Java beans) e MDB (message driven beans). Si noti in particolare che le applicazioni Java EE utilizzano MDB per elaborare i messaggi in modo asincrono e tutti i messaggi vengono consegnati agli MDB come messaggi JMS .
- I factory di connessione e le destinazioni possono essere memorizzati come oggetti gestiti JMS in un repository centrale piuttosto che essere codificati in un'applicazione.

  Un amministratore può creare e gestire gli oggetti gestiti JMS in un repository centrale e le applicazioni IBM MQ classes for JMS possono recuperare tali oggetti utilizzando Java Naming Directory Interface (JNDI). Le JMS factory di connessione e le destinazioni incapsulano IBM MQle informazioni specifiche come i nomi dei gestori code, i nomi dei canali, le opzioni di connessione, i nomi delle code e i nomi degli argomenti. Se le factory di connessione e le destinazioni sono memorizzate come oggetti gestiti, queste informazioni non sono codificate in modo permanente in un'applicazione. Questa disposizione fornisce quindi all'applicazione un grado di indipendenza dalla configurazione IBM MQ sottostante.
- JMS è un'API standard del settore che può fornire la portabilità dell'applicazione.

  Un'applicazione JMS può utilizzare JNDI per richiamare le produzioni connessioni e le destinazioni memorizzate come oggetti gestiti e utilizzare solo le interfacce definite nel pacchetto `javax.jms` per eseguire operazioni di messaggistica. L'applicazione è quindi completamente indipendente da qualsiasi provider JMS , ad esempio IBM MQ classes for JMS, e può essere trasferita da un provider JMS a un altro senza alcuna modifica all'applicazione.

  Se JNDI non è disponibile in un particolare ambiente di applicazione, un'applicazione IBM MQ classes for JMS può utilizzare le estensioni all'API JMS per creare e configurare le factory di connessione e le destinazioni in modo dinamico al runtime. L'applicazione è quindi completamente autonoma, ma è collegata a IBM MQ classes for JMS come provider JMS .
- Le applicazioni bridge potrebbero essere più semplici da scrivere utilizzando JMS.

  Un'applicazione bridge è un'applicazione che riceve i messaggi da un sistema di messaggistica e li invia a un altro sistema di messaggistica. La scrittura di un'applicazione bridge può essere complicata utilizzando le API specifiche del prodotto e i formati del messaggio. Invece, è possibile scrivere un'applicazione bridge utilizzando due provider JMS , uno per ogni sistema di messaggistica. L'applicazione utilizza quindi solo un'API, l'API JMS ed elabora solo i messaggi JMS .

Ambienti distribuibili

Per fornire l'integrazione con un server delle applicazioni Java EE , gli standard Java EE richiedono che i fornitori di messaggistica forniscano un adattatore risorse. Seguendo la specifica Java EE Connector Architecture (JCA), IBM MQ fornisce un adattatore di risorse che utilizza JMS per fornire funzioni di messaggistica in qualsiasi ambiente Java EE certificato.

Anche se è stato possibile utilizzare IBM MQ classes for Java all'interno di Java EE, questa API non è stata progettata o ottimizzata per questo scopo. Per ulteriori informazioni sulle considerazioni IBM MQ classes for Java all'interno di Java EE, consultare “Esecuzione di applicazioni IBM MQ classes for Java in Java EE” a pagina 352.

Al di fuori dell'ambiente Java EE , vengono forniti i file OSGi e JAR, semplificando l'ottenimento di IBM MQ classes for JMS. Questi file JAR sono più facilmente distribuibili in modo autonomo o all'interno di framework di gestione software come Maven. Per ulteriori informazioni, fare riferimento a “Ottenimento di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging separatamente” a pagina 129.

Concetti correlati

V 9.3.0 V 9.3.0 Classi IBM MQ per Jakarta Messaging: una panoramica

“Perché dovrei utilizzare IBM MQ classes for Jakarta Messaging?” a pagina 85
L'utilizzo di IBM MQ classes for Jakarta Messaging ha una serie di vantaggi, tra cui la possibilità di riutilizzare tutte le competenze Jakarta Messaging esistenti nella tua organizzazione e le applicazioni che sono più indipendenti dal provider Jakarta Messaging e dalla configurazione IBM MQ sottostante.

“Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88
IBM MQ fornisce tre interfacce di linguaggio Java .

Accesso a IBM MQ da Java - Scelta dell'API

IBM MQ fornisce tre interfacce di linguaggio Java .

- V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging consente alle applicazioni scritte utilizzando le Jakarta Messaging 3.0 API di utilizzare IBM MQ come provider di messaggistica.

Jakarta Messaging è la direzione strategica per la messaggistica nelle applicazioni Java .

Jakarta Messaging 3.0 è funzionalmente equivalente a JMS 2.0, quindi per ulteriori informazioni consultare “Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84.

IBM MQ classes for JMS

IBM MQ classes for JMS consente alle applicazioni scritte utilizzando le JMS 2.0 API di utilizzare IBM MQ come provider di messaggistica.

Poiché Jakarta Messaging sostituisce JMS, IBM MQ classes for JMS è consigliato per l'utilizzo in applicazioni esistenti o in ambienti (ad esempio, WebSphere Application Server) che non supportano Jakarta Messaging.

Non è possibile utilizzare sia IBM MQ classes for Jakarta Messaging che IBM MQ classes for JMS nella stessa applicazione.

Per ulteriori informazioni, fare riferimento a “Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84.

IBM MQ classes for Java

Stabilized L'altra API che le applicazioni Java possono utilizzare per accedere alle risorse IBM MQ è IBM MQ classes for Java, che fornisce un'API orientata a IBM MQper i programmi da utilizzare IBM MQ come provider di messaggistica. Tuttavia, IBM MQ classes for Java è funzionalmente stabilizzato al livello fornito in IBM MQ 8.0. Per ulteriori informazioni, consultare “Perché dovrei utilizzare IBM MQ classes for Java?” a pagina 351. Anche se le applicazioni esistenti che utilizzano IBM MQ classes for Java continuano ad essere completamente supportate, le nuove applicazioni dovrebbero utilizzare IBM MQ classes for Jakarta Messaging.

Funzioni comuni di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging forniscono l'accesso sia alle funzioni di messaggistica point-to-point che di pubblicazione / sottoscrizione di IBM MQ. Oltre a inviare messaggi JMS che forniscono supporto per il modello di messaggistica standard JMS , le applicazioni possono anche inviare e ricevere messaggi senza ulteriori intestazioni e quindi possono interagire con altre

applicazioni IBM MQ , ad esempio, le applicazioni C MQI. È disponibile il pieno controllo dei payload dei messaggi di MQMD e MQ .

Sono disponibili anche ulteriori funzioni IBM MQ come il flusso di messaggi, i messaggi di inserimento asincroni e i messaggi di report.

Utilizzando le classi helper PCF fornite, i messaggi di gestione PCF IBM MQ possono essere inviati e ricevuti tramite JMS API e possono essere utilizzati per gestire i gestori code.

Le funzioni che sono state recentemente aggiunte a IBM MQ, come il consumo asincrono e la riconnessione automatica, non sono disponibili in IBM MQ classes for Java, ma in IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Richiesta di miglioramenti

Se è necessario accedere a funzioni IBM MQ che non sono disponibili tramite IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, è possibile generare un'idea.

IBM può quindi indicare se l'implementazione è possibile nell'implementazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging oppure se è possibile seguire una procedura ottimale.

Per le funzioni di messaggistica aggiuntive, poiché IBM è un contributore allo standard aperto, queste funzioni possono essere sollevate come parte del processo JCP. Questo vale solo per Jakarta Messaging.

Informazioni correlate

Benvenuti in IBM Ideas Portal

Processo di revisione specifica JMS Java

Utilizzo di JMS per inviare messaggi PCF

V 9.3.0 JM 3.0 V 9.3.0 Prerequisiti per IBM MQ classes for Jakarta Messaging

Questo argomento indica le informazioni necessarie prima di utilizzare IBM MQ classes for Jakarta Messaging. Per sviluppare ed eseguire applicazioni IBM MQ classes for Jakarta Messaging , è necessario disporre di determinati componenti software come prerequisiti.

Per informazioni sui prerequisiti per IBM MQ classes for Jakarta Messaging, consultare Requisiti di sistema per IBM MQ.

Per sviluppare applicazioni IBM MQ classes for Jakarta Messaging , è necessario un SDK (Software Development Kit) Java SE . per i dettagli dei JDK supportati dal proprio sistema operativo, consultare Requisiti di sistema per IBM MQ.

Per eseguire applicazioni IBM MQ classes for Jakarta Messaging , sono necessari i seguenti componenti software:

- Un gestore code IBM MQ .
- Un Java runtime environment (JRE), per ogni sistema su cui si eseguono le applicazioni.
- IBM i Per IBM i, Qshell, che è l'opzione 30 del sistema operativo.
- z/OS Per z/OS, z/OS UNIX System Services (z/OS UNIX).

Il provider JSSE IBM include un provider di crittografia certificato FIPS, quindi può essere configurato in modo programmatico per la conformità FIPS 140-2 pronta per l'uso immediato. Pertanto, la conformità FIPS 140-2 può essere supportata direttamente da IBM MQ classes for Jakarta Messaging.

Il provider JSSE di Oraclepuò avere un provider di crittografia certificato FIPS configurato in esso, ma non è pronto per l'uso immediato e non è disponibile per la configurazione programmatica. Pertanto, in questo caso IBM MQ classes for Jakarta Messaging non può abilitare direttamente la compatibilità FIPS 140-2. Potresti essere in grado di abilitare manualmente tale conformità, ma IBM non può attualmente fornire una guida su questo.

È possibile utilizzare gli indirizzi Internet Protocol versione 6 (IPv6) nelle proprie applicazioni IBM MQ classes for Jakarta Messaging se gli indirizzi IPv6 sono supportati dalla JVM (Java virtual machine) e dall'implementazione TCP/IP sul proprio sistema operativo. Lo strumento di amministrazione IBM MQ Jakarta Messaging , **JMS30Admin**, accetta anche indirizzi IPv6 . Per ulteriori informazioni su questo strumento, consultare Configurazione degli oggetti JMS e Jakarta Messaging utilizzando gli strumenti di gestione.

Lo strumento di amministrazione IBM MQ JMS e IBM MQ Explorer utilizzano Java Naming Directory Interface (JNDI) per accedere a un servizio directory, che memorizza gli oggetti gestiti. Le applicazioni IBM MQ classes for Jakarta Messaging possono inoltre utilizzare JNDI per richiamare gli oggetti gestiti da un servizio directory.

Nota: Per Jakarta Messaging 3.0, non è possibile gestire JNDI utilizzando IBM MQ Explorer. La gestione JNDI è supportata dalla variante Jakarta Messaging 3.0 di **JMSAdmin**, che è **JMS30Admin**.

Un provider di servizio è un codice che fornisce l'accesso a un servizio di directory associando le chiamate JNDI al servizio di directory. Un provider del servizio file system nei file `fscontext.jar` e `providerutil.jar` viene fornito con IBM MQ classes for Jakarta Messaging. Il provider di servizi del file system fornisce l'accesso a un servizio di directory basato sul file system locale.

Se si intende utilizzare un servizio di directory basato su un server LDAP, è necessario installare e configurare un server LDAP o avere accesso a un server LDAP esistente. In particolare, è necessario configurare il server LDAP per memorizzare oggetti Java . Per informazioni su come installare e configurare il server LDAP, consultare la documentazione fornita con il server.

JMS 2.0 Prerequisiti per IBM MQ classes for JMS

Questo argomento indica le informazioni necessarie prima di utilizzare IBM MQ classes for JMS. Per sviluppare ed eseguire applicazioni IBM MQ classes for JMS , è necessario disporre di determinati componenti software come prerequisiti.

Per informazioni sui prerequisiti per IBM MQ classes for JMS, consultare Requisiti di sistema per IBM MQ.

Per sviluppare applicazioni IBM MQ classes for JMS , è necessario un SDK (Software Development Kit) Java SE . per i dettagli dei JDK supportati dal proprio sistema operativo, consultare Requisiti di sistema per IBM MQ.

Per eseguire applicazioni IBM MQ classes for JMS , sono necessari i seguenti componenti software:

- Un gestore code IBM MQ .
- Un Java runtime environment (JRE), per ogni sistema su cui si eseguono le applicazioni.
- IBM i Per IBM i, Qshell, che è l'opzione 30 del sistema operativo.
- z/OS Per z/OS, z/OS UNIX System Services (z/OS UNIX).

Il provider JSSE IBM include un provider di crittografia certificato FIPS, quindi può essere configurato in modo programmatico per la conformità FIPS 140-2 pronta per l'uso immediato. Pertanto, la conformità FIPS 140-2 può essere supportata direttamente da IBM MQ classes for Java e IBM MQ classes for JMS.

Il provider JSSE di Oraclepuò avere un provider di crittografia certificato FIPS configurato in esso, ma non è pronto per l'uso immediato e non è disponibile per la configurazione programmatica. Pertanto, in questo caso IBM MQ classes for Java e IBM MQ classes for JMS non possono abilitare direttamente la compatibilità FIPS 140-2. Potresti essere in grado di abilitare manualmente tale conformità, ma IBM non può attualmente fornire una guida su questo.

È possibile utilizzare gli indirizzi Internet Protocol versione 6 (IPv6) nelle proprie applicazioni IBM MQ classes for JMS se gli indirizzi IPv6 sono supportati dalla JVM (Java virtual machine) e dall'implementazione TCP/IP sul proprio sistema operativo. Lo strumento di amministrazione IBM MQ JMS (consultare Configurazione degli oggetti JMS utilizzando lo strumento di gestione) accetta anche indirizzi IPv6 .

Lo strumento di amministrazione IBM MQ JMS e IBM MQ Explorer utilizzano Java Naming Directory Interface (JNDI) per accedere a un servizio directory, che memorizza gli oggetti gestiti. Le applicazioni

IBM MQ classes for JMS possono inoltre utilizzare JNDI per richiamare gli oggetti gestiti da un servizio directory. Un provider di servizio è un codice che fornisce l'accesso a un servizio di directory associando le chiamate JNDI al servizio di directory. Un provider del servizio file system nei file `fscontext.jar` e `providerutil.jar` viene fornito con IBM MQ classes for JMS. Il provider di servizi del file system fornisce l'accesso a un servizio di directory basato sul file system locale.

Se si intende utilizzare un servizio di directory basato su un server LDAP, è necessario installare e configurare un server LDAP o avere accesso a un server LDAP esistente. In particolare, è necessario configurare il server LDAP per memorizzare oggetti Java . Per informazioni su come installare e configurare il server LDAP, consultare la documentazione fornita con il server.

Installazione e configurazione di IBM MQ classes for JMS/Jakarta Messaging

Questa sezione descrive le directory e i file che vengono creati quando si installano IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaginge indica come configurare IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging dopo l'installazione.

Concetti correlati

“Utilizzo dell'adattatore di risorse IBM MQ” a pagina 437
L'adattatore di risorse consente alle applicazioni in esecuzione in un application server di accedere alle risorse IBM MQ . Supporta la comunicazione in entrata e in uscita.

Cosa è installato per IBM MQ classes for JMS

Una serie di file e directory vengono creati quando si installa IBM MQ classes for JMS. Su Windows, alcune configurazioni vengono eseguite durante l'installazione impostando automaticamente le variabili di ambiente. Su altre piattaforme e in determinati ambienti Windows , è necessario impostare le variabili di ambiente prima di eseguire applicazioni IBM MQ classes for JMS .

Per la maggior parte dei sistemi operativi, IBM MQ classes for JMS viene installato come componente facoltativo quando si installa IBM MQ.

Per ulteriori informazioni sull'installazione di IBM MQ, consultare:

- Multi Installazione di IBM MQ
- z/OS Installazione di IBM MQ for z/OS

Importante: A parte i file JAR riposizionabili descritti in “IBM MQ classes for JMS/Jakarta Messaging file JAR rilocabili” a pagina 93, la copia dei file JAR IBM MQ classes for JMS o delle librerie native su altre macchine o in un'altra ubicazione su una macchina su cui è stato installato IBM MQ classes for JMS non è supportata.

Directory di installazione

Tabella 5 a pagina 91 mostra dove sono installati i file IBM MQ classes for JMS su ciascuna piattaforma.

Tabella 5. Directory di installazione IBM MQ classes for JMS

Piattaforma	Directory
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*`/java`
Windows Windows	*MQ_INSTALLATION_PATH*`\java`
IBM i IBM i	`/QIBM/ProdData/mqm/java`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java`

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

La directory di installazione include il seguente contenuto:

- I file JAR IBM MQ classes for JMS , inclusi i file JAR rilocabili, che si trovano nella directory *MQ_INSTALLATION_PATH*\java\lib .
- Le librerie native IBM MQ , utilizzate dalle applicazioni che utilizzano Java Native Interface.

  Le librerie native a 32 bit sono installate nella directory *MQ_INSTALLATION_PATH*\java\lib e le librerie native a 64 bit sono disponibili nella directory *MQ_INSTALLATION_PATH*\java\lib64 .

  Per ulteriori informazioni sulle librerie native IBM MQ , consultare "Configurazione delle librerie JNI (Java Native Interface)" a pagina 98.
- Ulteriori script descritti in "Script forniti con IBM MQ classes for JMS/Jakarta Messaging" a pagina 125. Questi script si trovano nella directory *MQ_INSTALLATION_PATH*\java\bin .
- Le specifiche dell'API IBM MQ classes for JMS . Lo strumento Javadoc è stato utilizzato per generare le pagine HTML che contengono le specifiche dell'API.

  Le pagine HTML si trovano nella directory *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses :

  – ALW In AIX, Linux, and Windows, questa sottodirectory contiene le singole pagine HTML.

  – IBM i Su IBM i, le pagine HTML si trovano in un file denominato wmqjms_javadoc.jar.

  – z/OS Su z/OS, le pagine HTML si trovano in un file denominato wmqjms_javadoc.jar.
- Supporto per OGSi. I bundle OSGi sono installati nella directory java\lib\OSGi e descritti in "Supporto per OSGi con IBM MQ classes for JMS" a pagina 126.
- L'adattatore di risorse IBM MQ , che può essere distribuito in qualsiasi server delle applicazioni conforme a Java Platform, Enterprise Edition 7 (Java EE 7) o Jakarta EE .

  L'adattatore di risorse IBM MQ si trova nella directory *MQ_INSTALLATION_PATH*\java\lib\jca ; per ulteriori informazioni, consultare "Utilizzo dell'adattatore di risorse IBM MQ" a pagina 437
- Windows Su Windows, i simboli che possono essere utilizzati per il debug sono installati nella directory *MQ_INSTALLATION_PATH*\java\lib\symbols .

La directory di installazione include anche alcuni file che appartengono ad altri componenti IBM MQ .

Applicazioni di esempio

JMS 2.0 Alcune applicazioni di esempio vengono fornite con IBM MQ classes for JMS. Tabella 6 a pagina 92 mostra dove sono installate le applicazioni di esempio su ciascuna piattaforma.

JM 3.0 Per IBM MQ classes for Jakarta Messaging, si stanno preparando nuovi esempi.

JMS 2.0

Tabella 6. Directory di esempi per IBM MQ classes for JMS

Piattaforma	**Directory**
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/samp/jms
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms
IBM i IBM i	/QIBM/ProdData/mqm/java/samples/jms
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

In questa tabella *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ .

Dopo l'installazione, potrebbe essere necessario eseguire alcune attività di configurazione per compilare ed eseguire le applicazioni.

"Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging" a pagina 95 descrive il percorso classi richiesto per eseguire le applicazioni IBM MQ classes for JMS di esempio. Questo argomento descrive inoltre i file JAR aggiuntivi a cui è necessario fare riferimento in circostanze speciali e le variabili di ambiente che è necessario impostare per eseguire gli script forniti con IBM MQ classes for JMS.

Per controllare le proprietà, come la traccia e la registrazione di un'applicazione, è necessario fornire un file delle proprietà di configurazione. Il file delle proprietà di configurazione IBM MQ classes for JMS è descritto in "Il file di configurazione IBM MQ classes for JMS/Jakarta Messaging" a pagina 100.

Concetti correlati

Problemi nella distribuzione dell'adattatore di risorse

Attività correlate

"Utilizzo di applicazioni di esempio IBM MQ classes for JMS" a pagina 122
Le applicazioni di esempio IBM MQ classes for JMS forniscono una panoramica delle funzioni comuni dell'API JMS . È possibile utilizzarle per verificare l'installazione e la configurazione del server di messaggistica e per creare le proprie applicazioni.

IBM MQ classes for JMS/Jakarta Messaging file JAR rilocabili

I file JAR riposizionabili possono essere spostati su sistemi che devono eseguire IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Importante:

- A parte i file JAR riposizionabili descritti in File JAR riposizionabili, la copia dei file JAR IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging o delle librerie native su altre macchine o in un'ubicazione diversa su una macchina in cui sono stati installati IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging non è supportata.
- Non includere i file JAR riposizionabili all'interno delle applicazioni distribuite nei server delle applicazioni Java EE , come WebSphere Application Server o WebSphere Liberty. In questi ambienti, l'adattatore di risorse IBM MQ deve essere distribuito e utilizzato. Tenere presente che WebSphere Application Server integra l'adattatore di risorse IBM MQ , quindi non è necessario distribuirlo manualmente in questo ambiente.
- Per evitare conflitti del classloader, non è consigliabile raggruppare i file JAR riposizionabili all'interno di più applicazioni all'interno dello stesso runtime Java . In questo scenario, rendere disponibili i file JAR IBM MQ rilocabili nel percorso classi del runtime Java .
- Se si stanno raggruppando i file JAR riposizionabili all'interno delle applicazioni, assicurarsi di includere tutti i file JAR prerequisiti come descritto in File JAR riposizionabili. Inoltre, è necessario accertarsi di disporre delle procedure appropriate per aggiornare i file JAR integrati come parte della manutenzione dell'applicazione, per garantire che IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging siano ancora correnti e che i problemi noti vengano rimediati.

File JAR rilocabili

All'interno di un'azienda, i seguenti file possono essere spostati su sistemi che devono eseguire IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging:

- V 9.3.0 V 9.3.0 `bcpkix-jdk15to18.jar` "4" a pagina 94
- V 9.3.5 `bcpkix-jdk18on.jar` "3" a pagina 94
- V 9.3.0 V 9.3.0 `bcprov-jdk15to18.jar` "4" a pagina 94
- V 9.3.5 `bcprov-jdk18on.jar` "3" a pagina 94
- V 9.3.0 V 9.3.0 `bcutil-jdk15to18.jar` "4" a pagina 94

- V 9.3.5 bcutil-jdk18on.jar “3” a pagina 94
- JMS 2.0 com.ibm.mq.allclient.jar “1” a pagina 94
- V 9.3.0 JM 3.0 V 9.3.0 com.ibm.mq.jakarta.client.jar “2” a pagina 94
- V 9.3.3 Removed com.ibm.mq.traceControl.jar
- fscontext.jar
- V 9.3.3 jackson-annotations.jar
- V 9.3.3 jackson-core.jar
- V 9.3.3 jackson-databind.jar
- jakarta.jms-api.jar
- jms.jar
- org.json.jar
- providerutil.jar

Note:

1. JMS 2.0 e JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery da IBM MQ 9.3.5
4. Long Term Support e Continuous Delivery prima IBM MQ 9.3.5

JMS File JAR

V 9.3.0 V 9.3.0 jms.jar contiene le interfacce JMS 1.1 e JMS 2.0 , denominate javax.jms.*.

V 9.3.0 JM 3.0 V 9.3.0 jakarta.jms-api.jar contiene le interfacce Jakarta Messaging 3.0 , denominate jakarta.jms.*.

fscontext.jar e providerutil.jar

I file fscontext.jar e providerutil.jar sono richiesti se l'applicazione esegue ricerche JNDI utilizzando un contesto del file system.

Provider di sicurezza Bouncy Castle e file JAR di supporto CMS

Sono necessari il provider di sicurezza Bouncy Castle e i file JAR di supporto CMS . Per ulteriori informazioni, vedi Support for non -IBM JREs with AMS.

V 9.3.5 Per Continuous Delivery da IBM MQ 9.3.5, sono obbligatori i seguenti file JAR:

- bcpkix-jdk18on.jar
- bcprov-jdk18on.jar
- bcutil-jdk18on.jar

LTS Per Long Term Support e Continuous Delivery prima di IBM MQ 9.3.5, sono obbligatori i seguenti file JAR:

- bcpkix-jdk15to18.jar
- bcprov-jdk15to18.jar
- bcutil-jdk15to18.jar

org.json.jar

Il file `org.json.jar` è richiesto se la propria applicazione IBM MQ classes for JMS utilizza una CCDT in formato JSON.

com.ibm.mq.allclient.jar e com.ibm.mq.jakarta.client.jar

I file `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar` contengono le classi IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging, IBM MQ classes for Javae PCF e Headers. Se si spostano questi file JAR in una nuova ubicazione, assicurarsi di eseguire le operazioni necessarie per mantenere questa nuova ubicazione con i nuovi Fix Pack IBM MQ . Inoltre, assicurarsi che l'utilizzo dei file sia reso noto al supporto IBM se si sta ottenendo una fix temporanea.

Per determinare la versione dei file `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar`, utilizzare il comando seguente:

V 9.3.0 JM 3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

Il seguente esempio mostra alcuni output di esempio di questo comando:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar, jackson-core.jar e jackson-databind.jar

V 9.3.3

I tre file JAR Jackson sono richiesti se l'applicazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging crea connessioni TLS sicure a un gestore code.

Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging
Prima di poter compilare ed eseguire le applicazioni IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , l'impostazione per la variabile di ambiente **CLASSPATH** deve includere il file JAR IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging Java . A seconda dei requisiti, potrebbe essere necessario aggiungere altri file JAR al percorso di classe. Per eseguire gli script forniti con IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, è necessario impostare altre variabili di ambiente.

Prima di iniziare

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, consultare Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging.

Importante: L'impostazione dell'opzione Java *-Xbootclasspath* per includere IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging non è supportata.

Informazioni su questa attività

Per compilare ed eseguire applicazioni IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , utilizzare l'impostazione **CLASSPATH** per la propria piattaforma e la versione di messaggistica Java , come mostrato nelle seguenti tabelle. In alternativa, è possibile specificare il percorso classe nel comando **java** invece di utilizzare la variabile di ambiente.

JMS 2.0 Per IBM MQ classes for JMS, l'impostazione include la directory degli esempi, in modo che sia possibile compilare ed eseguire applicazioni di esempio IBM MQ classes for JMS .

JM 3.0 Per IBM MQ classes for Jakarta Messaging, si stanno preparando nuovi esempi.

JM 3.0

*Tabella 7. Impostazioni **CLASSPATH** per Jakarta Messaging 3.0 per compilare e eseguire applicazioni IBM MQ classes for Jakarta Messaging*

Piattaforma	CLASSPATH impostazione
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

*Tabella 8. Impostazioni **CLASSPATH** per JMS 2.0 per compilare ed eseguire applicazioni IBM MQ classes for JMS , incluse le applicazioni di esempio*

Piattaforma	impostazione CLASSPATH
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mq / java/samples/jms/samples:

*Tabella 8. Impostazioni **CLASSPATH** per JMS 2.0 per compilare ed eseguire applicazioni IBM MQ classes for JMS , incluse le applicazioni di esempio (Continua)*	
Piattaforma	**impostazione CLASSPATH**
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\sample;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

In queste tabelle, *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ .

Il manifest del file JAR `com.ibm.mq.jakarta.client.jar` o `com.ibm.mq.allclient.jar` contiene riferimenti alla maggior parte degli altri file JAR richiesti dalle applicazioni IBM MQ classes for JMS , quindi non è necessario aggiungere tali file JAR al percorso di classe. Questi file JAR includono quelli richiesti dalle applicazioni che utilizzano JNDI (Java Naming Directory Interface) per richiamare gli oggetti gestiti da un servizio di directory e dalle applicazioni che utilizzano JTA (Java Transaction API).

Tuttavia, è necessario includere ulteriori file JAR nel percorso di classe nelle seguenti circostanze:

- Se si utilizzano classi di uscita canale che implementano le interfacce di uscita canale definite nel pacchetto `com.ibm.mq` , invece di quelle definite nel pacchetto `com.ibm.mq.exits` , è necessario aggiungere il file JAR IBM MQ classes for Java , `com.ibm.mq.jar`, al percorso di classe.
- Se l'applicazione utilizza JNDI per richiamare gli oggetti gestiti da un servizio di directory, è necessario aggiungere anche i seguenti file JAR al percorso di classe:
  - `fscontext.jar`
  - `providerutil.jar`
- Se l'applicazione utilizza JTA, è necessario aggiungere anche `jta.jar` al percorso classe.

Nota: Questi file JAR aggiuntivi sono richiesti solo per la compilazione delle applicazioni, non per la loro esecuzione.

Gli script forniti con IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging utilizzano le seguenti variabili di ambiente:

MQ_JAVA_DATA_PATH
Questa variabile di ambiente specifica la directory per l'output di log e traccia.

PERCORSO_INSTALL_JAVA_MQ
Questa variabile di ambiente specifica la directory in cui è installato IBM MQ classes for JMS .

MQ_JAVA_LIB_PATH
Questa variabile di ambiente specifica la directory in cui sono memorizzate le librerie IBM MQ classes for JMS , come mostrato nelle tabelle precedenti.

Procedura

- Windows
  Su Windows, dopo l'installazione di IBM MQ, eseguire il comando **setmqenv**.
  Se non si esegue prima questo comando, potrebbe essere visualizzato il seguente messaggio di errore quando si immette un comando **dspmqver** :

  ```
  AMQ8351: IBM MQ non è stato configurato
  o la funzione JRE IBM MQ non è stata installata.
  ```

  Nota: Questo messaggio è previsto se non è stato installato IBM MQ Java runtime environment (JRE) (consultare Verifica dei prerequisiti delle funzioni aggiuntive di Windows.
- Linux AIX

Sui sistemi AIX and Linux , impostare le variabili di ambiente:

JMS 2.0 Per JMS 2.0, utilizzare uno dei seguenti script per impostare le variabili di ambiente:

- Se si utilizza una JVM a 32 bit, utilizzare lo script `setjmsenv`.
- Se si utilizza una JVM a 64 bit su un sistema AIX o Linux , utilizzare lo script `setjmsenv64`.

JM 3.0 Per Jakarta Messaging 3.0, utilizzare uno dei seguenti script per impostare le variabili di ambiente:

- Se si utilizza una JVM a 32 bit, utilizzare lo script `setjms30env`.
- Se si utilizza una JVM a 64 bit, utilizzare lo script `setjms30env64`.

Questi script si trovano nella directory *MQ_INSTALLATION_PATH*`/java/bin` , dove *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ .

È possibile utilizzare questi script in diversi modi. È possibile utilizzare lo script come base per l'impostazione delle variabili di ambiente richieste, come mostrato nella tabella, oppure aggiungerle a `.profile` utilizzando un editor di testo. Se si dispone di una configurazione non tipica, modificare il contenuto dello script come necessario. In alternativa, è possibile eseguire lo script in ogni sessione da cui devono essere eseguiti gli script di avvio JMS . Se si sceglie questa opzione, è necessario eseguire lo script in ogni finestra della shell avviata, durante il processo di verifica JMS :

- JMS 2.0 Per JMS 2.0, immettere `. ./setjmsenv` o `../setjmsenv64`.
- JM 3.0 Per Jakarta Messaging 3.0, immettere `. ./setjms30env` o `. ./setjms30env64`.

IBM i Su IBM i, è necessario impostare la variabile d'ambiente **`QIBM_MULTI_THREADED`** su Y. È quindi possibile eseguire le applicazioni a più thread nello stesso modo in cui si eseguono le applicazioni a singolo thread. Per ulteriori informazioni, consultare Impostazione di IBM MQ con Java e JMS.

Attività correlate

“Utilizzo di applicazioni di esempio IBM MQ classes for JMS” a pagina 122
Le applicazioni di esempio IBM MQ classes for JMS forniscono una panoramica delle funzioni comuni dell'API JMS . È possibile utilizzarle per verificare l'installazione e la configurazione del server di messaggistica e per creare le proprie applicazioni.

Riferimenti correlati

“Script forniti con IBM MQ classes for JMS/Jakarta Messaging” a pagina 125
Viene fornito un certo numero di script per assistere le attività comuni che devono essere eseguite quando si utilizzano IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Configurazione delle librerie JNI (Java Native Interface)

Le applicazioni IBM MQ classes for JMS , che si connettono a un gestore code utilizzando il trasporto dei collegamenti o che si connettono a un gestore code utilizzando il trasporto client e utilizzano i programmi di uscita del canale scritti in lingue diverse da Java, devono essere eseguite in un ambiente che consenta l'accesso alle librerie JNI (Java Native Interface).

Prima di iniziare

Consultare Configurazione del provider di messaggistica IBM MQ con le informazioni sulle librerie native per ulteriori informazioni sull'utilizzo dell'ambiente WebSphere Application Server .

Informazioni su questa attività

Per impostare questo ambiente, è necessario configurare il percorso della libreria dell'ambiente in modo che la JVM (Java Virtual Machine) possa caricare la libreria mqjbnd prima di avviare l'applicazione IBM MQ classes for JMS .

IBM MQ fornisce due librerie JNI (Java Native Interface):

mqjbnd
Questa libreria è utilizzata da applicazioni che si connettono a un gestore code utilizzando il trasporto dei bind. Fornisce l'interfaccia tra IBM MQ classes for JMS e il gestore code. La libreria mqjbnd installata con IBM MQ 9.3 può essere utilizzata per connettersi a qualsiasi gestore code IBM MQ 9.3 (o precedente).

mqjexitstub02
La libreria mqjexitstub02 viene caricata da IBM MQ classes for JMS quando un'applicazione si connette a un gestore code utilizzando il trasporto del client e utilizza un programma di uscita del canale scritto in un linguaggio diverso da Java.

Su alcune piattaforme, IBM MQ installa le versioni a 32 bit e a 64 bit di tali librerie JNI. La posizione delle librerie per ciascuna piattaforma viene mostrata nella Tabella 1.

Tabella 9. L'ubicazione delle librerie IBM MQ classes for JMS per ciascuna piattaforma

Piattaforma	**Directory contenente le librerie IBM MQ classes for JMS**
AIX Linux (POWER, x86-64 e piattaforme zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (librerie a 32 bit) *MQ_INSTALLATION_PATH*/java/lib64 (librerie a 64 bit)
Windows	*MQ_INSTALLATION_PATH*\java\lib (librerie a 32 bit) *MQ_INSTALLATION_PATH*\java\lib64 (librerie a 64 bit)
z/OS	*MQ_INSTALLATION_PATH*/java/lib (librerie a 31 bit e a 64 bit)

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Nota: z/OS Su z/OS, è possibile utilizzare un Java Virtual Machine (JVM) a 31 bit o a 64 bit. Non è necessario specificare quali librerie JNI utilizzare; IBM MQ classes for JMS può determinare autonomamente quali librerie JNI caricare.

Procedura

1. Configurare la proprietà **`java.library.path`** della JVM, che può essere eseguita in due modi:
   - Specificando l'argomento JVM come mostrato nel seguente esempio:

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux Ad esempio, per una JVM a 64 bit su Linux per un'installazione di ubicazione predefinita, specificare:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - Configurando l'ambiente della shell in modo che la JVM configuri il proprio `java.library.path`. Questo percorso varia in base alla piattaforma e all'ubicazione in cui è installato IBM MQ. Ad esempio, per una JVM a 64 bit e un'ubicazione di installazione IBM MQ predefinita, è possibile utilizzare le seguenti impostazioni:

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

Linux `export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH`

Windows `set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%`

Di seguito è riportato un esempio dello stack delle eccezioni che viene visualizzato quando l'ambiente non è stato configurato correttamente:

```
Causato da: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Impossibile caricare la libreria JNI nativa di WebSphere MQ : 'mqjbnd'.
    all'indirizzo com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    all'indirizzo com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    in java.security.AccessController.doPrivileged(AccessController.java:400)
    all'indirizzo com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    all'ubicazione com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    all'indirizzo com.ibm.mq.jmqi.local.LocalMQ. < init> (LocalMQ.java:1350)
    all'indirizzo com.ibm.mq.jmqi.local.LocalServer. < init> (LocalServer.java:230)
    in sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native nativo)
    in
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    all'indirizzo
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a:58)
    all'indirizzo java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    all'indirizzo com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    all'indirizzo com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    all'indirizzo
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 more
Causato da: java.lang.UnsatisfiedLinkError: mqjbnd (non trovato in java.library.path)
    all'indirizzo java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    all'indirizzo java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    in java.lang.System.loadLibrary(System.java:534)
    all'indirizzo com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... Altri 20
```

2. Una volta impostato l'ambiente a 32 bit o a 64 bit, avviare l'applicazione IBM MQ classes for JMS utilizzando il comando:

   ```
   java application-name
   ```

   dove *nome - applicazione* è il nome dell'applicazione IBM MQ classes for JMS da eseguire.

   Un'eccezione contenente il codice di errore IBM MQ 2495 (MQRC_MODULE_NOT_FOUND) viene generata da IBM MQ classes for JMS se:

   - L'applicazione IBM MQ classes for JMS viene eseguita in un Java runtime environmenta 32 bit ed è stato impostato un ambiente a 64 bit per IBM MQ classes for JMS, in quanto Java runtime environment a 32 bit non è in grado di caricare la Java Native Library a 64 bit.
   - L'applicazione IBM MQ classes for JMS viene eseguita in un Java runtime environmenta 64 bit ed è stato impostato un ambiente a 32 bit per IBM MQ classes for JMS, poiché Java runtime environment a 64 bit non è in grado di caricare la Java Native Library a 32 bit.

Il file di configurazione IBM MQ classes for JMS/Jakarta Messaging
I file di configurazione di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging specificano le proprietà utilizzate per configurare IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Nota: Le proprietà definite nel file di configurazione possono essere impostate anche come proprietà del sistema JVM. Se una proprietà è impostata sia nel file di configurazione che come proprietà di sistema, la proprietà di sistema ha la precedenza. Pertanto, se richiesto, è possibile sovrascrivere qualsiasi proprietà nel file di configurazione specificandola come proprietà di sistema nel comando **java** .

Il formato di un file di configurazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging è quello di un file delle proprietà Java standard. Un file di configurazione di esempio denominato `jms.config` viene fornito nella sottodirectory `bin` della directory di installazione di IBM MQ classes for JMS . Questo file documenta tutte le proprietà supportate e i relativi valori predefiniti.

È possibile scegliere il nome e il percorso di un file di configurazione di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Quando si avvia l'applicazione, utilizzare un comando **java** con il seguente formato:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

Nel comando, *config_file_url* è un URL (Uniform Resource Locator) (URL) che specifica il nome e l'ubicazione del file di configurazione di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Sono supportati URL dei seguenti tipi: http, file, ftp e jar.

Di seguito è riportato un esempio di comando **java** :

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

Questo comando identifica il file di configurazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging come il file `D:\mydir\mjms.config` sul sistema Windows locale.

Quando un'applicazione viene avviata, IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging legge il contenuto del file di configurazione e memorizza le proprietà specificate in un archivio delle proprietà interno. Se il comando **java** non identifica un file di configurazione o se non è possibile trovare il file di configurazione, IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging utilizza i valori predefiniti per tutte le proprietà.

È possibile utilizzare un file di configurazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging con uno qualsiasi dei trasporti supportati tra un'applicazione e un gestore code o broker.

Sostituzione delle proprietà specificate in un file di configurazione IBM MQ MQI client

Un file di configurazione IBM MQ MQI client può anche specificare le proprietà utilizzate per configurare IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging. Tuttavia, le proprietà specificate in un file di configurazione IBM MQ MQI client si applicano solo quando un'applicazione si connette a un gestore code in modalità client.

Se necessario, è possibile sovrascrivere qualsiasi attributo in un file di configurazione IBM MQ MQI client specificandolo come proprietà in un file di configurazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Per sovrascrivere un attributo in un file di configurazione IBM MQ MQI client , utilizzare una voce con il formato seguente nel file di configurazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging :

```
com.ibm.mq.cfg. stanza. propName = propValue
```

Le variabili nella voce hanno i seguenti significati:

stanza
Il nome della sezione nel file di configurazione IBM MQ MQI client che contiene l'attributo

propName
Il nome dell'attributo come specificato nel file di configurazione IBM MQ MQI client

propValue
Il valore della proprietà che sovrascrive il valore dell'attributo specificato nel file di configurazione IBM MQ MQI client

In alternativa, è possibile sovrascrivere un attributo in un file di configurazione IBM MQ MQI client specificando la proprietà come proprietà di sistema nel comando **java** . Utilizzare il formato precedente per specificare la proprietà come proprietà di sistema.

Solo i seguenti attributi in un file di configurazione IBM MQ MQI client sono rilevanti per IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging. Se si specificano o si sovrascrivono altri attributi, non ha alcun effetto. In particolare, si noti che `ChannelDefinitionFile` e `ChannelDefinitionDirectory` nella stanza CHANNELS del file di configurazione client non vengono utilizzati. Consultare “Utilizzo di una

tabella di definizione di canale client con IBM MQ classes for JMS” a pagina 286 per i dettagli su come utilizzare CCDT con IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Tabella 10. Quale stanza del file di configurazione client contiene quale attributo

stanza	**Attributo**
Stanza CHANNELS del file di configurazione del client	Put1DefaultAlwaysSync
Stanza CHANNELS del file di configurazione del client	DefRecon
Stanza CHANNELS del file di configurazione del client	ReconDelay
Stanza CHANNELS del file di configurazione del client	PasswordProtection
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath64
ClientExitStanza del percorso del file di configurazione client	JavaExitsClasspath
Stanza JMQI del file di configurazione del client	useMQCSPauthentication
Stanza MessageBuffer del file di configurazione client	MaximumSize
Stanza MessageBuffer del file di configurazione client	PurgeTime
Stanza MessageBuffer del file di configurazione client	UpdatePercentage
Stanza TCP del file di configurazione client	ClntRcvBufSize
Stanza TCP del file di configurazione client	ClntSndBufSize
Stanza TCP del file di configurazione client	Timeout connessione
Stanza TCP del file di configurazione client	KeepAlive

Per ulteriori dettagli sulla configurazione IBM MQ MQI client , consultare il file di configurazione IBM MQ MQI client , `mqclient.ini`

Utilizzo della traccia dell'ambiente standard Java per configurare la traccia JMS
Utilizzare la stanza Java Standard Environment Trace Settings per configurare la funzione di traccia IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging .

`com.ibm.msg.client.commonservices.trace.outputName` = ***traceOutputNome***
`traceOutputName` è la directory e il nome file a cui viene inviato l'output di traccia.

Per impostazione predefinita, le informazioni di traccia vengono scritte in un file di traccia nella directory di lavoro corrente dell'applicazione. Il nome del file di traccia dipende dall'ambiente in cui è in esecuzione l'applicazione:

- V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, se l'applicazione ha caricato IBM MQ classes for Jakarta Messaging dal file JAR riposizionabile `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) o IBM MQ classes for JMS dal file JAR riposizionabile `com.ibm.mq.allclient.jar` (JMS 2.0), la traccia viene scritta in un file denominato `mqjavaclient_%PID%.cl%u.trc`.

- Da IBM MQ 9.1.5 e IBM MQ 9.1.0 Fix Pack 5:
  - Se l'applicazione ha caricato il IBM MQ classes for JMS dal file JAR riposizionabile `com.ibm.mq.allclient.jar`, la traccia viene scritta in un file denominato `mqjavaclient_%PID%.cl%u.trc`.
  - Se l'applicazione ha caricato IBM MQ classes for JMS dal file JAR `com.ibm.mqjms.jar`, la traccia viene scritta in un file denominato `mqjava_%PID%.cl%u.trc`.
- Da IBM MQ 9.0.0 Fix Pack 2:
  - Se l'applicazione ha caricato il IBM MQ classes for JMS dal file JAR riposizionabile `com.ibm.mq.allclient.jar`, la traccia viene scritta in un file denominato `mqjavaclient_%PID%.trc`.
  - Se l'applicazione ha caricato IBM MQ classes for JMS dal file JAR `com.ibm.mqjms.jar`, la traccia viene scritta in un file denominato `mqjava_%PID%.trc`.
- Per IBM MQ classes for JMS per IBM MQ 9.0.0 Fix Pack 1 o versioni precedenti, la traccia viene scritta in un file denominato `mqjms_%PID%.trc`.

dove *%PID%* è l'identificativo del processo dell'applicazione di cui viene eseguita la traccia e *%u* è un numero univoco per differenziare i file tra i thread che eseguono la traccia in diversi programmi di caricamento classi Java .

Se un ID processo non è disponibile, viene generato un numero casuale con la lettera `f`come prefisso. Per includere l'ID processo in un nome file specificato, utilizzare la stringa `%PID%`.

Se si specifica una directory alternativa, è necessario che esista e che si disponga dell'autorizzazione di scrittura per questa directory. Se non si dispone dell'autorizzazione di scrittura, l'output di traccia viene scritto in `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList è un elenco di package e classi di cui viene eseguita la traccia o i valori speciali `ALL` o `NONE`.

Separare i nomi pacchetto o classe con un punto e virgola, `;`. *includeList* assume il valore predefinito `ALL`e traccia tutti i package e le classi in IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Nota: È possibile includere un package, ma escludere i package secondari di tale package. Ad esempio, se si include il pacchetto `a.b` e si esclude il pacchetto `a.b.x`, la traccia include tutto in `a.b.y` e `a.b.z`, ma non in `a.b.x` o `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList è un elenco di package e classi di cui non viene eseguita la traccia o i valori speciali `ALL` o `NONE`.

Separare i nomi pacchetto o classe con un punto e virgola, `;`. *excludeList* assume il valore predefinito `NONE`e pertanto non esclude dalla traccia alcun package e classe in IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

Nota: È possibile escludere un package ma includere i package secondari di tale package. Ad esempio, se si esclude il pacchetto `a.b` e si include il pacchetto `a.b.x`, la traccia include tutto in `a.b.x` e `a.b.x.1`, ma non `a.b.y` o `a.b.z`.

Sono inclusi tutti i package o le classi specificati, allo stesso livello, inclusi ed esclusi.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayByte*
maxArrayBytes è il numero massimo di byte tracciati da qualsiasi array di byte.

Se *maxArrayBytes* è impostato su un numero intero positivo, limita il numero di byte in una schiera di byte scritti nel file di traccia. Tronca la schiera di byte dopo la scrittura di *maxArrayBytes* . L'impostazione *maxArrayBytes* riduce la dimensione del file di traccia risultante e l'effetto della traccia sulle prestazioni dell'applicazione.

Un valore di `0` per questa proprietà indica che nessuno dei contenuti delle schiere di byte viene inviato al file di traccia.

Il valore predefinito è `-1`, che elimina qualsiasi limite sul numero di byte in una schiera di byte inviati al file di traccia.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceByte*
maxTraceBytes è il numero massimo di byte scritti in un file di output di traccia.

maxTraceBytes funziona con *traceCycles*. Se il numero di byte di traccia scritti è vicino al limite, il file viene chiuso e viene avviato un nuovo file di output di traccia.

Un valore `0` indica che un file di output di traccia ha lunghezza zero. Il valore predefinito è `-1`, che significa che la quantità di dati da scrivere in un file di output di traccia è illimitata.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles è il numero di file di output di traccia da scorrere.

Se il file di output di traccia corrente raggiunge il limite specificato da *maxTraceBytes*, il file viene chiuso. Un ulteriore output di traccia viene scritto nel successivo file di output di traccia in sequenza. Ogni file di output di traccia è distinto da un suffisso numerico aggiunto al nome file. Il file di output di traccia corrente o più recente è `mqjms.trc.0`, il successivo file di output di traccia più recente è `mqjms.trc.1`. I file di traccia più vecchi seguono lo stesso schema di numerazione fino al limite.

Il valore predefinito di *traceCycles* è 1. Se *traceCycles* è 1, quando il file di output di traccia corrente raggiunge la dimensione massima, il file viene chiuso ed eliminato. Viene avviato un nuovo file di output di traccia con lo stesso nome. Pertanto, esiste un solo file di output di traccia alla volta.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters controlla se i parametri del metodo e i valori di ritorno sono inclusi nella traccia.

traceParameters assume il valore predefinito TRUE. Se *traceParameters* è impostato su FALSE, viene eseguita la traccia solo delle firme del metodo.

com.ibm.msg.client.commonservices.trace.startup = *avvio*
Esiste una fase di inizializzazione di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging durante la quale vengono assegnate le risorse. La funzione di traccia principale viene inizializzata durante la fase di allocazione delle risorse.

Se *startup* è impostato su TRUE, viene utilizzata la traccia di avvio. Le informazioni di traccia vengono prodotte immediatamente e includono la configurazione di tutti i componenti, inclusa la funzionalità di traccia stessa. Le informazioni di traccia di avvio possono essere utilizzate per diagnosticare i problemi di configurazione. Le informazioni sulla traccia di avvio vengono sempre scritte in `System.err`.

startup assume il valore predefinito FALSE.

startup viene controllato prima che l'inizializzazione sia completa. Per questo motivo, specificare la proprietà sulla riga comandi solo come proprietà di sistema Java . Non specificarlo nel file di configurazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Impostare *compressedTrace* su TRUE per comprimere l'output di traccia.

Il valore predefinito di *compressedTrace* è FALSE.

Se *compressedTrace* è impostato su TRUE, l'output di traccia viene compresso. Il nome file di output di traccia predefinito ha l'estensione `.trz`. Se la compressione è impostata su FALSE, il valore predefinito, il file ha l'estensione `.trc` per indicare che è decompresso. Tuttavia, se il nome file per l'output di traccia è stato specificato in *traceOutputName* , viene utilizzato tale nome; al file non viene applicato alcun suffisso.

L'output della traccia compressa è più piccolo di quello decompresso. Poiché c'è meno I/O, può essere scritto più velocemente della traccia non compressa. La traccia compressa ha un effetto minore sulle prestazioni di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging rispetto alla traccia non compressa.

Se *maxTraceBytes* e *traceCycles* sono impostati, vengono creati più file di traccia compressi invece di più file flat.

Se IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging termina in modo non controllato, un file di traccia compresso potrebbe non essere valido. Per questo motivo, la compressione della traccia deve essere utilizzata solo quando IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging si chiude in modo controllato. Utilizzare la compressione della traccia solo se i problemi esaminati non causano l'arresto imprevisto della stessa JVM. Non utilizzare la compressione di traccia quando si diagnosticano problemi che possono causare `System.Halt()` arresti o terminazioni JVM anomale e non controllate.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel specifica un livello di filtro per la traccia. I livelli di traccia definiti sono i seguenti:

- `TRACE_NONE: 0`
- `TRACE_EXCEPTION: 1`
- `TRACE_WARNING: 3`
- `TRACE_INFO: 6`
- `TRACE_ENTRYEXIT: 8`
- `TRACE_DATA: 9`
- `TRACE_ALL: Integer.MAX_VALUE`

Ogni livello di traccia include tutti i livelli inferiori. Ad esempio, se il livello di traccia è impostato su `TRACE_INFO`, qualsiasi punto di traccia con un livello definito `TRACE_EXCEPTION`, `TRACE_WARNINGo` `TRACE_INFO` viene scritto nella traccia. Tutti gli altri punti di traccia sono esclusi.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace controlla se il servizio di traccia del client IBM MQ JMS viene utilizzato in un ambiente WebSphere Application Server .

Se *standaloneTrace* è impostato su TRUE, le proprietà di traccia del client IBM MQ JMS vengono utilizzate per determinare la configurazione della traccia.

Se *standaloneTrace* è impostato su FALSE, e il client IBM MQ JMS è in esecuzione in un contenitore WebSphere Application Server , viene utilizzato il servizio di traccia WebSphere Application Server . Le informazioni di traccia generate dipendono dalle impostazioni di traccia del server delle applicazioni.

Il valore predefinito di *standaloneTrace* è FALSE.

Stanza di registrazione
Utilizzare la stanza di registrazione per configurare la funzione di registrazione IBM MQ classes for JMS .

Le seguenti proprietà possono essere incluse nella stanza di registrazione:

com.ibm.msg.client.commonservices.log.outputName = percorso
Il nome del file di log utilizzato dalla funzione di log IBM MQ classes for JMS . Il valore predefinito è `mqjms.log`, che viene scritto nella directory di lavoro corrente per Java Runtime Environment in cui è in esecuzione IBM MQ classes for JMS .

La proprietà può assumere uno dei seguenti valori:

- un nome percorso singolo
- un elenco separato da virgole di nomi percorso (tutti i dati vengono registrati in tutti file)

Ogni nome percorso può essere un nome percorso assoluto o relativo oppure:

"stderr" o "System.err"
Rappresenta il flusso di errore standard.

"stdout" o "System.out"
Rappresenta il flusso di output standard.

com.ibm.msg.client.commonservices.log.maxBytes
Il numero massimo di byte registrati da qualsiasi chiamata ai dati del messaggio di log.

Intero positivo
I dati vengono scritti fino al valore di byte per chiamata di log.

0
Non viene scritto alcun dato.

-1
Vengono scritti dati illimitati (valore predefinito).

com.ibm.msg.client.commonservices.log.limit
Il numero massimo di byte scritti in qualsiasi file di log (il valore predefinito è 262144).

Intero positivo
I dati vengono scritti fino al valore di byte per file di log.

0
Non viene scritto alcun dato.

-1
Vengono scritti dati illimitati.

com.ibm.msg.client.commonservices.log.count
Il numero di file di log da scorrere. Poiché ogni file raggiunge la traccia `com.ibm.msg.client.commonservices.trace.limit` inizierà nel file successivo, il valore predefinito è 3.

Intero positivo
Numero di file da scorrere.

0
Un singolo file.

Stanza Java SE Specifics
Utilizzare la stanza Java SE Specifics per configurare proprietà utilizzate quando IBM MQ classes for JMS viene utilizzato in un ambiente Java Standard Edition .

com.ibm.msg.client.commonservices.j2se.produceJavaCore = VERO|FALSO
Determina se un file JavaCore viene scritto immediatamente dopo che IBM MQ classes for JMS ha generato un file FDC. Se è impostato su TRUE, viene prodotto un file di base Javanella directory di lavoro di Java Runtime Environment in cui sono in esecuzione IBM MQ classes for JMS .

VERO
Generare JavaCore, in base alla capacità di Java Runtime Environment di farlo.

FALSO
Non generare JavaCore; questo è il valore predefinito.

Stanza IBM MQ Properties
Utilizzare la stanza IBM MQ Properties per impostare le proprietà che influenzano il modo in cui IBM MQ classes for JMS interagisce con IBM MQ.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
Quando un'applicazione che utilizza il IBM MQ classes for JMS si connette a un gestore code IBM MQ utilizzando la modalità di migrazione del provider di messaggi IBM MQ , IBM MQ classes for JMS utilizza una dimensione buffer predefinita di 4 KB quando riceve i messaggi. Se il messaggio che l'applicazione sta tentando di ottenere è maggiore di 4 KB, IBM MQ classes for JMS ridimensiona il buffer in modo che sia abbastanza grande da contenere il messaggio. La dimensione del buffer maggiore viene quindi utilizzata quando vengono ricevuti i messaggi successivi.

Questa proprietà controlla quando la dimensione del buffer viene ridotta nuovamente a 4 KB. Per impostazione predefinita, quando vengono ricevuti dieci messaggi consecutivi inferiori alla dimensione del buffer maggiore, la dimensione del buffer viene ridotta a 4 KB. Per ripristinare la dimensione del buffer a 4 KB ogni volta che viene ricevuto un messaggio, impostare la proprietà sul valore 0.

0

Il buffer viene sempre reimpostato sulla dimensione predefinita.

10

Questo è il valore predefinito. Il buffer verrà ridimensionato dopo il decimo messaggio.

com.ibm.msg.client.wmq.receiveConversionCCSID

Quando un'applicazione che utilizza IBM MQ classes for JMS si connette a un gestore code IBM MQ utilizzando la modalità normale del provider di messaggistica IBM MQ , è possibile impostare la proprietà `receiveConversionCCSID` per sovrascrivere il valore CCSID predefinito nella struttura MQMD utilizzata per ricevere i messaggi dal gestore code. Per impostazione predefinita, MQMD contiene un campo CCSID impostato su 1208, ma questo può essere modificato se, ad esempio, il gestore code non è in grado di convertire i messaggi in questa codepage.

I valori validi sono qualsiasi numero CCSID valido o uno dei seguenti valori:

-1

Utilizzare il valore predefinito della piattaforma.

1208

Questo è il valore predefinito.

Stanza specifica modalità client

Utilizzare la stanza specifica della modalità client per specificare le proprietà utilizzate quando IBM MQ classes for JMS si connette a un gestore code che utilizza il trasporto CLIENT.

com.ibm.mq.polling.RemoteRequestEntry

Specifica l'intervallo di polling utilizzato da IBM MQ classes for JMS per controllare le connessioni interrotte quando è in attesa di risposta da un gestore code.

Intero positivo

Il numero di millisecondi da attendere prima del controllo. Il valore predefinito è 10000 o 10 secondi. Il valore minimo è 3000 e i valori più bassi vengono trattati allo stesso modo di questo valore minimo.

Proprietà utilizzate per configurare il comportamento del client JMS

Utilizzare queste proprietà per configurare il comportamento del client JMS .

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE

La specifica JMS 2.0 introduce modifiche al modo in cui funzionano determinati comportamenti. IBM MQ 8.0 include la proprietà `com.ibm.mq.jms.SupportMQExtensions`, che può essere impostata su *TRUE*, per ripristinare questi comportamenti modificati alle implementazioni precedenti. Il ripristino dei comportamenti modificati potrebbe essere necessario per le applicazioni JMS 2.0 e anche per alcune applicazioni che utilizzano l'API JMS 1.1 ma vengono eseguite rispetto a IBM MQ 8.0 IBM MQ classes for JMS.

VERO

Le seguenti tre aree di funzionalità vengono ripristinate impostando SupportMQExtensions su TRUE:

Priorità messaggio

Ai messaggi può essere assegnata la priorità *0* - *9*. Prima di JMS 2.0, i messaggi potevano anche utilizzare il valore *-1*, che indica che viene utilizzata la priorità predefinita della coda. JMS 2.0 non consente l'impostazione di una priorità di messaggio *-1* . L'attivazione di `SupportMQExtensions` consente di utilizzare il valore *-1* .

ID client

La specifica JMS 2.0 richiede che gli ID client non null vengano controllati per l'univocità quando effettuano una connessione. L'attivazione di `SupportMQExtensions`significa che questo requisito viene ignorato e che un ID client può essere riutilizzato.

`NoLocal`

La specifica JMS 2.0 richiede che quando questa costante è attivata, un utente non può ricevere messaggi pubblicati dallo stesso ID client. Prima di JMS 2.0, questo attributo era

impostato su un sottoscrittore per evitare che ricevesse messaggi pubblicati dalla propria connessione. L'attivazione di `SupportMQExtensions` ripristina questo comportamento alla sua implementazione precedente.

FALSO
I cambiamenti di comportamento vengono mantenuti.

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
Da IBM MQ 8.0.0 Fix Pack 2, dopo che un'applicazione ha inviato un messaggio Byte o Flusso, IBM MQ classes for JMS può impostare lo stato del messaggio che è stato appena inviato su sola lettura o sola scrittura.

VERO
Gli oggetti vengono impostati per la sola lettura dopo l'invio. L'impostazione di questo valore mantiene la compatibilità con la specifica JMS 2.0

FALSO
Gli oggetti sono impostati per la scrittura solo dopo l'invio. Questo è il valore predefinito.

Concetti correlati

“proprietà SupportMQExtensions” a pagina 330
La specifica JMS 2.0 ha introdotto modifiche al modo in cui funzionano determinati comportamenti. IBM MQ 8.0 e versioni successive include la proprietà **com.ibm.mq.jms.SupportMQExtensions**, che può essere impostata su TRUE per ripristinare questi comportamenti modificati alle implementazioni precedenti.

Configurazione STEPLIB per IBM MQ classes for JMS su z/OS
Su z/OS, la STEPLIB utilizzata in fase di runtime deve contenere le librerie IBM MQ SCSQAUTH e SCSQANLE. Specificare queste librerie nel JCL di avvio o utilizzando il file `.profile` .

Da z/OS UNIX System Services, puoi aggiungerli utilizzando una riga nel `.profile` come mostrato nel seguente frammento di codice, sostituendo `thlqual` con il qualificatore del dataset di livello superiore che hai scelto quando hai installato IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In altri ambienti, di solito è necessario modificare il JCL di avvio per includere SCSQAUTH e SCSQANLE nella concatenazione STEPLIB:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS e strumenti di gestione software
Gli strumenti di gestione software come Apache Maven possono essere utilizzati con IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Molte grandi organizzazioni di sviluppo utilizzano questi strumenti per gestire centralmente i repository di librerie di terze parti.

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono composti da un numero di file JAR. Quando si sviluppano le applicazioni del linguaggio Java utilizzando questa API, è richiesta un'installazione di IBM MQ Server, IBM MQ Client o IBM MQ Client SupportPac sulla macchina su cui si sta sviluppando l'applicazione.

Se si desidera utilizzare tale strumento e aggiungere i file JAR che costituiscono IBM MQ classes for JMS a un repository gestito centralmente, è necessario osservare i seguenti punti:

- Un repository o un contenitore deve essere reso disponibile solo agli sviluppatori all'interno della propria organizzazione. Non è consentita alcuna distribuzione al di fuori dell'organizzazione.
- Il repository deve contenere una serie completa e coerente di file JAR da una singola release IBM MQ o Fix Pack.

- L'utente è responsabile dell'aggiornamento del repository con qualsiasi manutenzione fornita dal supporto IBM .

I file JAR seguenti devono essere installati nel repository:

- JMS 2.0 `com.ibm.mq.allclient.jar` e `jms.jar` sono obbligatori se si utilizza IBM MQ classes for JMS.
- JM 3.0 `com.ibm.mq.jakarta.client.jar` e `jakarta.jms-api.jar` sono obbligatori se si utilizza IBM MQ classes for Jakarta Messaging.
- `fscontext.jar` è richiesto se si utilizza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging e si accede agli oggetti gestiti JMS memorizzati in un contesto JNDI del file system.
- `providerutil.jar` è richiesto se si sta utilizzando IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging e si sta accedendo agli oggetti gestiti JMS memorizzati in un contesto JNDI del file system.
- Il provider di sicurezza Bouncy Castle e i file JAR di supporto CMS sono richiesti per il supporto per JRE nonIBM . Per ulteriori informazioni, vedi Support for non -IBM JREs.

Esecuzione delle applicazioni IBM MQ classes for JMS in Java security manager

IBM MQ classes for JMS può essere eseguito con Java security manager abilitato. Per eseguire correttamente le applicazioni con Java security manager abilitato, è necessario configurare Java Virtual Machine (JVM) con un file di configurazione della politica adatto.

Il modo più semplice per creare un file di definizione della politica adatto è modificare il file di configurazione della politica fornito con Java runtime environment (JRE). Sulla maggior parte dei sistemi, questo file si trova nella directory `lib/security/java.policy` relativa alla propria directory JRE. È possibile modificare il file di configurazione della politica utilizzando l'editor preferito o il programma dello strumento della politica fornito con il JRE.

File di configurazione della politica di esempio

Di seguito viene riportato un esempio di un file di configurazione della politica che consente a IBM MQ classes for JMS di essere eseguito correttamente nel gestore della sicurezza predefinito. Questo file dovrà essere personalizzato, per specificare le ubicazioni di alcuni file e directory: *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ , *MQ_DATA_DIRECTORY* rappresenta l'ubicazione della directory di dati di MQ e *QM_NAME* è il nome del gestore code per cui è stato configurato l'accesso.

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
  permission javax.management.MBeanPermission "*","*";

  // We need to be able to read manifests etc from the jar files in the installation directory
  permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";
```

```
   //Required if mqclient.ini/mqs.ini configuration files are used
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

   //For the client transport type.
   permission java.net.SocketPermission "*","connect,resolve";

   //For the bindings transport type.
   permission java.lang.RuntimePermission "loadLibrary.*";

   //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

   //For applications that use User Exits
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
   permission java.lang.RuntimePermission "createClassLoader";

   //Required for the z/OS platform
   permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

   // Used by the internal ConnectionFactory implementation
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

   // Used for controlled class loading
   permission java.lang.RuntimePermission "setContextClassLoader";

   // Used to default the Application name in Client mode connections
   permission java.util.PropertyPermission "sun.java.command","read";

   // Used by the IBM JSSE classes
   permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

   //Required to determine if an IBM Java Runtime is running in FIPS mode,
   //and to modify the property values status as required.
   permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
   permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
   //Required if an IBM FIPS provider is to be used for SSL communication.
   permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

   // Required for non-IBM Java Runtimes that establish secure client
   // transport mode connections using mutual TLS authentication
   permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
   permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

Nell'esempio, l'istruzione `grant` contiene le autorizzazioni richieste da IBM MQ classes for JMS. Per utilizzare queste istruzioni di concessione nel file di configurazione della politica, potrebbe essere necessario modificare i nomi percorso in base a dove è stato installato IBM MQ classes for JMS e dove si memorizzano le applicazioni.

Le applicazioni di esempio fornite con IBM MQ classes for JMSe gli script per eseguirle non abilitano il gestore della sicurezza.

Importante:

La funzione di traccia IBM MQ classes for JMS richiede ulteriori autorizzazioni in quanto esegue ulteriori query delle proprietà di sistema e ulteriori operazioni del file system.

Un file della politica di sicurezza del template adatto per l'esecuzione in un gestore sicurezza con la traccia abilitata viene fornito nella directory `samples/wmqjava` dell'installazione di IBM MQ come `example.security.policy`.

Configurazione post - installazione per applicazioni IBM MQ classes for JMS

Questo argomento indica le autorizzazioni necessarie alle applicazioni IBM MQ classes for JMS per accedere alle risorse di un gestore code. Inoltre, introduce le modalità di connessione e descrive come configurare un gestore code in modo che le applicazioni possano connettersi in modalità client.

Verificare il file readme IBM MQ . Potrebbe contenere informazioni che sostituiscono le informazioni contenute in questo argomento.

Oggetti utilizzati da JMS che richiedono l'autorizzazione per utenti non privilegiati
Gli utenti non privilegiati hanno bisogno dell'autorizzazione concessa per accedere alle code utilizzate da JMS. Ogni applicazione JMS necessita dell'autorizzazione per il gestore code con cui opera.

Per i dettagli sul controllo degli accessi in IBM MQ, vedi Impostazione della sicurezza.

Le applicazioni IBM MQ classes for JMS richiedono l'autorizzazione `connect` e `inq` al gestore code. È possibile impostare le autorizzazioni appropriate utilizzando il comando di controllo **setmqaut** , ad esempio:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Per il dominio point - to - point, sono richieste le seguenti autorizzazioni:

- Le code utilizzate dagli oggetti MessageProducer richiedono l'autorità `put` .
- Le code utilizzate dagli oggetti MessageConsumer e QueueBrowser richiedono le autorizzazioni `get`, `inq`e `browse` .
- Il metodo QueueSession.createTemporaryQueue () deve accedere alla coda modello specificata dalla proprietà TEMPMODEL dell'oggetto factory QueueConnection. Per impostazione predefinita, questa coda modello è SYSTEM.TEMP.MODEL.QUEUE.

Se una di queste code è una coda alias, le relative code di destinazione richiedono l'autorizzazione di interrogazione. Se la coda di destinazione è una coda cluster, è necessaria anche l'autorizzazione di ricerca.

Per il dominio di pubblicazione / sottoscrizione, le seguenti code vengono utilizzate se IBM MQ classes for JMS si sta connettendo a un gestore code IBM MQ in modalità di migrazione del provider di messaggistica IBM MQ :

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Per ulteriori informazioni sulla modalità di migrazione del fornitore di messaggistica IBM MQ , consultare Configurazione della proprietà JMS **PROVIDERVERSION**

Inoltre, se IBM MQ classes for JMS si connette a un gestore code in questa modalità, qualsiasi applicazione che pubblica messaggi deve accedere alla coda di flusso specificata dal factory TopicConnectiono dall'oggetto argomento. Per impostazione predefinita, questa coda è SYSTEM.BROKER.DEFAULT.STREAM.

Se si utilizza ConnectionConsumer, IBM MQ Resource Adapter o il fornitore di messaggistica WebSphere Application Server IBM MQ , potrebbe essere necessaria un'ulteriore autorizzazione.

Le code che devono essere lette da ConnectionConsumer devono avere le autorizzazioni `get`, `inq`e `browse` . La coda di messaggi non recapitabili del sistema e qualsiasi coda di backout-requeue o coda di report utilizzata da ConnectionConsumer devono disporre delle autorizzazioni `put` e `passall` .

Quando un'applicazione utilizza la modalità normale del provider di messaggistica IBM MQ per eseguire la messaggistica di pubblicazione / sottoscrizione, l'applicazione utilizza la funzionalità di pubblicazione / sottoscrizione integrata fornita dal gestore code. Per informazioni sulla protezione degli argomenti e delle code utilizzati, consultare Sicurezza di pubblicazione / sottoscrizione .

Modalità di connessione per IBM MQ classes for JMS
Un'applicazione IBM MQ classes for JMS può connettersi a un gestore code in modalità client o bind. In modalità client, IBM MQ classes for JMS si connette al gestore code tramite TCP/IP. In modalità bind, IBM MQ classes for JMS si connette direttamente al gestore code utilizzando JNI (Java Native Interface).

Un'applicazione in esecuzione in WebSphere Application Server su z/OS può connettersi a un gestore code in modalità bind o client, ma un'applicazione in esecuzione in qualsiasi altro ambiente su z/OS può connettersi a un gestore code solo in modalità bind. Un'applicazione in esecuzione su qualsiasi altra piattaforma può connettersi a un gestore code in modalità bind o client.

È possibile utilizzare la versione corrente o precedente supportata di IBM MQ classes for JMS con un gestore code corrente ed è possibile utilizzare una versione corrente o precedente supportata del gestore code con la versione corrente di IBM MQ classes for JMS. Se si mischiano versioni differenti, la funzione è limitata al livello della versione precedente.

Le seguenti sezioni descrivono ogni modalità di connessione in modo più dettagliato.

Modalità client

Per connettersi a un gestore code in modalità client, un'applicazione IBM MQ classes for JMS può essere eseguita sullo stesso sistema su cui è in esecuzione il gestore code o su un sistema differente. In ogni caso IBM MQ classes for JMS si connette al gestore code tramite TCP/IP.

Modalità bind

Per connettersi a un gestore code in modalità bind, un'applicazione IBM MQ classes for JMS deve essere eseguita sullo stesso sistema su cui è in esecuzione il gestore code.

IBM MQ classes for JMS si connette direttamente al gestore code utilizzando JNI (Java Native Interface). Per utilizzare il trasporto dei collegamenti, IBM MQ classes for JMS deve essere eseguito in un ambiente che abbia accesso alle librerie IBM MQ Java Native Interface; consultare “Configurazione delle librerie JNI (Java Native Interface)” a pagina 98 per ulteriori informazioni.

IBM MQ classes for JMS supporta i seguenti valori per *ConnectOption*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Per modificare le opzioni di connessione utilizzate da IBM MQ classes for JMS, modificare la proprietà del factory di connessione CONNOPT.

Per ulteriori informazioni sulle opzioni di connessione, consultare “Connessione a un gestore code utilizzando la chiamata MQCONNX” a pagina 738

Per utilizzare il trasporto dei collegamenti, Java Runtime Environment utilizzato deve supportare il CCSID (Coded Character Set Identifier) del gestore code a cui si connette IBM MQ classes for JMS .

I dettagli su come determinare quali CCSID sono supportati da un Java Runtime Environment sono disponibili in IBM MQ FDC con ID probe 21 generato quando si utilizzano le classi IBM MQ V7 per Java o IBM MQ V7 per JMS.

Configurazione del gestore code in modo che le applicazioni IBM MQ classes for JMS possano connettersi in modalità client
Per configurare il gestore code in modo che le applicazioni IBM MQ classes for JMS possano connettersi in modalità client, è necessario creare una definizione di canale di connessione server e avviare un listener.

Creazione di una definizione di canale di connessione server

Su tutte le piatteforme, è possibile utilizzare il comando MQSC DEFINE CHANNEL per creare una definizione di canale di connessione server. Vedi il seguente esempio:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i Su IBM i, è possibile utilizzare il comando CL CRTMQMCHL, come nel seguente esempio:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

In questo comando, *QMGRNAME* è il nome del gestore code.

Linux Windows Su Linux e Windows, è possibile anche creare una definizione di canale di connessione server utilizzando IBM MQ Explorer.

z/OS Su z/OS è possibile utilizzare le operazioni e i pannelli di controllo per creare una definizione di canale di connessione server.

Il nome del canale (JAVA.CHANNEL negli esempi precedenti) deve corrispondere al nome canale specificato dalla proprietà CHANNEL della factory di connessione utilizzata dall'applicazione per connettersi al gestore code. Il valore predefinito della proprietà CHANNEL è SYSTEM.DEF.SVRCONN.

Avvio di un listener

È necessario avviare un listener per il gestore code se non ne è già stato avviato uno.

Multi In Multiplatforms, è possibile utilizzare il comando MQSC START LISTENER per avviare un listener dopo aver creato prima un oggetto listener utilizzando il comando MQSC DEFINE LISTENER, come mostrato nel seguente esempio:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS Su z/OS, si utilizza solo il comando START LISTENER, come nel seguente esempio, ma si noti che lo spazio di indirizzo dell'iniziatore di canale deve essere avviato prima di poter iniziare un listener:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i Su IBM i, è anche possibile utilizzare il comando CL STRMQMLSR per avviare un listener, come nel seguente esempio:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

In questo comando, *QMGRNAME* è il nome del gestore code.

ALW Su AIX, Linux, and Windows, è anche possibile utilizzare il comando di controllo **runmqlsr** per avviare un listener, come nel seguente esempio:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

In questo comando, *QMgrName* è il nome del gestore code.

Linux Windows Su Linux e Windows, è anche possibile avviare un listener utilizzando IBM MQ Explorer.

z/OS Su z/OS, è anche possibile utilizzare le operazioni e i pannelli di controllo per avviare un listener.

Il numero della porta su cui il listener è in ascolto deve corrispondere al numero di porta specificato dalla proprietà PORT della factory di connessione utilizzata dall'applicazione per connettersi al gestore code. Il valore predefinito della proprietà PORT è 1414.

IVT point-to-point per IBM MQ classes for JMS

Un programma IVT (point - to - point installation verification test) viene fornito con IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging. Il programma si connette a un gestore code in modalità bind o client, invia un messaggio alla coda denominata SYSTEM.DEFAULT.LOCAL.QUEUEe riceve il messaggio dalla coda. Il programma può creare e configurare tutti gli oggetti richiesti in modo dinamico al runtime oppure può utilizzare JNDI per richiamare gli oggetti gestiti da un servizio di directory.

Eseguire il test di verifica dell'installazione senza utilizzare prima JNDI perché il test è autonomo e non richiede l'uso di un servizio directory. Per una descrizione degli oggetti gestiti, consultare Configurazione degli oggetti JMS utilizzando lo strumento di gestione.

Il test di verifica dell'installazione point-to-point senza utilizzare JNDI

In questo test, il programma IVT crea e configura tutti gli oggetti che richiede dinamicamente al runtime e non utilizza JNDI.

Multi Su più piattaforme, è fornito uno script per eseguire il programma IVT. Lo script è denominato **IVTRun** su sistemi AIX and Linux e **IVTRun.bat** su Windows. lo script è ubicato nella directory secondaria `bin` della directory di installazione IBM MQ classes for JMS . Il percorso classi deve contenere `com.ibm.mqjms.jar`.

Per eseguire il test in modalità bind, immettere il seguente comando:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Per eseguire la verifica in modalità client, configurare prima il gestore code come descritto in “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072. Tenere presente che il canale da utilizzare assume il valore predefinito **SYSTEM.DEF.SVRCONN** e la coda da utilizzare è **SYSTEM.DEFAULT.LOCAL.QUEUE**, quindi immettere il seguente comando:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

z/OS Non viene fornito alcuno script equivalente sui sistemi z/OS . Invece, si esegue l'IVT in modalità bind richiamando direttamente la classe Java . Su z/OS, si sceglie tra due istanze funzionalmente identiche del programma IVT:

- `com.ibm.mq.jms.MQJMSIVT`, disponibile con IBM MQ classes for JMS (JMS 2.0). Per utilizzare questo programma, il percorso classi deve contenere `com.ibm.mqjms.jar` o `com.ibm.mq.allclient.jar`.
- `com.ibm.mq.jakarta.jms.MQJMSIVT`, che è disponibile con IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0). Per utilizzare questo programma, il percorso classi deve contenere `com.ibm.mq.jakarta.client.jar`.

Per eseguire il test in modalità bind su z/OS, immettere il seguente comando:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

I parametri sui comandi hanno i seguenti significati:

-m *gestore code*
Il nome del gestore code a cui si connette il programma IVT. Se si esegue la verifica in modalità bind e si omette questo parametro, il programma IVT si connette al gestore code predefinito.

-host *nome host*
Il nome host o l'indirizzo IP del sistema su cui è in esecuzione il gestore code.

-port *porta*
Il numero della porta su cui il listener del gestore code è in ascolto. Il valore predefinito è 1414.

-channel *canale*
Il nome del canale MQI che il programma IVT utilizza per connettersi al gestore code. Il valore predefinito è SYSTEM.DEF.SVRCONN.

-v *providerVersion*
Il livello di rilascio del gestore code a cui il programma IVT prevede di connettersi.

Questo parametro viene utilizzato per impostare la proprietà PROVIDERVERSION di un oggetto factory MQQueueConnectione ha gli stessi valori validi della proprietà PROVIDERVERSION. Per ulteriori informazioni su questo parametro, inclusi i valori validi, consultare JMS: changes to PROVIDERVERSION property e la descrizione della proprietà PROVIDERVERSION in Properties of IBM MQ classes for JMS objects.

Il valore predefinito è unspecified.

-ccsid *idccs*
L'identificativo (CCSID) della serie di caratteri codificati, o codepage, che deve essere utilizzata dal collegamento. Il valore predefinito è 819.

-t
La traccia è abilitata. Per impostazione predefinita, la traccia è disabilitata.

Un test riuscito produce un output simile al seguente output di esempio:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
```

```
Closing Connection
IVT completed OK
IVT finished
```

Il test di verifica dell'installazione point-to-point utilizzando JNDI

Multi

In questa verifica, il programma IVT utilizza JNDI per richiamare gli oggetti gestiti da un servizio directory.

Prima di poter eseguire il test, è necessario configurare un servizio di directory basato su un server LDAP (Lightweight Directory Access Protocol) o sul file system locale. È inoltre necessario configurare lo strumento di gestione IBM MQ JMS in modo che possa utilizzare il servizio directory per memorizzare gli oggetti gestiti. Per ulteriori informazioni su questi prerequisiti, consultare “Prerequisiti per IBM MQ classes for JMS” a pagina 90. Per informazioni sulla configurazione dello strumento di amministrazione IBM MQ JMS , consultare Configurazione dello strumento di amministrazione JMS.

Il programma IVT deve essere in grado di utilizzare JNDI per richiamare un oggetto factory MQQueueConnectione un oggetto MQQueue dal servizio directory. Viene fornito uno script per creare questi oggetti gestiti. Lo script è denominato IVTSetup su AIX and Linux e IVTSetup.bat su Windowse si trova nella sottodirectory `bin` della directory di installazione IBM MQ classes for JMS . Per eseguire lo script, immettere il comando seguente:

```
IVTSetup
```

Lo script richiama lo strumento di gestione IBM MQ JMS per creare gli oggetti gestiti.

L'oggetto factory MQQueueConnectionè collegato con il nome ivtQCF e viene creato con i valori predefiniti per tutte le proprietà, il che significa che il programma IVT viene eseguito in modalità bind e si connette al gestore code predefinito. Se si desidera che il programma IVT venga eseguito in modalità client o connettersi a un gestore code diverso dal gestore code predefinito, è necessario utilizzare lo strumento di amministrazione IBM MQ JMS o IBM MQ Explorer per modificare le proprietà appropriate dell'oggetto factory MQQueueConnection. Per informazioni su come utilizzare lo strumento di amministrazione IBM MQ Explorer JMS , vedere Configurazione degli oggetti JMS utilizzando lo strumento di amministrazione. Per informazioni su come utilizzare IBM MQ Explorer, consultare Introduzione a IBM MQ Explorer o la guida fornita con IBM MQ Explorer.

L'oggetto MQQueue è collegato al nome ivtQ e viene creato con i valori predefiniti per tutte le proprie proprietà, tranne per la proprietà QUEUE, che ha il valore SYSTEM.DEFAULT.LOCAL.QUEUE.

Una volta creati gli oggetti gestiti, è possibile eseguire il programma IVT. Per eseguire il test utilizzando JNDI, immettere il seguente comando:

```
IVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

I parametri sul comando hanno i seguenti significati:

-url "*providerURL*"
L'URL (Uniform Resource Locator) (URL) del servizio directory. L' URL può avere uno dei formati seguenti:

- `ldap://hostname/contextName` , per un servizio di directory basato su un server LDAP
- `file:/directoryPath` , per un servizio di directory basato sul file system locale

È necessario racchiudere l' URL tra virgolette (").

-icf *initCtxFact*
Il nome classe del factory di contesto iniziale, che deve essere uno dei seguenti valori:

- `com.sun.jndi.ldap.LdapCtxFactory`, per un servizio di directory basato sul server LDAP. Questo è il valore predefinito.

- `com.sun.jndi.fscontext.RefFSContextFactory`, per un servizio directory basato sul file system locale.

-t
La traccia è abilitata. Per impostazione predefinita, la traccia è disabilitata.

Un test riuscito produce un output simile a quello per un test riuscito senza utilizzare JNDI. La differenza principale è che l'output indica che la verifica sta utilizzando JNDI per richiamare un oggetto factory MQQueueConnectione un oggetto MQQueue.

Sebbene non sia strettamente necessario, si consiglia di riordinare dopo il test eliminando gli oggetti gestiti creati dallo script IVTSetup. Per questo scopo viene fornito uno script. Lo script è denominato IVTTidy sui sistemi AIX and Linux e IVTTidy.bat su Windowse si trova nella sottodirectory `bin` della directory di installazione di IBM MQ classes for JMS .

Determinazione dei problemi per il test di verifica dell'installazione point - to - point

Il test di verifica dell'installazione potrebbe avere esito negativo per i seguenti motivi:

- Se il programma IVT scrive un messaggio che indica che non è possibile trovare una classe, verificare che il percorso di classe sia impostato correttamente, come descritto in “Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging” a pagina 95.
- Il test potrebbe avere esito negativo con il seguente messaggio:

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

  e un codice di errore associato 2059. Le variabili nel messaggio hanno i seguenti significati:

  qmgr
  Il nome del gestore code a cui il programma IVT sta tentando di connettersi. Questo inserimento del messaggio è vuoto se il programma IVT sta tentando di connettersi al gestore code predefinito in modalità bind.

  connMode
  La modalità di connessione, che è `Bindings` o `Client`.

  nomehost
  Il nome host o l'indirizzo IP del sistema su cui è in esecuzione il gestore code.

  Questo messaggio indica che il gestore code a cui il programma IVT sta tentando di connettersi non è disponibile. Verificare che il gestore code sia in esecuzione e, se il programma IVT sta tentando di connettersi al gestore code predefinito, accertarsi che il gestore code sia definito come gestore code predefinito per il sistema.
- Il test potrebbe avere esito negativo con il seguente messaggio:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

  Questo messaggio indica che la coda SYSTEM.DEFAULT.LOCAL.QUEUE non esiste sul gestore code a cui è connesso il programma IVT. In alternativa, se la coda esiste, il programma IVT non può aprire la coda poiché non è abilitata per l'inserimento e il richiamo dei messaggi. Verificare che la coda esista e che sia abilitata per l'inserimento e il richiamo dei messaggi.
- Il test potrebbe avere esito negativo con il seguente messaggio:

```
Unable to bind to object
```

  Questo messaggio indica che esiste una connessione con il server LDAP, ma che il server non è configurato correttamente. Il server LDAP non è configurato per memorizzare gli oggetti Java oppure le

autorizzazioni sugli oggetti o il suffisso non sono corrette. Per ulteriore assistenza in questa situazione, consultare la documentazione per il proprio server LDAP.

- Il test potrebbe avere esito negativo con il seguente messaggio:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

Questo messaggio indica che il gestore code non è configurato correttamente per accettare una connessione client dal sistema. Consultare “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072 per i dettagli.

L'IVT di pubblicazione / sottoscrizione per IBM MQ classes for JMS

Un programma IVT (publish/subscribe installation verification test) viene fornito con IBM MQ classes for JMS. Il programma si connette a un gestore code in modalità bind o client, effettua la sottoscrizione a un argomento, pubblica un messaggio sull'argomento e riceve il messaggio appena pubblicato. Il programma può creare e configurare tutti gli oggetti richiesti in modo dinamico al runtime oppure può utilizzare JNDI per richiamare gli oggetti gestiti da un servizio di directory.

Eseguire il test di verifica dell'installazione senza utilizzare prima JNDI perché il test è autonomo e non richiede l'uso di un servizio directory. Per una descrizione degli oggetti gestiti, consultare Configurazione degli oggetti JMS utilizzando lo strumento di gestione.

Il test di verifica dell'installazione di pubblicazione / sottoscrizione senza utilizzare JNDI

In questo test, il programma IVT crea e configura tutti gli oggetti che richiede dinamicamente al runtime e non utilizza JNDI.

Viene fornito uno script per eseguire il programma IVT. Lo script è denominato PSIVTRun su sistemi AIX and Linux e PSIVTRun.bat su Windowse si trova nella sottodirectory `bin` della directory di installazione IBM MQ classes for JMS .

Per eseguire il test in modalità bind, immettere il seguente comando:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

Per eseguire la verifica in modalit ... client, configurare prima il gestore code come descritto in “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072 , osservando che il canale da utilizzare assume il valore predefinito SYSTEM.DEF.SVRCONN, quindi immettere il seguente comando:

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

I parametri sui comandi hanno i seguenti significati:

-m *gestore code*
Il nome del gestore code a cui si connette il programma IVT. Se si esegue la verifica in modalità bind e si omette questo parametro, il programma IVT si connette al gestore code predefinito.

-host *nome host*
Il nome host o l'indirizzo IP del sistema su cui è in esecuzione il gestore code.

-port *porta*
Il numero della porta su cui il listener del gestore code è in ascolto. Il valore predefinito è 1414.

-channel *canale*
Il nome del canale MQI che il programma IVT utilizza per connettersi al gestore code. Il valore predefinito è `SYSTEM.DEF.SVRCONN`.

-bqm *brokerQmgr*
Il nome del gestore code su cui è in esecuzione il broker. Il valore predefinito è il nome del gestore code a cui si connette il programma IVT.

Questo parametro non è rilevante per il numero di versione del gestore code v uguale o superiore a 7.

-v *providerVersion*
Il livello di rilascio del gestore code a cui il programma IVT prevede di connettersi.

Questo parametro viene utilizzato per impostare la proprietà PROVIDERVERSION di un oggetto factory MQTopicConnectione ha gli stessi valori validi della proprietà PROVIDERVERSION. Per ulteriori informazioni su questo parametro, inclusi i relativi valori validi, consultare la descrizione della proprietà PROVIDERVERSION in Proprietà degli oggetti IBM MQ classes for JMS.

Il valore predefinito è `unspecified`.

-ccsid *idccs*
L'identificativo (CCSID) della serie di caratteri codificati, o codepage, che deve essere utilizzata dal collegamento. Il valore predefinito è 819.

-t
La traccia è abilitata. Per impostazione predefinita, la traccia è disabilitata.

Un test riuscito produce un output simile al seguente output di esempio:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

Il test di verifica dell'installazione di pubblicazione / sottoscrizione utilizzando JNDI

In questa verifica, il programma IVT utilizza JNDI per richiamare gli oggetti gestiti da un servizio directory.

Prima di poter eseguire il test, è necessario configurare un servizio di directory basato su un server LDAP (Lightweight Directory Access Protocol) o sul file system locale. È inoltre necessario configurare lo strumento di gestione IBM MQ JMS in modo che possa utilizzare il servizio directory per memorizzare gli oggetti gestiti. Per ulteriori informazioni su questi prerequisiti, consultare “Prerequisiti per IBM MQ classes for JMS” a pagina 90. Per informazioni sulla configurazione dello strumento di amministrazione IBM MQ JMS , consultare Configurazione dello strumento di amministrazione JMS.

Il programma IVT deve essere in grado di utilizzare JNDI per richiamare un oggetto factory MQTopicConnectione un oggetto MQTopic dal servizio directory. Viene fornito uno script per creare questi oggetti gestiti. Lo script è denominato IVTSetup su AIX and Linux e IVTSetup.bat su Windowse si trova nella sottodirectory `bin` della directory di installazione IBM MQ classes for JMS . Per eseguire lo script, immettere il comando seguente:

```
IVTSetup
```

Lo script richiama lo strumento di gestione IBM MQ JMS per creare gli oggetti gestiti.

L'oggetto factory MQTopicConnectionè collegato con nome ivtTCF e viene creato con i valori predefiniti per tutte le relative proprietà, il che significa che il programma IVT viene eseguito in modalità bind, si connette al gestore code predefinito e utilizza la funzione di pubblicazione / sottoscrizione integrata. Se si desidera che il programma IVT venga eseguito in modalità client, connettersi a un gestore code diverso dal gestore code predefinito o utilizzare IBM Integration Bus invece della funzione di pubblicazione / sottoscrizione integrata, è necessario utilizzare lo strumento di gestione IBM MQ JMS o IBM MQ Explorer per modificare le proprietà appropriate dell'oggetto factory MQTopicConnection. Per informazioni su come utilizzare lo strumento di amministrazione IBM MQ JMS , vedere Configurazione degli oggetti JMS utilizzando lo strumento di amministrazione. Per informazioni su come utilizzare Esplora risorse di IBM MQ , consultare la guida fornita con Esplora risorse IBM MQ .

L'oggetto MQTopic è collegato con il nome ivtT ed è creato con i valori predefiniti per tutte le relative proprietà, ad eccezione della proprietà TOPIC, che ha il valore MQJMS/PSIVT/Information.

Una volta creati gli oggetti gestiti, è possibile eseguire il programma IVT. Per eseguire il test utilizzando JNDI, immettere il seguente comando:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

I parametri sul comando hanno i seguenti significati:

-url "*providerURL*"
L'URL (Uniform Resource Locator) (URL) del servizio directory. L' URL può avere uno dei formati seguenti:

- `ldap://hostname/contextName` , per un servizio di directory basato su un server LDAP
- `file:/directoryPath` , per un servizio di directory basato sul file system locale

È necessario racchiudere l' URL tra virgolette (").

-icf *initCtxFact*
Il nome classe del factory di contesto iniziale, che deve essere uno dei seguenti valori:

- `com.sun.jndi.ldap.LdapCtxFactory`, per un servizio di directory basato sul server LDAP. Questo è il valore predefinito.
- `com.sun.jndi.fscontext.RefFSContextFactory`, per un servizio directory basato sul file system locale.

-t
La traccia è abilitata. Per impostazione predefinita, la traccia è disabilitata.

Un test riuscito produce un output simile a quello per un test riuscito senza utilizzare JNDI. La differenza principale è che l'output indica che la verifica sta utilizzando JNDI per richiamare un oggetto factory MQTopicConnectione un oggetto MQTopic.

Sebbene non sia strettamente necessario, si consiglia di riordinare dopo il test eliminando gli oggetti gestiti creati dallo script IVTSetup. Per questo scopo viene fornito uno script. Lo script è denominato IVTTidy sui sistemi AIX and Linux e IVTTidy.bat su Windowse si trova nella sottodirectory `bin` della directory di installazione di IBM MQ classes for JMS .

Determinazione dei problemi per il test di verifica dell'installazione di pubblicazione / sottoscrizione

Il test di verifica dell'installazione potrebbe avere esito negativo per i seguenti motivi:

- Se il programma IVT scrive un messaggio che indica che non è possibile trovare una classe, verificare che il percorso di classe sia impostato correttamente, come descritto in "Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging" a pagina 95.
- Il test potrebbe avere esito negativo con il seguente messaggio:

  ```
  Failed to connect to queue manager ' qmgr ' with
  connection mode ' connMode ' and host name ' hostname '
  ```

  e un codice di errore associato 2059. Le variabili nel messaggio hanno i seguenti significati:

  qmgr
  Il nome del gestore code a cui il programma IVT sta tentando di connettersi. Questo inserimento del messaggio è vuoto se il programma IVT sta tentando di connettersi al gestore code predefinito in modalità bind.

  connMode
  La modalità di connessione, che è `Bindings` o `Client`.

  nomehost
  Il nome host o l'indirizzo IP del sistema su cui è in esecuzione il gestore code.

  Questo messaggio indica che il gestore code a cui il programma IVT sta tentando di connettersi non è disponibile. Verificare che il gestore code sia in esecuzione e, se il programma IVT sta tentando di connettersi al gestore code predefinito, accertarsi che il gestore code sia definito come gestore code predefinito per il sistema.
- Il test potrebbe avere esito negativo con il seguente messaggio:

  ```
  Unable to bind to object
  ```

  Questo messaggio indica che esiste una connessione con il server LDAP, ma che il server non è configurato correttamente. Il server LDAP non è configurato per memorizzare gli oggetti Java oppure le autorizzazioni sugli oggetti o il suffisso non sono corrette. Per ulteriore assistenza in questa situazione, consultare la documentazione per il proprio server LDAP.
- Il test potrebbe avere esito negativo con il seguente messaggio:

  ```
  The security authentication was not valid that was supplied for
  QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
  ```

  Questo messaggio indica che il gestore code non è configurato correttamente per accettare una connessione client dal sistema. Per ulteriori informazioni, consultare "Configurazione di un gestore code per accettare connessioni client su Multiplatforms" a pagina 1072.

JMS 2.0 *Utilizzo di applicazioni di esempio IBM MQ classes for JMS*

Le applicazioni di esempio IBM MQ classes for JMS forniscono una panoramica delle funzioni comuni dell'API JMS . È possibile utilizzarle per verificare l'installazione e la configurazione del server di messaggistica e per creare le proprie applicazioni.

Informazioni su questa attività

Se si ha bisogno di aiuto per creare le proprie applicazioni, è possibile utilizzare le applicazioni di esempio come punto di partenza. Sia l'origine che una versione compilata vengono fornite per ciascuna applicazione. Esaminare il codice di origine di esempio e identificare i passi chiave per creare ciascun oggetto richiesto per l'applicazione (ConnectionFactory, Connection, Session, Destination e un Producer o un Consumer o entrambi) e per impostare le proprietà specifiche necessarie per specificare il funzionamento dell'applicazione. Per ulteriori informazioni, consultare “Scrittura di applicazioni IBM MQ classes for JMS/Jakarta Messaging” a pagina 142. Gli esempi potrebbero essere soggetti a modifiche nelle release future di IBM MQ.

Per JMS 2.0, Tabella 11 a pagina 122 mostra dove sono installate le applicazioni di esempio IBM MQ classes for JMS su ciascuna piattaforma.

Nota:

JM 3.0 Per IBM MQ classes for Jakarta Messaging, si stanno preparando nuovi esempi.

Tabella 11. Directory di installazione per le applicazioni di esempio IBM MQ classes for JMS

Piattaforma	**Directory**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/jms/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\jms\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/jms/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/jms`

All'interno di questa directory, ci sono sottodirectory che contengono una o più applicazioni di esempio come mostrato in Tabella 12 a pagina 122.

Tabella 12. IBM MQ classes for JMSApplicazioni di esempio

Nome del campione	**Descrizione**
JmsBrowser.java	Un'applicazione browser della coda JMS che esamina tutti i messaggi disponibili nella coda denominata, senza rimuoverli, nell'ordine in cui vengono ricevuti da un'applicazione consumer.
JmsConsumer.java	Un'applicazione browser della coda JMS che esamina tutti i messaggi disponibili sulla coda denominata, senza rimuoverli, nell'ordine in cui vengono ricevuti da un'applicazione consumer, ricercando l'istanza della factory di connessione e l'istanza di destinazione in un contesto iniziale (questo esempio supporta solo il contesto del file system).
JmsJndiConsumer.java	Un'applicazione consumer JMS (ricevente o sottoscrittore) che riceve un messaggio dalla destinazione denominata (coda o argomento) ricercando l'istanza del factory di connessione e l'istanza di destinazione in un contesto iniziale (questo esempio supporta solo il contesto del file system).

Tabella 12. IBM MQ classes for JMSApplicazioni di esempio (Continua)	
Nome del campione	**Descrizione**
JmsJndiProducer.java	Un'applicazione del produttore JMS (mittente o publisher) che invia un messaggio semplice alla destinazione denominata (coda o argomento) ricercando l'istanza del factory di connessione e l'istanza di destinazione in un contesto iniziale (questo esempio supporta solo il contesto del file system).
JmsProducer.java	Un'applicazione del produttore JMS (mittente o publisher) che invia un messaggio semplice alla destinazione denominata (coda o argomento).
/interattivo/	
SampleConsumerJava.java	Ricevere messaggi da un argomento / coda.
SampleProducerJava.java	Inviare messaggi a un argomento / coda.
/interattivo/helper	
BaseOptions.java	Una classe astratta che può essere estesa per fornire funzionalità di opzioni utente.
IsValidType.java	Classe astratta per le classi di controllo validità.
JmsApp.java	Una classe astratta che può essere estesa per fornire la funzionalità consumer / producer.
Keys.java	Una serie di chiavi che definiscono opzioni per applicazioni di esempio.
Literals.java	Una serie di costanti letterali.
MyContext.java	Il contesto in cui vengono presentate le opzioni.
Options.java	Fornisce funzionalità per le opzioni utente.
OptionsPresenter.java	Contesto in cui vengono presentate le opzioni correnti.
/semplice/	
SimpleAsyncPutPTP.java	Una semplice applicazione per la messaggistica punto a punto; il messaggio viene inviato in modo asincrono (noto anche come messaggistica *fire - and - forget*). Non viene ricevuto alcun messaggio.
SimpleDurableSub.java	Un'applicazione semplice che dimostra la funzione di sottoscrizione durevole.
SimpleJNDILookup.java	Un'applicazione semplice e minima che dimostra la ricerca di oggetti JMS utilizzando il contesto iniziale. Non viene effettuata alcuna connessione al gestore code e non viene inviato o ricevuto alcun messaggio.
SimpleMQMDRead.java	Un'applicazione semplice che dimostra in che modo un'applicazione JMS può avvalersi dei campi MQMD (MQ Message Descriptor) come proprietà del messaggio JMS . Non viene inviato alcun messaggio; si presume che la coda in uso sia popolata con alcuni messaggi.
SimpleMQMDWrite.java	Un'applicazione semplice che dimostra in che modo un'applicazione JMS può scrivere campi MQMD (MQ Message Descriptor). Non viene ricevuto alcun messaggio.

Tabella 12. IBM MQ classes for JMSApplicazioni di esempio (Continua)	
Nome del campione	**Descrizione**
SimplePTP.java	Un'applicazione semplice e minima per la messaggistica point - to - point.
SimplePubSub.java	Un'applicazione semplice e minima per la messaggistica di pubblicazione - sottoscrizione.
SimpleReadAheadPTP.java	Una semplice applicazione per la messaggistica punto a punto; i messaggi vengono trasmessi dal gestore code (noto anche come funzione di lettura anticipata). Non viene inviato alcun messaggio; si presume che la coda in uso sia popolata con alcuni messaggi.
SimpleRequestor.java	Un'applicazione semplice che utilizza un richiedente per inviare un messaggio di richiesta e quindi attendere e ricevere la risposta. Nota: si presume che qualche altra applicazione elaborerà il messaggio di richiesta e invierà il messaggio di risposta.
SimpleResponder.java	Una semplice applicazione che ascolta una destinazione per un messaggio e quindi invia una risposta alla destinazione replyTo del messaggio. L'applicazione viene scritta per funzionare insieme all'esempio SimpleRequestor .
SimpleRetainedPub.java	Un'applicazione semplice che dimostra una pubblicazione conservata. Non viene ricevuto alcun messaggio.
SimpleWMQJMSPTP.java	Un'applicazione semplice e minima per la messaggistica point - to - point.
SimpleWMQJMSPubSub.java	Un'applicazione semplice e minima per la messaggistica di pubblicazione / sottoscrizione.

Il IBM MQ classes for JMS fornisce uno script denominato `runjms` che può essere utilizzato per eseguire le applicazioni di esempio. Questo script imposta l'ambiente IBM MQ per consentire l'esecuzione delle applicazioni di esempio IBM MQ classes for JMS .

Tabella 13 a pagina 124 mostra l'ubicazione dello script su ciascuna piattaforma:

Tabella 13. Ubicazione dello script `runjms`	
Piattaforma	**Directory**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	o
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

Per utilizzare uno script `runjms` per richiamare un'applicazione di esempio, completare la seguente procedura:

Procedura

1. Visualizzare un prompt dei comandi e passare alla directory contenente l'applicazione di esempio che si desidera eseguire.
2. Immettere il seguente comando:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   L'applicazione di esempio visualizza un elenco di parametri necessari.
3. Immettere il seguente comando per eseguire l'esempio con questi parametri:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Esempio

Linux Ad esempio, per eseguire l'esempio `JmsBrowser` su Linux, immettere i seguenti comandi:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

Concetti correlati

“Cosa è installato per IBM MQ classes for JMS” a pagina 91
Una serie di file e directory vengono creati quando si installa IBM MQ classes for JMS. Su Windows, alcune configurazioni vengono eseguite durante l'installazione impostando automaticamente le variabili di ambiente. Su altre piattaforme e in determinati ambienti Windows , è necessario impostare le variabili di ambiente prima di eseguire applicazioni IBM MQ classes for JMS .

Script forniti con IBM MQ classes for JMS/Jakarta Messaging

Viene fornito un certo numero di script per assistere le attività comuni che devono essere eseguite quando si utilizzano IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Tabella 14 a pagina 125 elenca tutti gli script e i loro utilizzi. Gli script si trovano nella sottodirectory `bin` della directory di installazione di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

Tabella 14. Script forniti con IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Utilità	**Utilizzo**
Ripulitura [1]	Questo script viene mantenuto per la compatibilit ... con le release precedenti ma non esegue alcuna funzione. La ripulitura manuale delle informazioni di sottoscrizione non è più necessaria.
DefaultConfiguration	Esegue l'applicazione di configurazione predefinita su piattaforme diverse da Windows.
formatLog [1]	Questo script viene mantenuto per la compatibilit ... con le release precedenti ma non esegue alcuna funzione. L'output del log viene ora prodotto in testo leggibile.
IVTRUN [1] Configurazione IVT [1] IVTTidy [1]	Utilizzato nel test di verifica dell'installazione point-to-point, come descritto in “IVT point-to-point per IBM MQ classes for JMS” a pagina 114.
JM 3.0 JMS30Admin [1]	Esegue lo strumento di amministrazione IBM MQ Jakarta Messaging , come descritto in Avvio dello strumento di amministrazione.
JM 3.0 JMS30Admin.config	Il file di configurazione per lo strumento di amministrazione IBM MQ Jakarta Messaging , come descritto in Configurazione dello strumento di amministrazione JMS.

Tabella 14. Script forniti con IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging (Continua)	
Utilità	**Utilizzo**
JMS 2.0 Admin JMS [1]	Esegue lo strumento di amministrazione IBM MQ JMS , come descritto in Avvio dello strumento di amministrazione.
JMS 2.0 JMSAdmin.config	Il file di configurazione per lo strumento di amministrazione IBM MQ JMS , come descritto in Configurazione dello strumento di amministrazione JMS.
PSIVTRun [1]	Esegue il programma di verifica dell'installazione di pubblicazione / sottoscrizione, come descritto in “L'IVT di pubblicazione / sottoscrizione per IBM MQ classes for JMS” a pagina 118.
PSReportDump.class	Questa classe viene mantenuta per la compatibilità con le release precedenti, ma non esegue alcuna funzione.
V 9.3.0 JM 3.0 V 9.3.0 setjms30env “2” a pagina 126	Linux AIX Per Jakarta Messaging 3.0, imposta le variabili di ambiente per l'esecuzione di un'applicazione IBM MQ classes for JMS in una JVM (Java virtual machine) a 32 bit su sistemi AIX and Linux , come descritto in “Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging” a pagina 95.
JMS 2.0 setjmsenv “2” a pagina 126	Linux AIX Per JMS 2.0, imposta le variabili di ambiente per l'esecuzione di un'applicazione IBM MQ classes for JMS in una JVM (Java virtual machine) a 32 bit su sistemi AIX and Linux , come descritto in “Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging” a pagina 95.
V 9.3.0 JM 3.0 V 9.3.0 setjms30env64 “2” a pagina 126	Linux AIX Per Jakarta Messaging 3.0, imposta le variabili di ambiente per l'esecuzione di un'applicazione IBM MQ classes for JMS in un JVM a 64 bit su sistemi AIX and Linux , come descritto in “Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging” a pagina 95.
JMS 2.0 setjmsenv64 “2” a pagina 126	Linux AIX Per JMS 2.0, imposta le variabili di ambiente per l'esecuzione di un'applicazione IBM MQ classes for JMS in un JVM a 64 bit su sistemi AIX and Linux , come descritto in “Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging” a pagina 95.

Nota:

1. Su Windows, il nome file ha l'estensione `.bat`.
2. Questi script sono disponibili solo su AIX and Linux . Su Windows, dopo l'installazione di IBM MQ, eseguire il comando **setmqenv**. Per ulteriori informazioni, consultare “Impostazione delle variabili di ambiente per IBM MQ classes for JMS/Jakarta Messaging” a pagina 95.

Supporto per OSGi con IBM MQ classes for JMS

OSGi fornisce un framework che supporta la distribuzione delle applicazioni come bundle. I bundle OSGi vengono forniti come parte di IBM MQ classes for JMS.

IBM MQ classes for JMS include i seguenti bundle OSGi.

com.ibm.msg.client.osgi.jms*version_number*.jar
Il livello comune di codice in IBM MQ classes for JMS. Per informazioni sull'architettura a livelli delle classi IBM MQ per JMS, consultare IBM MQ classes for JMS architecture.

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
I file JAR (Java archive) prerequisiti per il livello comune.

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Servizi comuni per applicazioni Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_*version_number*.jar
Messaggi per il livello comune.

com.ibm.msg.client.osgi.wmq_*version_number*.jar
Il provider di messaggistica IBM MQ in IBM MQ classes for JMS. Per informazioni sull'architettura a livelli di IBM MQ classes for JMS, consultare IBM MQ classes per l'architettura JMS.

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
I file JAR prerequisiti per il provider di messaggistica IBM MQ .

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
Messaggi per il provider di messaggistica IBM MQ .

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, questo file JAR consente alle applicazioni di utilizzare sia IBM MQ classes for JMS che IBM MQ classes for Javae include anche il codice per gestire i messaggi PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, questo file JAR fornisce i prerequisiti per `com.ibm.mq.jakarta.osgi.allclient_`*`version_number`*`.jar`.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Per JMS 2.0, questo file JAR consente alle applicazioni di utilizzare sia IBM MQ classes for JMS che IBM MQ classes for Javae include anche il codice per gestire i messaggi PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Per JMS 2.0, questo file JAR fornisce i prerequisiti per `com.ibm.mq.osgi.allclient_`*`version_number`*`.jar`.

dove *`version_number`* è il numero di versione di IBM MQ installato.

I bundle sono installati nella sottodirectory `java/lib/OSGi` dell'installazione di IBM MQ o nella cartella `java\lib\OSGi` su Windows.

Da IBM MQ 8.0, utilizzare i bundle `com.ibm.mq.osgi.allclient_8.0.0.0.jare` `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` per le nuove applicazioni. L'uso di questi bundle rimuove la limitazione di non essere in grado di eseguire sia IBM MQ classes for JMS che IBM MQ classes for Java nello stesso framework OSGi, tuttavia, tutte le altre restrizioni si applicano ancora.

Il bundle `com.ibm.mq.osgi.java`*`version_number`*`.jar`, che è anche installato nella sottodirectory `java/lib/OSGi` dell'installazione di IBM MQ , o la cartella `java\lib\OSGi` su Windows, fa parte di IBM MQ classes for Java. Questo bundle non deve essere caricato in un ambiente di runtime OSGi su cui è caricato IBM MQ classes for JMS .

I bundle OSGi per IBM MQ classes for JMS sono stati scritti nella specifica OSGi Release 4. Non funzionano in un ambiente OSGi Release 3.

È necessario impostare correttamente il percorso di sistema o il percorso della libreria in modo che l'ambiente di runtime OSGi possa trovare i file DLL o le librerie condivise richiesti.

Se si utilizzano i bundle OSGi per IBM MQ classes for JMS, gli argomenti temporanei non funzionano. Inoltre, le classi di uscita del canale scritte in Java non sono supportate a causa di un problema inerente al caricamento delle classi in un ambiente con più programmi di caricamento classi come OSGi. Un bundle utente può essere a conoscenza dei bundle IBM MQ classes for JMS , ma i bundle IBM MQ classes for JMS non sono a conoscenza di alcun bundle utente. Di conseguenza, il programma di caricamento classi utilizzato in un bundle IBM MQ classes for JMS non può caricare una classe di uscita del canale che si trova in un bundle utente.

Per ulteriori informazioni su OSGi, consultare il sito Web OSGi Alliance .

z/OS MQ Adv. VUE JMS/Jakarta Messaging connettività client alle applicazioni batch in esecuzione su z/OS

In determinate condizioni, un'applicazione IBM MQ classes for JMS/Jakarta Messaging su z/OS può connettersi a un gestore code su z/OS utilizzando una connessione client. L'utilizzo di una connessione client può semplificare le topologie IBM MQ .

Utilizzando una connessione client, se un'applicazione IBM MQ classes for JMS/Jakarta Messaging viene eseguita in un ambiente batch e si applica una delle condizioni riportate di seguito, l'applicazione può connettersi a un gestore code z/OS remoto:

- V 9.3.4 LTS Il codice IBM MQ classes for JMS/Jakarta Messaging è all'indirizzo IBM MQ 9.3.4 o successivo oppure Long Term Support con l'APAR PH56722 applicato. Il gestore code può essere di qualsiasi versione supportata.
- Il gestore code a cui ci si connette è in esecuzione con la titolarità IBM MQ Advanced for z/OS Value Unit Edition e pertanto il parametro **ADVCAP** è impostato su ENABLED. Il gestore code può essere di qualsiasi versione supportata.

Per ulteriori informazioni su IBM MQ Advanced for z/OS Value Unit Edition , consultare IBM MQ product identifier and export information.

Consultare DISPLAY QMGR per ulteriori informazioni su **ADVCAP** e START QMGR per ulteriori informazioni su **QMGRPROD**.

Si noti che il batch è l'unico ambiente supportato; non esiste alcun supporto per JMS/Jakarta Messaging per CICS o JMS/Jakarta Messaging per IMS.

Un'applicazione IBM MQ classes for JMS/Jakarta Messaging su z/OS non può utilizzare una connessione in modalità client per connettersi a un gestore code che non è in esecuzione su z/OS.

Se un'applicazione IBM MQ classes for JMS/Jakarta Messaging su z/OS tenta di connettersi utilizzando la modalità client e non è consentita, viene emesso il messaggio di eccezione JMSFMQ0005 .

Supporto Advanced Message Security (AMS)

Le applicazioni client IBM MQ classes for JMS/Jakarta Messaging possono utilizzare AMS durante la connessione ai gestori code z/OS remoti, in base alle condizioni precedentemente descritte in questa sezione.

Per utilizzare AMS in questo modo, le applicazioni client devono utilizzare un tipo di keystore `jceracfks` in `keystore.conf`, dove:

- Il prefisso del nome della proprietà è `jceracfks` e questo prefisso del nome non è sensibile al maiuscolo / minuscolo.
- Il keystore è un keyring RACF .
- Le password non sono richieste e verranno ignorate. Questo perché i keyring RACF non utilizzano le password.
- Se si specifica il fornitore, il fornitore deve essere `IBMJCE`.

Quando si utilizza `jceracfks` con AMS, il keystore deve essere nel formato: `safkeyring://`*`user`*`/`*`keyring`*, dove:

- `safkeyring` è un valore letterale e questo nome non è sensibile al maiuscolo / minuscolo
- *`user`* è l'ID utente RACF che possiede il keyring
- *`keyring`* è il nome del keyring RACF e il nome del keyring è sensibile al maiuscolo / minuscolo

Il seguente esempio utilizza il keyring standard AMS per l'utente `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Concetti correlati

“Java connettività client alle applicazioni batch in esecuzione su z/OS” a pagina 373
In determinate condizioni, un'applicazione di IBM MQ classes for Java su z/OS può connettersi a un gestore code su z/OS utilizzando una connessione client. L'utilizzo di una connessione client può semplificare le topologie IBM MQ .

Ottenimento di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging separatamente

Le librerie e i programmi di utilità necessari per l'esecuzione delle applicazioni che utilizzano IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging sono disponibili in un file JAR autoestraente scaricato da Fix Central. Eseguire questa operazione se si desidera ottenere solo questi file, ad esempio per la distribuzione in uno strumento di gestione software o per l'utilizzo con applicazioni client autonome.

Prima di iniziare

Prima di avviare questa attività, assicurarsi di avere un JRE (Java runtime environment) installato sulla macchina e che il JRE sia stato aggiunto al percorso di sistema.

Il programma di installazione Java utilizzato in questo processo di installazione non richiede l'esecuzione come root o come utente specifico. L'unico requisito è che l'utente con cui viene eseguito disponga dell'accesso in scrittura alla directory in cui si desidera inserire i file.

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, consultare Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging.

Informazioni su questa attività

Un file JAR autoestraente viene utilizzato per ridurre la dimensione del download e il tempo impiegato per eseguire l'estrazione. Il contenuto esatto di questo file JAR e le sottodirectory in cui estrae i file dipendono dalla versione di IBM MQ.

Quando si esegue il file JAR autoestraente, viene visualizzato l'accordo di licenza IBM MQ , che deve essere accettato. Consente inoltre di modificare la directory principale per l'estrazione.

Per IBM MQ 9.3, il file JAR autoestraente estrae i file nella seguente struttura di directory:

wmq/JavaEE
I file EAR e RAR dell'adattatore risorse IBM MQ .

JMS 2.0 I seguenti file sono da utilizzare con gli oggetti JMS 2.0 e JMS 1.1 :

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 Esiste anche una serie equivalente da utilizzare con gli oggetti Jakarta Messaging 3.0 :

- `wmq.jakarta.jmsra.ivt.ear`. Contiene i file di verifica dell'installazione.
- `wmq.jakarta.jmsra.rar`. Contiene i file dell'adattatore risorse.

wmq/JavaSE

wmq/JavaSE/bin
Gli strumenti **JMSAdmin** e **JMS30Admin** . Utilizzato per definire entità JNDI che rappresentano oggetti JMS o Jakarta Messaging .

JMS 2.0 I seguenti file sono da utilizzare con gli oggetti JMS 2.0 e JMS 1.1 :

- `JMSAdmin.bat`

- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 Esiste anche una serie equivalente da utilizzare con gli oggetti Jakarta Messaging 3.0 :

- `JMS30Admin.bat`. Un file utilizzato per avviare lo strumento su Windows.
- `JMS30Admin`. Uno script utilizzato per avviare lo strumento sulle piattaforme Linux e UNIX .
- `JMS30Admin.config`. Un file di configurazione di esempio per lo strumento.

Nota:

- Prima che lo strumento **JMSAdmin** fosse aggiunto al file JAR ad estrazione automatica, i file in questa directory si trovavano nella directory principale `wmq/JavaSE`.
- Un client installato utilizzando il file JAR autoestraente può utilizzare lo strumento **JMSAdmin** o **JMS30Admin** per creare oggetti gestiti di messaggistica Java all'interno di un contesto del file system (file `.bindings`). Il client può anche ricercare e utilizzare questi oggetti gestiti.
- JMS 2.0 Lo strumento **JMSAdmin** per l'utilizzo con oggetti JMS 2.0 e JMS 1.1 è stato aggiunto al file JAR autoestraente in IBM MQ 9.2.0 Fix Pack 2 e IBM MQ 9.2.2.
- JM 3.0 Lo strumento **JMS30Admin** da utilizzare con gli oggetti Jakarta Messaging 3.0 è stato aggiunto al file JAR autoestraente in IBM MQ 9.3.0.

wmq/JavaSE/lib

Advanced Message Security utilizza i seguenti pacchetti Bouncy Castle open source per supportare la sintassi dei messaggi crittografici (CMS). Consultare Supporto per JRE nonIBM con AMS.

V 9.3.5 Per Continuous Delivery da IBM MQ 9.3.5:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS Per Long Term Support e Continuous Delivery prima di IBM MQ 9.3.5:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- `bcutil-jdk15on.jar`

I seguenti file contengono le classi per il loro livello specifico JMS o Jakarta Messaging :

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 e JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

Altri file JAR prerequisiti:

- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`. Utilizzato per controllare dinamicamente la traccia per applicazioni IBM MQ classes for JMS .
- `fscontext.jar`. Richiesto se l'applicazione esegue ricerche JNDI utilizzando un contesto di file system.
- V 9.3.3 `jackson-annotations.jar`, `jackson-core.jar`, `jackson-databind.jar`: contiene classi utilizzate per eseguire le associazioni CipherSuite e CipherSpec durante la creazione di connessioni TLS sicure a un gestore code.
- JM 3.0 `jakarta.jms-api.jar`. Contiene l'interfaccia Jakarta Messaging 3.0 e le definizioni di eccezione.

- JMS 2.0 `jms.jar`. Contiene l'interfaccia JMS 2.0 e le definizioni di eccezione.
- `org.json.jar`. Contiene le classi che consentono a IBM MQ classes for JMS di interpretare i file CCDT in formato JSON.
- `providerutil.jar`. Richiesto se l'applicazione esegue ricerche JNDI utilizzando un contesto di file system.

Nota: Stabilized `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar` contengono entrambi una copia di IBM MQ classes for Java. Tuttavia, in IBM MQ 9.0, queste classi vengono dichiarate come funzionalmente stabilizzate al livello fornito in IBM MQ 8.0. Consultare Depretazioni, stabilizzazioni e rimozioni all'indirizzo IBM MQ 9.0.

wmq/OSGi

I bundle del client OSGi IBM MQ :

- JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`
- JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

dove *V.R.M.F* è il numero di versione, release, modifica e fix pack.

Procedura

1. Scaricare il file JAR del client IBM MQ Java / JMS da Fix Central.
   a) Fare clic sul seguente collegamento: IBM MQ Java / JMS client.
   b) Trovare il client per la propria versione di IBM MQ nell'elenco visualizzato di correzioni disponibili.
   Ad esempio:

   ```
   release level: 9.3.0.0-IBM-MQ-Install-Java-All
   Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
   ```

   Quindi fare clic sul nome del file client e seguire il processo di download.
2. Avviare l'estrazione dalla directory in cui è stato scaricato il file.
   Per avviare l'estrazione, immettere un comando nel formato seguente:

   ```
   java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
   ```

   dove *V.R.M.F* è il numero di versione del prodotto, ad esempio `9.3.0.0`, e *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` è il nome del file scaricato da Fix Central.
   Ad esempio, per estrarre il client JMS per la release IBM MQ 9.3.0 , utilizzare il seguente comando:

   ```
   java -jar 9.3.0.0-IBM-MQ-Install-Java-All.jar
   ```

   Nota: Per eseguire questa installazione, è necessario avere un JRE installato sulla macchina e aggiunto al percorso di sistema.

   Quando si immettono i comandi, vengono visualizzate le seguenti informazioni:

   ```
   Prima di poter utilizzare, estrarre o installare IBM MQ V9.3, è necessario accettare
   i termini di 1. IBM Accordo di licenza internazionale per la valutazione di
   Programmi 2. IBM International Program License Agreement e ulteriori
   informazioni sulla licenza. Leggere attentamente i seguenti accordi di licenza.

   L'accordo di licenza è visualizzabile separatamente utilizzando il
   Opzione --viewLicenseAgreement.

   Premere Invio per visualizzare i termini della licenza o 'x' per ignorare.
   ```
3. Rivedere e accettare i seguenti termini di licenza:

a) Per visualizzare la licenza, premere Invio.

   In alternativa, premere x per saltare la visualizzazione della licenza.

   Una volta visualizzata la licenza, o immediatamente se si preme x, viene visualizzato il seguente messaggio:

```
Ulteriori informazioni sulla licenza sono visualizzabili separatamente utilizzando il
Opzione --viewLicenseInfo.

Premere Invio per visualizzare ora ulteriori informazioni sulla licenza oppure 'x' per
ignorare.
```

b) Per visualizzare i termini di licenza aggiuntivi, premere Invio.

   In alternativa, premere x per ignorare la visualizzazione dei termini di licenza aggiuntivi.

   Una volta visualizzati i termini di licenza aggiuntivi, o immediatamente se si preme x, viene visualizzato il seguente messaggio:

```
Scegliendo l'opzione "Accetto" di seguito, si accettano i termini del
accordo di licenza e clausole nonIBM , se applicabili. Se non si
accettare, selezionare "Non accetto".

Seleziona [ 1] Accetto, o [ 2] Non accetto:
```

c) Per accettare l'accordo di licenza e continuare con la selezione della directory di installazione, selezionare 1.

   In alternativa, selezionare 2 per terminare l'installazione immediatamente.

   Se si seleziona 1, viene visualizzato un messaggio simile al seguente:

```
Immettere la directory per i file del prodotto oppure lasciare vuoto per accettare il
valore predefinito.
La directory di destinazione predefinita è H: \downloads

Directory di destinazione per i file del prodotto?
```

4. Specificare la directory principale per l'estrazione.

   L'ubicazione predefinita è la directory corrente.

   - Se si desidera estrarre i file del prodotto nell'ubicazione predefinita, premere Invio senza specificare un valore.
   - Se si desidera estrarre i file del prodotto in un'ubicazione diversa, specificare il nome della directory in cui si desidera estrarre i file, quindi premere Invio per avviare l'estrazione.

   Il nome della directory specificato non deve esistere già, altrimenti, quando si avvia l'estrazione, viene riportato un errore e non viene installato alcun file.

   Se non esiste già, la directory specificata viene creata e i file di programma vengono estratti in questa directory. Durante l'installazione, viene creata una nuova directory con il nome `wmq` all'interno della directory principale specificata.

   Tre sottodirectory, `JavaEE`, `JavaSE` e `OSGi`, vengono create nella directory `wmq` con il seguente contenuto:

   JavaEE

   `wmq.jakarta.jmsra.ivt.ear`

   `wmq.jakarta.jmsra.rar`

   `wmq.jmsra.ivt.ear`

   `wmq.jmsra.rar`

   JavaSE

   Questa directory contiene le seguenti sottodirectory e file:

   JavaSE/lib

   `bcpkix-jdk18on.jar`

   `bcpkix-jdk15on.jar`

V 9.3.5 bcprov-jdk18on.jar

LTS bcprov-jdk15on.jar

V 9.3.5 bcutil-jdk18on.jar

LTS bcutil-jdk15on.jar

JMS 2.0 com.ibm.mq.allclient.jar

JM 3.0 com.ibm.mq.jakarta.client.jar

V 9.3.3 Removed com.ibm.mq.traceControl.jar

fscontext.jar

V 9.3.3 jackson-annotations.jar

V 9.3.3 jackson-core.jar

V 9.3.3 jackson-databind.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar

JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

Una volta completata l'estrazione, viene visualizzato un messaggio di conferma come mostrato nel seguente esempio:

```
Estrazione file in H: \downloads\wmq
Tutti i file del prodotto sono stati estratti correttamente.
```

Elenco consentito in IBM MQ classes for JMS/Jakarta Messaging

Il meccanismo di serializzazione e deserializzazione degli oggetti Java è stato identificato come un potenziale rischio per la sicurezza. Allowlisting in IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging forniscono una protezione contro alcuni rischi di serializzazione.

Informazioni su questa attività

Il meccanismo di serializzazione e deserializzazione degli oggetti Java è stato identificato come un potenziale rischio per la sicurezza perché la deserializzazione crea un'istanza di oggetti Java arbitrari, dove è possibile che i dati inviati in modo dannoso causino vari problemi. Un'applicazione notevole della serializzazione è in Jakarta Messaging 3.0 e Java Message Service 2.0 ObjectMessages che utilizzano la serializzazione per incapsulare e trasferire oggetti arbitrari.

La serializzazione allowlisting è una potenziale attenuazione rispetto ad alcuni dei rischi che la serializzazione comporta. Specificando esplicitamente quali classi possono essere incapsulate ed estratte da ObjectMessages, allowlisting fornisce una protezione contro alcuni rischi di serializzazione.

Concetti correlati

“Esecuzione delle applicazioni IBM MQ classes for JMS in Java security manager” a pagina 109
IBM MQ classes for JMS può essere eseguito con Java security manager abilitato. Per eseguire correttamente le applicazioni con Java security manager abilitato, è necessario configurare Java Virtual Machine (JVM) con un file di configurazione della politica adatto.

Concetti di elenco consentiti

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, è disponibile il supporto per allowlisting delle classi nell'implementazione dell'interfaccia JMS ObjectMessage . Ciò fornisce una potenziale attenuazione di alcuni dei rischi di sicurezza potenzialmente correlati al meccanismo di serializzazione e deserializzazione degli oggetti Java .

Elenco consentito in IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Importante:

Ovunque possibile, il termine *allowlist* ha sostituito il termine *whitelist*. Per le release IBM MQ 9.0 e successive, sono inclusi alcuni nomi di proprietà di sistema Java menzionati in questo argomento. Non è necessario modificare alcuna configurazione esistente. Anche i precedenti nomi di proprietà di sistema continuano a funzionare.

IBM MQ classes for JMS (JMS 2.0) V 9.3.0 V 9.3.0 e IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) supportano allowlisting di classi nell'implementazione dell'interfaccia JMS ObjectMessage .

- JMS 2.0 Per IBM MQ classes for JMS, i relativi nomi proprietà sono **`com.ibm.mq.jms.allowlist.*`**.
- JM 3.0 Per IBM MQ classes for Jakarta Messaging, i relativi nomi proprietà sono **`com.ibm.mq.jakarta.jms.allowlist.*`**

Allowlist definisce quali classi Java possono essere serializzate con ObjectMessage.setObject() e deserializzate con ObjectMessage.getObject().

- JMS 2.0 I tentativi di serializzazione o deserializzazione di un'istanza di una classe non inclusa nella allowlist con ObjectMessage causano l'emissione di una javax.jms.MessageFormatException , con una java.io.InvalidClassException come causa.
- JM 3.0 I tentativi di serializzazione o deserializzazione di un'istanza di una classe non inclusa nell'allowlist con ObjectMessage causano la generazione di una jakarta.jms.MessageFormatException , con una java.io.InvalidClassException come causa.

Produzione della allowlist

Importante: IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging non possono essere distribuiti con un allowlist. La scelta delle classi da trasferire utilizzando ObjectMessages è una scelta di progettazione dell'applicazione e IBM MQ non può precedere tale scelta.

Per questo motivo, il meccanismo di elenco consentito consente due modalità di funzionamento:

RILEVAMENTO
: In questa modalità, il meccanismo produce un elenco di nomi classe completi, riportando tutte le classi che sono state serializzate o deserializzate in ObjectMessages.

Implementazione
: In questa modalità, il meccanismo applica allowlisting, rifiutando i tentativi di serializzazione o deserializzazione delle classi che non sono nell'allowlist.

Se si desidera utilizzare questo meccanismo, è necessario eseguire inizialmente in modalità DISCOVERY per raccogliere l'elenco delle classi attualmente serializzate e deserializzate, rivedere l'elenco e utilizzarlo

come base per l'elenco consentito. Potrebbe anche essere appropriato utilizzare l'elenco invariato, ma l'elenco deve essere rivisto prima di decidere di farlo.

Controllo del meccanismo allowlisting

Sono disponibili tre proprietà di sistema per controllare il meccanismo allowlisting:

com.ibm.mq.jms.allowlist (JMS 2.0) e com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
Questa proprietà può essere specificata in uno dei seguenti modi:

- Il nome del percorso del file che contiene l'elenco consentiti, in formato URI del file (che inizia con `file:`). In modalità DISCOVERY, questo file viene scritto dal meccanismo allowlisting. Il file non deve esistere. Se il file esiste, il meccanismo genera un'eccezione invece di sovrascriverla. In modalità ENFORCEMENT, questo file viene letto dal meccanismo allowlisting.
- Una virgola separata di nomi di classe completi che costituiscono l'elenco consentiti.

Se questa proprietà non è impostata, il meccanismo allowlist è inattivo.

Se si utilizza un Java security manager, è necessario assicurarsi che i file JAR IBM MQ classes for JMS abbiano accesso in lettura e scrittura a questo file.

com.ibm.mq.jms.allowlist.discover (JMS 2.0) e com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- Se questa proprietà non è impostata o è impostata su false, il meccanismo allowlist viene eseguito in modalità ENFORCEMENT.
- Se questa proprietà è impostata su true e l'allowlist è stato specificato come URI del file, il meccanismo allowlist viene eseguito in modalità DISCOVERY.
- Se questa proprietà è impostata su true e l'allowlist è stato specificato come un elenco di nomi classe, il meccanismo allowlist genera un'eccezione appropriata.
- Se questa proprietà è impostata su true e la allowlist non è stata specificata utilizzando la proprietà com.ibm.mq.jms.allowlist o com.ibm.mq.jakarta.jms.allowlist, il meccanismo allowlist è inattivo.
- Se questa proprietà è impostata su true e il file allowlist esiste già, il meccanismo allowlist genera una java.io.InvalidClassException e le voci non vengono aggiunte al file.

com.ibm.mq.jms.allowlist.mode (JMS 2.0) e com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)
Questa proprietà stringa può essere specificata in uno dei seguenti tre modi:

- Se questa proprietà è impostata su SERIALIZE, la modalità ENFORCEMENT esegue la convalida allowlist solo sul metodo ObjectMessage.setObject().
- Se questa proprietà è impostata su DESERIALIZE, la modalità ENFORCEMENT esegue la convalida allowlist solo sul metodo ObjectMessage.getObject().
- Se questa proprietà non è impostata o è impostata su qualsiasi altro valore, la modalità ENFORCEMENT esegue la convalida allowlist sia sui metodi ObjectMessage.getObject() che ObjectMessage.setObject().

Formato del file allowlist

Queste sono le caratteristiche principali del formato del file allowlist:

- Il file allowlist si trova nella codifica del file della piattaforma predefinita con le terminazioni di riga appropriate per la piattaforma.

  Nota: Se viene utilizzato un file allowlist, tale file viene sempre scritto e letto utilizzando la codifica file predefinita per la JVM.

  Ciò è valido se il file allowlist viene generato in uno dei seguenti modi:

  – z/OS Generato da un'applicazione autonoma in esecuzione in z/OS e utilizzato dalle altre applicazioni autonome in esecuzione anche in z/OS.

  - Generato da un'applicazione in esecuzione all'interno di WebSphere Application Server su qualsiasi piattaforma e utilizzato da un'altra istanza di WebSphere Application Server.
  - Multi Generato da un'applicazione autonoma in esecuzione su IBM MQ for Multiplatformse utilizzato da altre applicazioni autonome in esecuzione su IBM MQ for Multiplatformso da applicazioni in esecuzione all'interno di WebSphere Application Server su qualsiasi piattaforma.

  Tuttavia, poiché WebSphere Application Server utilizza ASCII e una JVM autonoma utilizza EBCDIC, si verificheranno problemi di codifica dei file se il file allowlist viene generato in uno dei seguenti modi:

  - Generato su z/OS, quindi utilizzato da applicazioni autonome in esecuzione su una piattaforma diversa da z/OS o da WebSphere Application Server.
  - Generato da WebSphere Application Server o da un'applicazione autonoma in esecuzione su una piattaforma diversa da z/OS, quindi utilizzato da un'applicazione autonoma su z/OS.
- Ogni riga non vuota contiene un nome classe completo. Le righe vuote vengono ignorate.
- I commenti possono essere inclusi - tutto ciò che segue un carattere '#', alla fine della riga, viene ignorato.
- Esiste un meccanismo di caratteri jolly di base:
  - '*' può essere l'ultimo elemento di un nome classe.
  - '*' corrisponde a un elemento **singolo** di un nome classe, vale a dire, la classe, ma non fa parte del package.

  Quindi, `com.ibm.mq.*` corrisponderebbe `com.ibm.mq.MQMessage` ma non `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  Il carattere jolly non funziona per le classi nel pacchetto predefinito che è per le classi senza un nome pacchetto esplicito, quindi un nome classe di "*" viene rifiutato.
- I file allowlist formattati in modo non corretto, ad esempio i file che contengono una voce come `com.ibm.mq.*.Message`, in cui il carattere jolly non è l'ultimo elemento, generano una java.lang.IllegalArgumentException .
- Un file allowlist vuoto ha l'effetto di disabilitare completamente l'utilizzo di ObjectMessage.

Formato dell'elenco consentiti come elenco separato da virgole

Lo stesso meccanismo di caratteri jolly è disponibile per un allowlist come elenco separato da virgole.

- Il carattere '*' può essere espanso dal sistema operativo se specificato su una riga comandi o in uno script shell o in un file batch, quindi potrebbe richiedere una gestione speciale.
- Il carattere di commento '#' è applicabile solo quando viene specificato un file. Se l'allowlist viene specificato come un elenco separato da virgole di nomi classe, supponendo che il sistema operativo o la shell non lo elaborino, poiché si tratta del carattere di commento predefinito in molte shell AIX and Linux , viene considerato come un carattere normale.

Quando avviene la creazione di un elenco?

Allowlisting viene avviato quando l'applicazione esegue per la prima volta un metodo ObjectMessage setMessage() o getMessage().

Le proprietà di sistema vengono valutate, il file allowlist viene aperto e, in modalità ENFORCEMENT, l'elenco delle classi consentite viene caricato quando il meccanismo viene inizializzato. A questo punto, viene scritta una voce nel file di log IBM MQ JMS per l'applicazione.

Quando il meccanismo viene inizializzato, i relativi parametri potrebbero non essere modificati. Poiché il tempo di inizializzazione non è facilmente previsto in quanto dipende dal funzionamento dell'applicazione. Le impostazioni delle proprietà di sistema e il contenuto del file allowlist devono quindi essere considerati fissi dal momento in cui viene avviata l'applicazione. Non modificare le proprietà o il contenuto del file allowlist mentre l'applicazione è in esecuzione, poiché i risultati non sono garantiti.

Punti da considerare

L'approccio migliore per mitigare i rischi intrinseci al meccanismo di serializzazione Java consiste nell'esplorare approcci alternativi al trasferimento dei dati, ad esempio l'utilizzo di JSON invece di ObjectMessage. Utilizzando i meccanismi AMS (Advanced Message Security) è possibile aggiungere ulteriore sicurezza assicurando che i messaggi provengano da origini attendibili.

Se si utilizza il meccanismo Java security manager con l'applicazione, è necessario concedere le seguenti autorizzazioni:

- FilePermission su qualsiasi file allowlist utilizzato, con autorizzazione di lettura per la modalità ENFORCEMENT, autorizzazione di scrittura per la modalità DISCOVER.
- JMS 2.0 PropertyPermission (lettura) nelle proprietà **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover**e **com.ibm.mq.jms.allowlist.mode** .
- JM 3.0 PropertyPermission (lettura) nelle proprietà **com.ibm.mq.jakarta.jms.allowlist**, **com.ibm.mq.jakarta.jms.allowlist.discover**e **com.ibm.mq.jakarta.jms.allowlist.mode** .

Ulteriori informazioni

Consultare “Impostazione e utilizzo di un elenco di consentiti JMS o Jakarta Messaging” a pagina 137 e “Elenco consentito in WebSphere Application Server” a pagina 139 per ulteriori informazioni sugli elenchi consentiti.

Concetti correlati

“Esecuzione delle applicazioni IBM MQ classes for JMS in Java security manager” a pagina 109
IBM MQ classes for JMS può essere eseguito con Java security manager abilitato. Per eseguire correttamente le applicazioni con Java security manager abilitato, è necessario configurare Java Virtual Machine (JVM) con un file di configurazione della politica adatto.

Impostazione e utilizzo di un elenco di consentiti JMS o Jakarta Messaging

Queste informazioni indicano come funziona un allowlist e come è possibile configurarne uno utilizzando la funzionalità contenuta in IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging per generare un file allowlist, contenente un elenco dei tipi di ObjectMessages che un'applicazione può elaborare.

Prima di iniziare

Importante:

Ovunque possibile, il termine *allowlist* ha sostituito il termine *whitelist*. Per le release IBM MQ 9.0 e successive, sono inclusi alcuni nomi di proprietà di sistema Java menzionati in questo argomento. Non è necessario modificare alcuna configurazione esistente. Anche i precedenti nomi di proprietà di sistema continuano a funzionare.

Prima di iniziare questa attività, assicurarsi di aver letto e compreso “Concetti di elenco consentiti” a pagina 134

Informazioni su questa attività

Poiché JMS e Jakarta Messaging condividono molto in comune, ulteriori riferimenti a JMS in questo argomento possono essere considerati come riferimenti ad entrambi. Eventuali differenze vengono evidenziate come necessario.

Una volta abilitata la funzionalità allowlisting, IBM MQ classes for JMS utilizza tale funzione nei modi seguenti:

- Quando un'applicazione desidera inviare un ObjectMessage, può crearlo in due modi, richiamando:
  - Session.createObjectMessage(Serializabile), passando l'oggetto che deve essere contenuto nel messaggio.

– Session.createObjectMessage() per creare un ObjectMessagevuoto e quindi richiamare ObjectMessage.setObject(Serializable) per memorizzare l'oggetto da inviare all'interno di ObjectMessage.

Quando vengono richiamati i metodi Session.createObjectMessage(Serializable) o ObjectMessage.setObject(Serializable), le classi per JMS verificano se l'oggetto inoltrato è di un tipo menzionato nell'allowlist.

Se è di un tipo menzionato, l'oggetto viene serializzato e memorizzato in ObjectMessage. Tuttavia, se l'oggetto è di un tipo che non è presente nell'allowlist, IBM MQ classes for JMS genera una JMSException contenente il messaggio:

```
JMSCC0052: Si è verificata un'eccezione durante la serializzazione dell'oggetto:
'java.io.InvalidClassException: < classe oggetto>; la classe potrebbe non essere serializzata
o deserializzato poiché non è stato incluso nell'elenco consentiti '< allowlist>'.
```

all'applicazione.

Importante: Se l'eccezione viene generata dal metodo Session.createObjectMessage(Serializable), ObjectMessage non verrà creato. Allo stesso modo, se JMSException viene generata dal metodo ObjectMessage.setObject(Serializable), l'oggetto non verrà aggiunto a ObjectMessage.

- Se un'applicazione riceve un ObjectMessage, richiama il metodo ObjectMessage.getObject() per ottenere l'oggetto contenuto al suo interno. Quando viene richiamato questo metodo, IBM MQ classes for JMS controlla il tipo di oggetto contenuto in ObjectMessage, per vedere se tale oggetto è di un tipo specificato nell'allowlist.

  In questo caso, l'oggetto viene deserializzato e restituito all'applicazione. Tuttavia, se l'oggetto è di un tipo che non è presente nell'allowlist, IBM MQ classes for JMS genera una JMSException contenente il messaggio:

  ```
  JMSCC0053: Si è verificata un'eccezione durante la deserializzazione di un messaggio:
  'java.io.InvalidClassException: < classe oggetto>; La classe potrebbe non essere
  serializzato o deserializzato poiché non è stato incluso nel
  allowlist '< allowlist>'. '.'
  ```

  all'applicazione.

Ad esempio, si supponga che l'applicazione contenga il seguente codice per l'invio di un ObjectMessage contenente un oggetto di tipo java.net.URI:

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

Poiché allowlisting non è abilitato, l'applicazione è in grado di inserire correttamente il messaggio nella destinazione richiesta.

Se si crea un file denominato `C:\allowlist.txt` contenente una singola voce, `java.net.URL`e si avvia di nuovo l'applicazione con la serie di proprietà di sistema Java :

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

la funzionalità allowlist è abilitata. L'applicazione è ancora in grado di creare e inviare l'oggetto ObjectMessage contenente un oggetto di tipo java.net.URI , poiché tale tipo è specificato nella allowlist.

Tuttavia, se si modifica il file `allowlist.txt` in modo che il file contenga la singola voce `java.util.Calendar`, poiché la funzionalità allowlist è ancora abilitata, quando l'applicazione richiama:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS controllare la allowlist e rilevare che non contiene una voce per java.net.URI.

Di conseguenza, viene generata una JMSException contenente il messaggio JMSCC0052 .

Allo stesso modo, si supponga di avere un'altra applicazione che riceve ObjectMessages utilizzando questo codice:

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

Se allowlisting non è abilitato, l'applicazione è in grado di ricevere ObjectMessages che contengono un oggetto di qualsiasi tipo. L'applicazione verifica quindi se l'oggetto è di tipo java.net.URL prima di eseguire l'elaborazione appropriata.

Se ora si avvia l'applicazione con la proprietà di sistema Java :

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

la funzionalità allowlisting è attivata. Quando l'applicazione richiama:

```
Object messageBody = objectMessage.getObject();
```

il metodo ObjectMessage.getObject() restituisce solo oggetti di tipo java.net.URL.

Se l'oggetto contenuto in ObjectMessage non è di questo tipo, il metodo ObjectMessage.getObject() genera una JMSException contenente il messaggio JMSCC0053 . L'applicazione deve quindi decidere cosa fare con il messaggio; ad esempio, il messaggio potrebbe essere spostato nella coda di messaggi non recapitabili per tale gestore code.

L'applicazione viene restituita normalmente solo se l'oggetto all'interno di ObjectMessage è del tipo java.net.URL.

Procedura

1. Eseguire l'applicazione che elabora ObjectMessages, con le seguenti proprietà di sistema Java specificate:

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Quando l'applicazione viene eseguita, il IBM MQ classes for JMS crea un file che contiene i tipi di oggetti elaborati dall'applicazione.
2. Una volta che l'applicazione ha elaborato un campione rappresentativo di ObjectMessages in un periodo di tempo, arrestarlo.

   Il file allowlist contiene ora un elenco di tutti i tipi di oggetti contenuti in ObjectMessages elaborati dall'applicazione durante l'esecuzione.

   Se l'applicazione è stata eseguita per un periodo di tempo sufficiente, questo elenco include tutti i possibili tipi di oggetti contenuti in ObjectMessages che l'applicazione probabilmente gestirà.
3. Riavviare l'applicazione con la seguente proprietà di sistema impostata:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Ciò abilita allowlisting e se IBM MQ classes for JMS rileva un ObjectMessage di un tipo che non è nell'allowlist, viene generata una JMSException contenente il messaggio JMSCC0052 o JMSCC0053 .

Elenco consentito in WebSphere Application Server

Come utilizzare IBM MQ classes for JMS allowlisting in WebSphere Application Server.

Importante:

Ovunque possibile, il termine *allowlist* ha sostituito il termine *whitelist*. Per le release IBM MQ 9.0 e successive, sono inclusi alcuni nomi di proprietà di sistema Java menzionati in questo argomento. Non è necessario modificare alcuna configurazione esistente. Anche i precedenti nomi di proprietà di sistema continuano a funzionare.

È necessario assicurarsi che l'installazione di WebSphere Application Server includa una versione dell'adattatore di risorse IBM MQ che supporta allowlisting.

Consultare “Utilizzo congiunto di IBM MQ e WebSphere Application Server” a pagina 500 per ulteriori informazioni sull'utilizzo dei due prodotti.

IBM MQ 9.0.0 Fix Pack 1 e successive includono la funzionalità appropriata.

Una volta aggiornato il server delle applicazioni, è possibile utilizzare le proprietà di sistema Java :

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

descritto in “Impostazione e utilizzo di un elenco di consentiti JMS o Jakarta Messaging” a pagina 137.

Nota: È necessario impostare le proprietà di sistema Java come argomenti JVM generici, sul Java Virtual Machine utilizzato per eseguire il server delle applicazioni e il server delle applicazioni riavviato per rendere effettive le modifiche.

Per ulteriori informazioni, consultare la sezione *Argomenti JVM generici* in Impostazioni della macchina virtualeJava .

Per impostare le proprietà, andare alla finestra Java Virtual Machine in *Definizioni di processo* e immettere l'argomento appropriato.

La seguente impostazione:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

fa sì che il server delle applicazioni utilizzi l'elenco di consentiti *youruserId_MyObject*. Solo gli oggetti del tipo vengono elaborati dal server delle applicazioni.

Le seguenti impostazioni:

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

configurare il server delle applicazioni per utilizzare la modalità *Rileva* e registrare i dettagli di JMS ObjectMessages, elaborati dal server delle applicazioni, nel file `C:\allowlist.txt`

La seguente impostazione:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

fa sì che il server delle applicazioni carichi il file `C:/allowlist.txte` utilizza le informazioni contenute in tale file per determinare l'elenco consentiti.

Concetti correlati

“Esecuzione delle applicazioni IBM MQ classes for JMS in Java security manager” a pagina 109
IBM MQ classes for JMS può essere eseguito con Java security manager abilitato. Per eseguire correttamente le applicazioni con Java security manager abilitato, è necessario configurare Java Virtual Machine (JVM) con un file di configurazione della politica adatto.

Conversioni di stringhe di caratteri in IBM MQ classes for JMS

IBM MQ classes for JMS utilizza CharsetEncoders e CharsetDecoders direttamente per la conversione della stringa di caratteri. Il funzionamento predefinito per la conversione della stringa di caratteri può essere configurato con due proprietà di sistema. La gestione dei messaggi che contengono caratteri non associabili può essere configurata mediante le proprietà del messaggio per impostare l'azione UnmappableCharactere i byte di sostituzione.

Prima di IBM MQ 8.0, le conversioni di stringhe in IBM MQ classes for JMS venivano eseguite richiamando i metodi `java.nio.charset.Charset.decode(ByteBuffer)` e `Charset.encode(CharBuffer)` .

L'utilizzo di uno di questi metodi determina una sostituzione predefinita (REPLACE) di dati non corretti o non convertibili. Questo comportamento può oscurare gli errori nelle applicazioni e causare caratteri imprevisti, ad esempio ?, nei dati tradotti.

Da IBM MQ 8.0, per rilevare tali problemi prima e in modo più efficace, IBM MQ classes for JMS utilizzare CharsetEncoders e CharsetDecoders direttamente e configurare esplicitamente la gestione di dati con formato non corretto e non convertibili. Il comportamento predefinito è di REPORT tali problemi generando un `MQException`adatto.

Configurazione

La conversione da UTF-16 (la rappresentazione dei caratteri utilizzata in Java) a una serie di caratteri nativo, come UTF-8, viene definita *codifica*, mentre la conversione nella direzione opposta viene definita *decodifica*.

La decodifica assume il comportamento predefinito per `CharsetDecoders`, riportando errori generando un'eccezione.

Un'impostazione viene utilizzata per specificare un `java.nio.charset.CodingErrorAction` per controllare la gestione degli errori sia sulla codifica che sulla decodificazione. Un'altra impostazione viene utilizzata per controllare il byte o i byte di sostituzione durante la codifica. La stringa di sostituzione Java predefinita verrà utilizzata nelle operazioni di decodifica.

UnmappableCharacterImpostazioni byte di sostituzione e azione in IBM MQ classes for JMS

Da IBM MQ 8.0, le seguenti proprietà sono disponibili per l'impostazione dell'azione UnmappableCharactere i byte di sostituzione. Le definizioni di costante appropriate si trovano in `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_UNMAPPABLE_AZIONE
Imposta o richiama il `CodingErrorAction` da applicare quando non è possibile associare un carattere in un'operazione di codifica o decodifica.

Impostare questo valore come `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` come riportato di seguito:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
Imposta o richiama i byte di sostituzione da applicare quando non è possibile associare un carattere in un'operazione di codifica.

La stringa di sostituzione Java predefinita viene utilizzata nelle operazioni di decodifica.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

Le proprietà `JMS_IBM_UNMAPPABLE_ACTION` e `JMS_IBM_UNMAPPABLE_REPLACEMENT` possono essere impostate su destinazioni o messaggi. Un valore impostato su un messaggio sovrascrive il valore impostato sulla destinazione a cui viene inviato il messaggio.

Si noti che `JMS_IBM_UNMAPPABLE_REPLACEMENT` deve essere impostato come byte singolo.

Proprietà di sistema per l'impostazione dei valori predefiniti di sistema

Da IBM MQ 8.0, le seguenti proprietà di sistema Java sono disponibili per configurare il funzionamento predefinito relativo alla conversione della stringa di caratteri.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Specifica l'azione da intraprendere per i dati non convertibili sulla codifica e la decodifica. Il valore può essere `REPORT`, `REPLACE`o `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Imposta o richiama i byte di sostituzione da applicare quando un carattere non può essere mappato in un'operazione di codifica. La stringa di sostituzione predefinita Java viene utilizzata nelle operazioni di decodifica.

Per evitare confusione tra rappresentazioni di byte nativi e di caratteri Java , è necessario specificare `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` come numero decimale che rappresenta il byte di sostituzione nella serie di caratteri nativi.

Ad esempio, il valore decimale di ?, come byte nativo, è 63 se la serie di caratteri nativi è basata su ASCII, ad esempio ISO-8859-1, mentre è 111 se la serie di caratteri nativi è EBCDIC.

Nota: Si noti che se un oggetto MQMD o MQMessage ha i campi **`unmappableAction`** o **`unMappableReplacement`** impostati, i valori di questi campi hanno la precedenza sulle proprietà di sistema Java . Ciò consente ai valori specificati dalle proprietà di sistema Java di essere sovrascritti per ogni messaggio, se necessario.

Concetti correlati

“Conversioni di stringhe di caratteri in IBM MQ classes for Java” a pagina 353
IBM MQ classes for Java utilizza CharsetEncoders e CharsetDecoders direttamente per la conversione della stringa di caratteri. Il funzionamento predefinito per la conversione della stringa di caratteri può essere configurato con due proprietà di sistema. La gestione dei messaggi che contengono caratteri non associabili può essere configurata tramite `com.ibm.mq.MQMD`.

Scrittura di applicazioni IBM MQ classes for JMS/Jakarta Messaging

Dopo una breve introduzione al modello JMS , questa sezione fornisce una guida dettagliata su come scrivere applicazioni IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging .

Informazioni su questa attività

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, consultare Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging.

Concetti correlati

IBM MQ classes for Jakarta Messaging: una panoramica

Il modello JMS e Jakarta Messaging

Il modello JMS e Jakarta Messaging definisce una serie di interfacce che le applicazioni Java possono utilizzare per eseguire operazioni di messaggistica. IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono entrambi provider di messaggistica. Definiscono il modo in cui gli oggetti JMS e Jakarta Messaging sono correlati ai concetti IBM MQ . Le specifiche JMS e Jakarta Messaging prevedono che alcuni oggetti JMS e Jakarta Messaging siano oggetti gestiti.

JMS 2.0 IBM MQ 8.0 ha aggiunto il supporto per la versione JMS 2.0 dello standard JMS , che ha introdotto un'API semplificata, conservando anche l'API classica, da JMS 1.1.

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, è supportato Jakarta Messaging 3.0 per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API JMS 2.0 che l'API Jakarta Messaging 3.0 nella stessa applicazione.

Nota: Per Jakarta Messaging 3.0, il controllo della specifica JMS passa da Oracle a Java Community Process. Tuttavia, Oracle conserva il controllo del nome "javax", che viene utilizzato in altre tecnologie Java che non sono state spostate nel processo della comunità Java . Quindi, mentre Jakarta Messaging 3.0 è funzionalmente equivalente a JMS 2.0 ci sono alcune differenze nella denominazione:

- Il nome ufficiale per la versione 3.0 è Jakarta Messaging invece di Java Message Service.

- I nomi di package e costanti hanno come prefisso `jakarta` anziché `javax`. Ad esempio, in JMS 2.0 la connessione iniziale a un fornitore di messaggistica è un oggetto `javax.jms.Connection` e in Jakarta Messaging 3.0 è un oggetto `jakarta.jms.Connection` .

JMS 2.0 I package javax.jms definiscono le interfacce JMS e un fornitore JMS implementa tali interfacce per un prodotto di messaggistica specifico. IBM MQ classes for JMS è un provider JMS che implementa le interfacce JMS per IBM MQ.

JM 3.0 I pacchetti jakarta.jms definiscono le interfacce Jakarta Messaging e un provider Jakarta Messaging implementa tali interfacce per un prodotto di messaggistica specifico. IBM MQ classes for Jakarta Messaging è un provider Jakarta Messaging che implementa le interfacce Jakarta Messaging per IBM MQ.

Poiché JMS e Jakarta Messaging condividono molto in comune, ulteriori riferimenti a JMS in questo argomento possono essere considerati come riferimenti ad entrambi. Eventuali differenze vengono evidenziate come necessario.

API semplificata

JMS 2.0 ha introdotto l'API semplificata, mantenendo anche le interfacce specifiche del dominio e indipendenti dal dominio da JMS 1.1. L'API semplificata riduce il numero di oggetti necessari per inviare e ricevere messaggi e consiste nelle seguenti interfacce:

ConnectionFactory
Un ConnectionFactory è un oggetto gestito utilizzato da un client JMS per creare una connessione. Questa interfaccia viene utilizzata anche nell'API classica.

JMSContesto
Questo oggetto combina gli oggetti Connection e Session dell'API classica. JMSGli oggetti di contesto possono essere creati da altri JMSoggetti di contesto, con la connessione sottostante duplicata.

JMSMittente
Un Producer JMSviene creato da un contesto JMSe viene utilizzato per inviare messaggi a una coda o a un argomento. L'oggetto Producer JMSprovoca la creazione degli oggetti richiesti per inviare il messaggio.

JMSDestinatario
Un consumer JMSviene creato da un contesto JMSe viene utilizzato per ricevere messaggi da un argomento o da una coda.

L'API semplificata ha una serie di effetti:

- L'oggetto Context JMSavvia sempre automaticamente la connessione sottostante.
- JMSI produttori e JMSi consumatori possono ora lavorare direttamente con il corpo del messaggio, senza dover ottenere l'intero oggetto del messaggio, utilizzando il metodo getBody del messaggio.
- Le proprietà dei messaggi possono essere impostati sull'oggetto JMSProducer, utilizzando il concatenamento di metodo, prima di inviare un 'corpo ', un contenuto di messaggi. Il Producer JMSgestirà la creazione di tutti gli oggetti necessari per inviare il messaggio. Utilizzando JMS 2.0, è possibile impostare le proprietà e inviare un messaggio nel modo seguente:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 ha anche introdotto sottoscrizioni condivise in cui i messaggi possono essere condivisi tra più consumer. Tutte le sottoscrizioni JMS 1.1 vengono considerate sottoscrizioni non condivise.

API classica

Il seguente elenco riepiloga le principali interfacce JMS dell'API classica:

Destinazione
Una destinazione è il punto in cui un'applicazione invia i messaggi o è un'origine da cui un'applicazione riceve i messaggi o entrambi.

ConnectionFactory
Un oggetto ConnectionFactory contiene una serie di proprietà di configurazione per una connessione. Un'applicazione utilizza una produzione connessioni per creare una connessione.

Connessione
Un oggetto Connection incapsula una connessione attiva dell'applicazione a un server di messaggistica. Un'applicazione utilizza una connessione per creare sessioni.

Sessione
Una sessione è un contesto a thread singolo per inviare e ricevere messaggi. Un'applicazione utilizza una sessione per creare messaggi, produttori di messaggi e utenti di messaggi. Una sessione è sottoposta a transazione o non è stata sottoposta a transazione.

Messaggio
Un oggetto Message incapsula un messaggio che un'applicazione invia o riceve.

MessageProducer
Un'applicazione utilizza un produttore di messaggi per inviare messaggi a una destinazione.

MessageConsumer
Un'applicazione utilizza un utente di messaggi per ricevere i messaggi inviati a una destinazione.

Figura 9 a pagina 144 mostra questi oggetti e le relazioni.

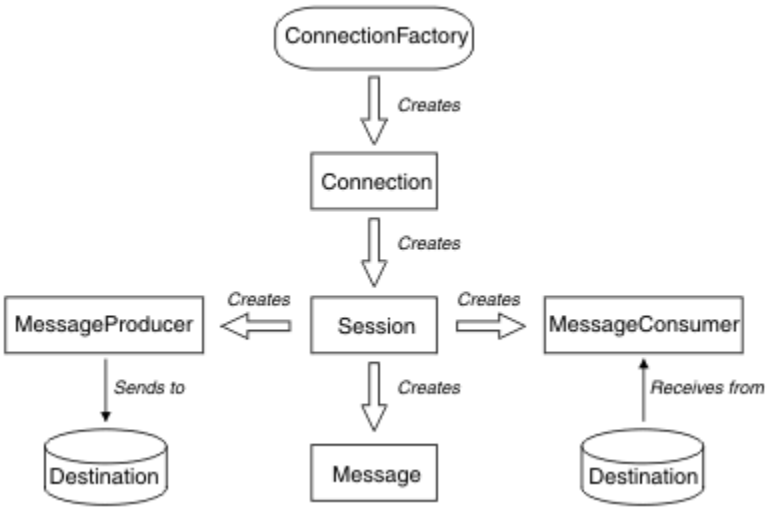


Figura 9. Oggetti JMS e relative relazioni

Un oggetto di destinazione, ConnectionFactoryo Connection può essere utilizzato contemporaneamente da thread differenti di un'applicazione a più thread, ma un oggetto Session, MessageProducero MessageConsumer non può essere utilizzato contemporaneamente da thread differenti. Il modo più semplice per garantire che un oggetto Session, MessageProducero MessageConsumer non venga utilizzato contemporaneamente consiste nel creare un oggetto Session separato per ciascun thread.

Domini di messaggistica

JMS supporta due stili di messaggistica:

- Messaggistica point-to-point
- Pubblicazione/sottoscrizione della messaggistica

Questi stili di messaggistica vengono anche indicati come *domini di messaggistica*ed è possibile combinare entrambi gli stili di messaggistica in un'applicazione. Nel dominio point-to-point, una

destinazione è una coda e, nel dominio di pubblicazione / sottoscrizione, una destinazione è un argomento.

Con le versioni di JMS precedenti a JMS 1.1, la programmazione per il dominio point-to-point utilizza una serie di interfacce e metodi, mentre la programmazione per il dominio di pubblicazione / sottoscrizione utilizza un'altra serie. I due set sono simili, ma separati. Da JMS 1.1, è possibile utilizzare una serie comune di interfacce e metodi che supporta entrambi i domini di messaggistica. Le interfacce comuni forniscono una vista indipendente dal dominio di ciascun dominio di messaggistica. Tabella 15 a pagina 145 elenca le interfacce indipendenti del dominio JMS e le relative interfacce specifiche del dominio corrispondenti.

Tabella 15. Interfacce specifiche del dominio e indipendenti dal dominio JMS

Interfacce indipendenti dal dominio	**Interfacce specifiche del dominio per il dominio point - to - point**	**Interfacce specifiche del dominio per il dominio di pubblicazione / sottoscrizione**
ConnectionFactory	Factory QueueConnection	Factory TopicConnection
Connessione	QueueConnection	TopicConnection
Destinazione	Coda	Argomento
Sessione	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 supporta sia le precedenti interfacce specifiche del dominio JMS 1.1 che l'API semplificata di JMS 2.0. IBM MQ classes for JMS 2.0 può quindi essere utilizzato per gestire le applicazioni esistenti, incluso lo sviluppo di nuove funzioni in applicazioni esistenti.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 supporta le versioni Jakarta Messaging delle stesse interfacce ed è consigliato per lo sviluppo di nuove applicazioni.

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, gli oggetti JMS sono correlati a concetti IBM MQ nei seguenti modi:

- Un oggetto Connection ha proprietà derivate dalle proprietà del factory di connessione utilizzato per creare la connessione. Queste proprietà controllano il modo in cui un'applicazione si connette a un gestore code. Esempi di queste proprietà sono il nome del gestore code e, per un'applicazione che si connette al gestore code in modalità client, il nome host o l'indirizzo IP del sistema su cui è in esecuzione il gestore code.
- Un oggetto Session incapsula un handle di connessione IBM MQ , che definisce quindi l'ambito transazionale della sessione.
- Un oggetto MessageProducer e un oggetto MessageConsumer incapsulano un handle di oggetto IBM MQ .

Quando si utilizza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, vengono applicate tutte le normali regole di IBM MQ . Notare, in particolare, che un'applicazione può inviare un messaggio a una coda remota, ma può ricevere un messaggio solo da una coda di proprietà del gestore code a cui è connessa l'applicazione.

La specifica JMS prevede che gli oggetti ConnectionFactory e Destination siano oggetti gestiti. Un amministratore crea e gestisce gli oggetti gestiti in un repository centrale e un'applicazione JMS richiama tali oggetti utilizzando JNDI (Java Naming and Directory Interface).

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, l'implementazione dell'interfaccia Destinazione è una superclasse astratta di Coda e Argomento, e quindi un'istanza di Destinazione è un oggetto Coda o un oggetto Argomento. Le interfacce indipendenti dal dominio trattano una coda o

un argomento come una destinazione. Il dominio di messaggistica per un oggetto MessageProducer o MessageConsumer è determinato se la destinazione è una coda o un argomento.

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging , quindi, gli oggetti dei seguenti tipi possono essere oggetti gestiti:

- ConnectionFactory
- Factory QueueConnection
- Factory TopicConnection
- Coda
- Argomento
- XAConnectionFactory
- Factory XAQueueConnection
- Factory XATopicConnection

Concetti correlati

Interfacce di lingua IBM MQ Java

“Creazione e configurazione di factory di connessione e destinazioni” a pagina 207

Un'applicazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging può creare factory di connessione e destinazioni richiamandoli come oggetti gestiti da uno spazio dei nomi JNDI (Java Naming and Directory Interface), utilizzando le estensioni IBM JMS o utilizzando le estensioni IBM MQ JMS . Un'applicazione può anche utilizzare le estensioni IBM JMS o IBM MQ JMS per impostare le proprietà delle factory di connessione e delle destinazioni.

JMS messaggi

I messaggi JMS sono composti da un'intestazione, proprietà e un corpo. JMS definisce cinque tipi di corpo del messaggio.

I messaggi JMS sono composti dalle parti seguenti:

Intestazione
Tutti i messaggi supportano la stessa serie di campi di intestazione. I campi di intestazione contengono valori utilizzati dai client e dai provider per identificare e instradare i messaggi.

Proprietà
Ogni messaggio contiene una funzionalità integrata per supportare i valori delle proprietà definite dall'applicazione. Le proprietà forniscono un meccanismo efficiente per filtrare i messaggi definiti dall'applicazione.

Corpo
JMS definisce cinque tipi di corpo del messaggio che coprono la maggior parte degli stili di messaggistica attualmente in uso:

Flusso
Un flusso di valori primitivi Java . Viene riempito e letto in modo sequenziale.

Associazione
Una serie di coppie nome - valore, dove i nomi sono stringhe e i valori sono Java tipi primitivi. È possibile accedere alle voci in modo sequenziale o casuale per nome. L'ordine delle voci non è definito.

Testo
Un messaggio contenente un java.lang.String.

Oggetto
Un messaggio che contiene un oggetto Java serializzabile

Byte
Un flusso di byte non interpretati. Questo tipo di messaggio è per codificare letteralmente un corpo in modo che corrisponda a un formato di messaggio esistente.

Il campo di intestazione JMSCorrelationID viene utilizzato per collegare un messaggio con un altro. In genere collega un messaggio di risposta con il messaggio di richiesta. JMSCorrelationID può contenere un ID messaggio specifico del provider, una stringa specifica dell'applicazione o un valore byte nativo del provider [].

Selettori di messaggi in JMS
I messaggi possono contenere valori di proprietà definiti dall'applicazione. Un'applicazione può utilizzare i selettori di messaggi per disporre di messaggi di filtro del provider JMS .

Un messaggio contiene una funzionalità integrata per supportare i valori delle proprietà definite dall'applicazione. In effetti, ciò fornisce un meccanismo per aggiungere campi di intestazione specifici dell'applicazione a un messaggio. Le proprietà consentono a un'applicazione, utilizzando i selettori di messaggi, di fare in modo che un provider JMS selezioni o filtri i messaggi per suo conto, utilizzando criteri specifici dell'applicazione. Le proprietà definite dall'applicazione devono rispettare le seguenti regole:

- I nomi delle proprietà devono rispettare le regole per un identificativo del selettore messaggi.
- I valori delle proprietà possono essere booleani, byte, short, int, long, float, double e String.
- I prefissi nome JMSX e JMS_ sono riservati.

I valori delle proprietà vengono impostati prima di inviare un messaggio. Quando un client riceve un messaggio, le proprietà del messaggio sono di sola lettura. Se un client tenta di impostare le proprietà a questo punto, viene generata unaWriteableException MessageNot. Se viene richiamato clearProperties , le proprietà possono ora essere lette e scritte.

Un valore di proprietà potrebbe duplicare un valore in un corpo del messaggio. JMS non definisce una politica per ciò che potrebbe essere trasformato in una proprietà. Tuttavia, gli sviluppatori di applicazioni devono essere consapevoli del fatto che i provider JMS probabilmente gestiscono i dati in un corpo del messaggio in modo più efficiente rispetto ai dati nelle proprietà del messaggio. Per ottenere prestazioni ottimali, le applicazioni devono utilizzare le proprietà del messaggio solo quando è necessario personalizzare un'intestazione del messaggio. Il motivo principale di questa operazione è il supporto della selezione di messaggi personalizzati.

Un selettore messaggi JMS consente a un client di specificare i messaggi a cui è interessato utilizzando l'intestazione del messaggio. Vengono recapitati solo i messaggi con intestazioni che corrispondono al selettore.

I selettori dei messaggi non possono fare riferimento ai valori del corpo del messaggio.

Un selettore di messaggi corrisponde a un messaggio quando il selettore viene valutato true quando il campo di intestazione del messaggio e i valori della proprietà vengono sostituiti per i corrispondenti identificativi nel selettore.

Un selettore di messaggi è una stringa, con sintassi basata su un sottoinsieme della sintassi dell'espressione condizionale SQL92 . L'ordine in cui viene valutato un selettore di messaggi è da sinistra a destra all'interno di un livello di precedenza. È possibile utilizzare le parentesi per modificare questo ordine. I valori letterali selettori predefiniti e i nomi degli operatori vengono scritti qui in maiuscolo; tuttavia, non sono sensibili al maiuscolo / minuscolo.

Contenuto di un selettore di messaggi

Un selettore di messaggi può contenere:

- Valori letterali
  - Un letterale stringa è racchiuso tra virgolette. Una virgoletta doppia rappresenta una virgoletta. Gli esempi sono 'literal' e 'literal' '. Come i letterali della stringa Java , questi utilizzano la codifica dei caratteri Unicode.
  - Una costante letterale numerica esatta è un valore numerico senza una virgola decimale, come 57, -957 e +62. Sono supportati i numeri compresi nell'intervallo Java long.

  - Un letterale numerico approssimativo è un valore numerico in notazione scientifica, come 7E3 o -57.9E2, oppure un valore numerico con un decimale, come 7., -95.7o +6.2. Sono supportati i numeri compresi nell'intervallo Java double.
  - I letterali booleani TRUE e FALSE.
- Identificativi:
  - Un identificativo è una sequenza di lunghezza illimitata di Java lettere e Java cifre, la prima delle quali deve essere una lettera Java . Una lettera è qualsiasi carattere per cui il metodo Character.isJavaLetter restituisce true. Sono inclusi _ e $. Una lettera o una cifra è qualsiasi carattere per cui il metodo Character.isJavaLetterOrDigit restituisce true.
  - Gli identificativi non possono essere i nomi NULL, TRUE o FALSE.
  - Gli identificatori non possono essere NOT, AND, OR, BETWEEN, LIKE, IN o IS.
  - Gli identificatori sono riferimenti di campi di intestazione o riferimenti di proprietà.
  - Gli identificatori sono sensibili al maiuscolo / minuscolo.
  - I riferimenti del campo di intestazione del messaggio sono limitati a:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID, JMSTimestamp, JMSCorrelationIDe i valori JMSType possono essere null e, in tal caso, vengono considerati come un valore NULL.
  - Qualsiasi nome che inizia con JMSX è un nome di proprietà definito da JMS.
  - Qualsiasi nome che inizia con JMS_ è un nome di proprietà specifico del provider.
  - Qualsiasi nome che non inizia con JMS è un nome di proprietà specifico dell'applicazione. Se esiste un riferimento a una proprietà che non esiste in un messaggio, il suo valore è NULL. Se esiste, il suo valore è il valore della proprietà corrispondente.
- Lo spazio è lo stesso definito per Java: spazio, tabulazione orizzontale, avanzamento modulo e terminazione riga.
- Espressioni:
  - Un selettore è un'espressione condizionale. Un selettore che assume il valore di true corrisponde; un selettore che assume il valore di false o unknown non corrisponde.
  - Le espressioni aritmetiche sono composte da se stesse, operazioni aritmetiche, identificatori (con un valore trattato come una costante letterale numerica) e costanti letterali numeriche.
  - Le espressioni condizionali sono composte da se stesse, operazioni di confronto e operazioni logiche.
- La parentesi standard (), per impostare l'ordine in cui vengono valutate le espressioni, è supportata.
- Operatori logici in ordine di precedenza: NOT, AND, OR.
- Operatori di confronto: =,>,> =, <, < =, <> (non uguale).
  - È possibile confrontare solo i valori dello stesso tipo. Un'eccezione è che è valido per confrontare valori numerici esatti e valori numerici approssimati. (La conversione del tipo richiesta è definita dalle regole della promozione numerica Java .) Se si tenta di confrontare tipi diversi, il selettore è sempre false.
  - Il confronto tra stringhe e valori booleani è limitato a = e <>. Due stringhe sono uguali solo se contengono la stessa sequenza di caratteri.
- Operatori aritmetici in ordine di precedenza:
  - +, - unario.

  - *, /, moltiplicazione e divisione.
  - +, -, addizione e sottrazione.
  - Le operazioni aritmetiche su un valore NULL non sono supportate. Se vengono tentati, il selettore completo è sempre false.
  - Le operazioni aritmetiche devono utilizzare la promozione numerica Java .
- Operatore di confronto arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 e arithmetic-expr3 :
  - L'età BETWEEN 15 and 19 è equivalente all'età> = 15 AND age < = 19.
  - L'età NON COMPRESA tra 15 e 19 è equivalente all'età < 15 OR età> 19.
  - Se una delle espressioni di un'operazione BETWEEN è NULL, il valore dell'operazione è false. Se una delle espressioni di un'operazione NOT BETWEEN è NULL, il valore dell'operazione è true.
- identificativo [NOT] IN (string-literal1, string-literal2, ...) operatore di confronto in cui l'identificativo ha un valore stringa o NULL.
  - Il paese IN ("Regno Unito", "Stati Uniti", "Francia") è vero per "Regno Unito" e falso per "Perù". Equivale all'espressione (Paese = 'UK ') OR (Paese = 'US') OR (Paese = 'Francia ').
  - Il paese NOT IN ("Regno Unito", "Stati Uniti", "Francia") è falso per "Regno Unito" e vero per "Perù". È equivalente all'espressione NOT ((Paese = 'UK ') OR (Paese = 'US') OR (Paese = 'Francia ')).
  - Se l'identificativo di un'operazione IN o NOT IN è NULL, il valore dell'operazione è sconosciuto.
- l'operatore di confronto [NOT] LIKE valore - modello [ESCAPE carattere - escape], dove l'identificativo ha un valore stringa. pattern - value è una stringa letterale, dove _ sta per qualsiasi carattere singolo e% sta per qualsiasi sequenza di caratteri (inclusa la sequenza vuota). Tutti gli altri personaggi rappresentano se stessi. Il carattere escape facoltativo è una stringa di caratteri singoli, con un carattere che viene utilizzato per eseguire l'escape del significato speciale di _ e% nel valore del modello.
  - telefono LIKE '12%3' è true per 123 e 12993 e false per 1234.
  - la parola LIKE 'l_se' è true per "lose" e false per "loose".
  - LIKE '_ %' ESCAPE' \ ' è true per "_foo" e false per "bar".
  - telefono NOT LIKE '12%3' è false per 123 e 12993 e true per 1234.
  - Se l'identificativo di un'operazione LIKE o NOT LIKE è NULL, il valore dell'operazione è sconosciuto.
- L'operatore di confronto IS NULL verifica un valore campo di intestazione null o un valore di proprietà mancante.
  - nome_prop IS NULL.
- L'operatore di confronto IS NOT NULL verifica l'esistenza di un valore campo di intestazione non null o di un valore di proprietà.
  - nome_prop NON È NULL.

Esempio di selettore di messaggi

Il seguente selettore di messaggi seleziona i messaggi con un tipo di messaggio di auto, un colore di blu e un peso maggiore di 2500 libbre:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

Valori della proprietà NULL

Come indicato nell'elenco precedente, i valori delle proprietà possono essere NULL. La valutazione delle espressioni del selettore che contengono valori NULL è definita dalla semantica SQL 92 NULL. Il seguente elenco fornisce una breve descrizione di queste semantiche:

- SQL considera un valore NULL come sconosciuto.

- Il confronto o l'aritmetica con un valore sconosciuto produce sempre un valore sconosciuto.
- L'operatore IS NULL converte un valore sconosciuto in un valore TRUE.
- L'operatore IS NOT NULL converte un valore sconosciuto in un valore FALSE.

Funzionamento speciale di JMSMessageID e JMSCorrelationID

Le classi IBM MQ per JMS contengono ottimizzazioni quando si selezionano i messaggi da una coda in base a JMSMessageID o JMSCorrelationID.

Se un'applicazione specifica un selettore del modulo:

JMSMessageID= 'ID:*id_messaggio*'

dove *message_id* è una stringa contenente un identificativo di messaggio IBM MQ standard, le classi IBM MQ per JMS utilizzano **MatchOption** MQMO_MATCH_MSG_ID per ottenere il messaggio con l'identificativo di messaggio specificato.

Ad esempio, per richiamare un messaggio con l'identificativo 414D51207061756C745639314C545320C57C1A5F25ECE602 da una coda, un'applicazione deve utilizzare il seguente selettore di messaggi:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

Allo stesso modo, se l'applicazione specifica un selettore che ha il seguente formato:

JMSCorrelationID = 'ID:*id_correlazione*'

dove *correlation_id* è una stringa contenente un identificativo di correlazione IBM MQ standard, le classi IBM MQ per JMS utilizzano **MatchOption** MQMO_MATCH_CORREL_ID per richiamare il messaggio con l'identificativo di correlazione specificato dalla coda.

Nel seguente esempio, un selettore di messaggi viene utilizzato per ottenere un messaggio che ha un identificativo di correlazione 414D51207061756C745639314C545320846E5B5F25B1CC02:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Se un selettore di messaggi contiene un valore di tutti zeri per *message_id* o *correlation_id*, corrisponde a qualsiasi messaggio sulla coda. Ad esempio, se un'applicazione utilizza il selettore:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

quindi qualsiasi messaggio sulla coda viene considerato una corrispondenza e restituito all'applicazione.

Per ulteriori informazioni su MQMO_MATCH_MSG_ID e MQMO_MATCH_CORREL_ID **MatchOptions**, consultare MatchOptions (MQLONG).

Limitazioni

Sebbene SQL supporti il confronto decimale fisso e l'aritmetica, i selettori dei messaggi JMS non lo fanno. Questo è il motivo per cui le costanti letterali numeriche esatte sono limitate a quelle senza un decimale. È anche il motivo per cui ci sono numeri con un decimale come rappresentazione alternativa per un valore numerico approssimativo.

I commenti SQL non sono supportati.

Mappatura dei messaggi JMS sui messaggi IBM MQ
I messaggi IBM MQ sono composti da un descrittore del messaggio, da un'intestazione MQRFH2 facoltativa e da un corpo. Il contenuto di un messaggio JMS è parzialmente mappato e in parte copiato in un messaggio IBM MQ .

Questo argomento descrive il modo in cui la struttura del messaggio JMS descritta nella prima parte di questa sezione viene associata ad un messaggio IBM MQ . È di interesse per i programmatori che desiderano trasmettere messaggi tra JMS e le applicazioni IBM MQ tradizionali. È inoltre di interesse per

le persone che desiderano manipolare i messaggi trasmessi tra due applicazioni JMS , ad esempio, in una implementazione IBM Integration Bus .

Questa sezione non si applica se un'applicazione utilizza una connessione in tempo reale ad un broker. Quando un'applicazione utilizza una connessione in tempo reale, tutte le comunicazioni vengono eseguite direttamente su TCP/IP; non sono coinvolti messaggi o code IBM MQ .

I messaggi IBM MQ sono composti da tre componenti:

- MQMD (IBM MQ Message Descriptor)
- Un'intestazione IBM MQ MQRFH2
- Il contenuto del messaggio.

MQRFH2 è facoltativo e la sua inclusione in un messaggio in uscita è regolata dall'indicatore TARGCLIENT nella classe di destinazione JMS . È possibile impostare questo indicatore utilizzando lo strumento di amministrazione IBM MQ JMS . Poiché MQRFH2 contiene informazioni specifiche di JMS, includerlo sempre nel messaggio quando il mittente sa che la destinazione di ricezione è un'applicazione JMS . In genere, omettere MQRFH2 quando si invia un messaggio direttamente a un'applicazione nonJMS . Ciò si verifica perché tale applicazione non prevede un MQRFH2 nel messaggio IBM MQ .

Se un messaggio in ingresso non dispone di un'intestazione MQRFH2 , l'oggetto Coda o Argomento derivato dal campo di intestazione JMSReplyTo del messaggio, per impostazione predefinita, ha questo indicatore impostato in modo che anche un messaggio di risposta inviato alla coda o all'argomento non abbia un'intestazione MQRFH2 . È possibile disattivare questo comportamento includendo un'intestazione MQRFH2 in un messaggio di risposta solo se il messaggio originale ha un'intestazione MQRFH2 , impostando la proprietà TARGCLIENTMATCHING del factory di connessione su NO.

Figura 10 a pagina 151 mostra come la struttura di un messaggio JMS viene trasformata in un messaggio IBM MQ e di nuovo:

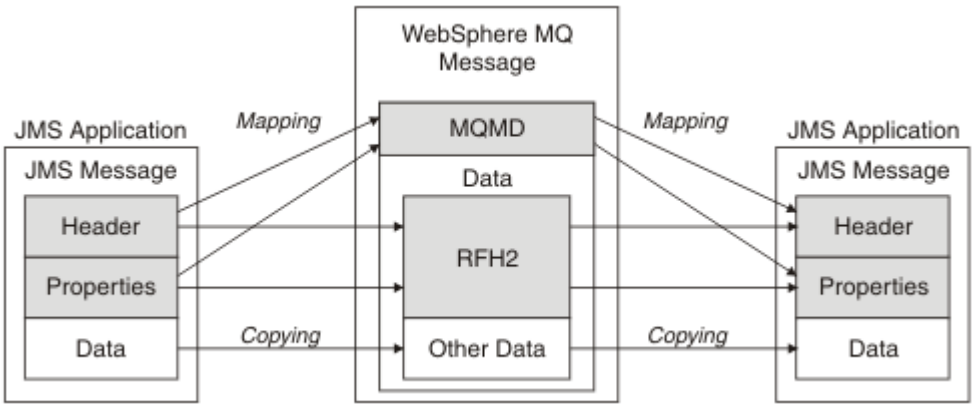


Figura 10. Come vengono trasformati i messaggi tra JMS e IBM MQ utilizzando l'intestazione MQRFH2

Le strutture si trasformano in due modi:

Associazione
Quando MQMD include un campo equivalente al campo JMS , il campo JMS viene associato al campo MQMD. Ulteriori campi MQMD sono esposti come proprietà JMS , poiché un'applicazione JMS potrebbe dover acquisire o impostare questi campi quando comunica con un'applicazione nonJMS .

In fase di copia
Se non esiste un equivalente MQMD, viene passato un campo o una proprietà di intestazione JMS , possibilmente trasformato, come un campo all'interno di MQRFH2.

L'intestazione MQRFH2 e JMS
Questa raccolta di argomenti descrive l'intestazione MQRFH Versione 2, che contiene i dati specifici di JMSassociati al contenuto del messaggio. L'intestazione MQRFH Versione 2 è estensibile e può contenere anche ulteriori informazioni non associate direttamente a JMS. Tuttavia, questa sezione riguarda solo il

suo utilizzo da parte di JMS.Per una descrizione completa, consultare MQRFH2 - Regole e intestazione di formattazione 2.

Esistono due parti dell'intestazione, una parte fissa e una parte variabile.

Parte fissa
La parte fissa è modellata sul modello di intestazione *standard* IBM MQ ed è composta dai seguenti campi:

StrucId (MQCHAR4)
Identificatore struttura.

Deve essere MQRFH_STRUC_ID (valore "RFH ") (valore iniziale).

MQRFH_STRUC_ID_ARRAY (valore: "R", "F", "H", " ") è definito anche.

Version (MQLONG)
Numero di versione della struttura.

Deve essere MQRFH_VERSION_2 (valore: 2) (valore iniziale).

StrucLength (MQLONG)
Lunghezza totale di MQRFH2, inclusi i campi di dati NameValue.

Il valore impostato in StrucLength deve essere un multiplo di 4 (i dati nei campi NameValueData possono essere riempiti con spazi).

Encoding (MQLONG)
Codifica dati.

Codifica di tutti i dati numerici nella porzione del messaggio che segue MQRFH2 (l'intestazione successiva o i dati del messaggio che seguono questa intestazione).

CodedCharSetId (MQLONG)
Coded character set identifier (CCSID)

Rappresentazione di qualsiasi dato carattere nella parte del messaggio che segue MQRFH2 (l'intestazione successiva o i dati del messaggio che seguono questa intestazione).

Formato (MQCHAR8)
Nome formato.

Nome formato per la parte del messaggio che segue MQRFH2.

Indicatori (MQLONG)
Indicatori.

MQRFH_NO_FLAGS = 0. Nessun indicatore impostato.

CCSID NameValue(MQLONG)
Il CCSID (coded character set identifier) per le stringhe di caratteri di dati NameValuecontenute in questa intestazione. I dati NameValuepossono essere codificati in una serie di caratteri diversa dalle altre stringhe di caratteri contenute nell'intestazione (StrucID e Format).

Se il CCSID di NameValueè un CCSID Unicode a 2 byte (1200, 13488 o 17584), l'ordine di byte di Unicode è lo stesso dell'ordine di byte dei campi numerici in MQRFH2. (Ad esempio, Version, StrucLengthe NameValueCCSID stesso.)

Tabella 16. Valori possibili per il campo CCSID NameValue

CCSID	**Significato**
1200	UTF-16, versione Unicode più recente supportata
13488	UTF-16, sottoinsieme Unicode versione 2.0
17584	UTF-16, sottoinsieme Unicode versione 3.0 (include il simbolo Euro)

Tabella 16. Valori possibili per il campo CCSID NameValue (Continua)	
CCSID	**Significato**
1208	UTF-8, versione Unicode più recente supportata

Parte variabile

La parte variabile segue la porzione fissa. La parte variabile contiene un numero variabile di cartelle MQRFH2 . Ogni cartella contiene un numero variabile di elementi o proprietà. Proprietà relative al gruppo di cartelle. Le intestazioni MQRFH2 create da JMS possono contenere una delle seguenti cartelle:

La cartella `mcd`

`mcd` contiene proprietà che descrivono il formato del messaggio. Ad esempio, la proprietà `Msd` del dominio del servizio di messaggi identifica un messaggio JMS come `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` o null.

La cartella `mcd` è sempre presente in un messaggio di JMS contenente un `MQRFH2`.

È sempre presente in un messaggio contenente un `MQRFH2` inviato da IBM Integration Bus. Descrive il dominio, il formato, il tipo e la serie di un messaggio.

Tabella 17. Nome, sinonimo, tipo di dati e cartella della proprietà `mcd`			
Sinonimo proprietà	**Nome proprietà**	**Tipo dati**	**Cartella**
	`mcd.Msd`	`string`	`<mcd><Msd>`*`messageDomain`*`</Msd></mcd>`
	`mcd.Set`	`string`	`<mcd><Set>`*`messageDomain`*`</Set></mcd>`
	`mcd.Type`	`string`	`<mcd><Type>`*`messageDomain`*`</Type></mcd>`
	`mcd.Fmt`	`string`	`<mcd><Fmt>`*`messageDomain`*`</Fmt></mcd>`

Non aggiungere le proprie proprietà nella cartella `mcd`.

La cartella `jms`

`jms` contiene campi di intestazione JMS e proprietà JMSX che non possono essere pienamente espresse nell'`MQMD`. La cartella `jms` è sempre presente in un `MQRFH2` JMS.

La cartella `usr`

`usr` contiene le proprietà di JMS definite dall'applicazione associate al messaggio. La cartella `usr` è presente solo se un'applicazione ha impostato una proprietà definita dall'applicazione.

La cartella `mqext`

`mqext` contiene i seguenti tipi di proprietà:

- Proprietà utilizzate solo da WebSphere Application Server.
- Proprietà relative al recapito ritardato dei messaggi.

La cartella è presente se l'applicazione ha impostato almeno una delle proprietà definite da IBM o ha utilizzato il ritardo di recapito.

Tabella 18. Nome, sinonimo, tipo di dati e cartella della proprietà mqext			
Sinonimo proprietà	**Nome proprietà**	**Tipo dati**	**Cartella**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

Non aggiungere le proprie proprietà nella cartella mqext.

La cartella mqps

mqps contiene proprietà che vengono utilizzate solo dalla pubblicazione/sottoscrizione di IBM MQ. La cartella è presente solo se l'applicazione ha impostato almeno una delle proprietà di pubblicazione/sottoscrizione integrate.

Tabella 19. Nome, sinonimo, tipo di dati e cartella della proprietà mqps			
Sinonimo proprietà	**Nome proprietà**	**Tipo dati**	**Cartella**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData...*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

Non aggiungere le proprie proprietà nella cartella mqps.

Tabella 20 a pagina 154 mostra un elenco completo di nomi di proprietà.

Tabella 20. MQRFH2 cartelle e proprietà utilizzate da JMS				
JMS nome campo	**Java tipo**	**Nome cartella MQRFH2**	**Nome proprietà**	**Tipo / valori**
JMSDestination	Destinazione	jms	Dst	stringa
JMSExpiration	lungo	jms	Esp	i8

Tabella 20. MQRFH2 cartelle e proprietà utilizzate da JMS (Continua)				
JMS nome campo	**Java tipo**	**Nome cartella MQRFH2**	**Nome proprietà**	**Tipo / valori**
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Stringa	jms	Cid	stringa
JMSReplyTo	Destinazione	jms	Rto	stringa
JMSTimestamp	lungo	jms	Tms	i8
JMSType	Stringa	mcd	Tipo, Serie, Fmt	stringa
JMSXGroupID	Stringa	jms	GID	stringa
JMSXGroupSeq	int	jms	Seq	i4
xxx (definito dall'utente)	Qualsiasi	USR	xxx	qualsiasi
		mcd	MSD	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValue(MQLONG)
Lunghezza in byte della stringa di dati NameValueche segue immediatamente questo campo di lunghezza (non include la propria lunghezza).

Dati NameValue(MQCHARn)
Una singola stringa di caratteri, la cui lunghezza in byte viene fornita dal precedente campo NameValueLength. Contiene una cartella contenente una sequenza di proprietà. Ciascuna proprietà è una tripletta nome / tipo/valore, contenuta in un elemento XML il cui nome è il nome della cartella, come segue:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

La tag `</foldername>` di chiusura può essere seguita da spazi come caratteri di riempimento. Ogni tripletta è codificata utilizzando una sintassi simile a XML:

```
<name dt='datatype'>value</name>
```

L'elemento `dt='datatype'` è facoltativo e viene omesso per molte proprietà, poiché il tipo di dati è predefinito. Se è incluso, uno o più caratteri spazio devono essere inclusi prima del tag `dt=` .

`name`
è il nome della proprietà; consultare Tabella 20 a pagina 154.

`datatype`
deve corrispondere, dopo il raggruppamento, ad uno dei tipi di dati elencati in Tabella 21 a pagina 156.

`value`
è una rappresentazione stringa del valore da trasmettere, utilizzando le definizioni in Tabella 21 a pagina 156.

Un valore null viene codificato utilizzando la sintassi seguente:

```
<name dt='datatype' xsi:nil='true'></name>
```

Non utilizzare `xsi:nil='false'`.

Tabella 21. Tipi di dati proprietà

Tipo dati	**Definizione**
stringa	Qualsiasi sequenza di caratteri esclusi < e &
booleano	Il carattere `0` o `1` (`0` = false, `1` = true)
bin.hex	Si tratta di cifre esadecimali che rappresentano ottetti.
i1	Un numero, espresso con cifre `0..9`, con segno facoltativo (nessuna frazione o esponente). Deve essere compreso nell'intervallo tra -128 e 127 inclusi
i2	Un numero, espresso con cifre `0..9`, con segno facoltativo (nessuna frazione o esponente). Deve essere compreso nell'intervallo tra -32768 e 32767 incluso
i4	Un numero, espresso con cifre `0..9`, con segno facoltativo (nessuna frazione o esponente). Deve essere compreso tra -2147483648 e 2147483647 inclusi
i8	Un numero, espresso con cifre `0..9`, con segno facoltativo (nessuna frazione o esponente). Deve essere compreso tra -9223372036854775808 e 9223372036854775807 inclusi
int	Un numero, espresso con cifre `0..9`, con segno facoltativo (nessuna frazione o esponente). Deve essere nello stesso intervallo di i8. Può essere utilizzato al posto di uno dei tipi i * se il mittente non desidera associare una particolare precisione alla proprietà
r4	Numero a virgola mobile, grandezza < = 3.40282347E+38,> = 1.175E-37 espresso con cifre `0..9`, segno facoltativo, cifre frazionarie facoltative, esponente facoltativo
r8	Numero a virgola mobile, magnitudine < = 1.7976931348623E+308,> = 2.225E-307 espresso con cifre `0..9`, segno facoltativo, cifre frazionarie facoltative, esponente facoltativo

Un valore stringa può contenere spazi. È necessario utilizzare le seguenti sequenze di escape in un valore stringa:

- `&` per il carattere &
- `<` per il carattere <

È possibile utilizzare le seguenti sequenze di escape, ma non sono richieste:

- `>` per il carattere >
- `'` per il carattere '
- `"` per il carattere "

Campi e proprietà JMS con campi MQMD corrispondenti
Queste tabelle mostrano campi MQMD equivalenti a campi di intestazione JMS , proprietà JMS e proprietà specifiche del fornitore JMS .

Tabella 22 a pagina 157 elenca i JMS campi intestazione e Tabella 23 a pagina 157 elenca le proprietà JMS associate direttamente ai campi MQMD. Tabella 24 a pagina 157 elenca le proprietà specifiche del provider e i campi MQMD a cui sono associati.

Tabella 22. Campi di intestazione JMS associati ai campi MQMD			
Campo di intestazione JMS	**Java tipo**	**Campo MQMD**	**Tipo C**
JMSDeliveryMode	int	Persistenza	MQLONG
JMSExpiration	lungo	Scadenza	MQLONG
JMSPriority	int	Priorità	MQLONG
JMSMessageID	Stringa	MsgID	MQBYTE24
JMSTimestamp	lungo	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Stringa	CorrelId	MQBYTE24

Tabella 23. Proprietà JMS associate ai campi MQMD			
JMS proprietà	**Java tipo**	**Campo MQMD**	**Tipo C**
JMSXUserID	Stringa	UserIdentifier	MQCHAR12
JMSXAppID	Stringa	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	Stringa	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tabella 24. Proprietà specifiche del provider JMS associate ai campi MQMD			
proprietà specifica del provider JMS	**Java tipo**	**Campo MQMD**	**Tipo C**
Eccezione JMS_IBM_Report_Exception	int	Prospetto	MQLONG
Scadenza report IBM JMS	int	Prospetto	MQLONG
COA report IBM JMS	int	Prospetto	MQLONG
COD report IBM JMS	int	Prospetto	MQLONG
PAN report IBM JMS	int	Prospetto	MQLONG
NAN report IBM JMS	int	Prospetto	MQLONG
ID_ms_ms_IBM_Report_Pass_Msg_ID	int	Prospetto	MQLONG
ID correlazione password report IBM JMS	int	Prospetto	MQLONG
JMS_IBM_Report_Discard_Msg	int	Prospetto	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
Feedback IBM JMS	int	Feedback	MQLONG
Formato_IBM_JMS	Stringa	Formato "1" a pagina 158	MQCHAR8
JMS_IBM_PutApplTipo	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Codifica	MQLONG

Tabella 24. Proprietà specifiche del provider JMS associate ai campi MQMD (Continua)			
proprietà specifica del provider JMS	**Java tipo**	**Campo MQMD**	**Tipo C**
Serie di caratteristiche IBM JMS	Stringa	CodedCharacterSetId "2" a pagina 158	MQLONG
JMS_IBM_PutDate	Stringa	PutDate	MQCHAR8
JMS_IBM_PutTime	Stringa	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	booleano	MsgFlags	MQLONG

Nota:

1. JMS_IBM_Format rappresenta il formato del corpo del messaggio. Questo valore può essere definito dall'applicazione che imposta la proprietà JMS_IBM_Format del messaggio (notare che è presente un limite di 8 caratteri) o può essere impostato sul formato IBM MQ del corpo del messaggio appropriato al tipo di messaggio JMS . JMS_IBM_Format si associa al campo Formato MQMD solo se il messaggio non contiene sezioni RFH o RFH2 . In un messaggio tipico, si associa al campo Formato di RFH2 immediatamente precedente il corpo del messaggio.
2. Il valore della proprietà JMS_IBM_Character_Set è un valore stringa che contiene l'equivalente della serie di caratteri Java per il valore numerico CodedCharacterSetId . Il campo MQMD CodedCharacterSetId è un valore numerico che contiene l'equivalente della stringa della serie di caratteri Java specificata dalla proprietà JMS_IBM_Character_Set.

Associazione di campi JMS su campi IBM MQ (messaggi in uscita)
Queste tabelle mostrano il modo in cui i campi di intestazione e proprietà JMS vengono mappati nei campi MQMD e MQRFH2 al momento di send () o publish ().

Tabella 25 a pagina 158 mostra in che modo i campi di intestazione JMS vengono associati nei campi MQMD/RFH2 al momento di send () o publish (). Tabella 26 a pagina 159 mostra come le proprietà JMS vengono associate nei campi MQMD/RFH2 al momento di send () o publish (). Tabella 27 a pagina 160 mostra in che modo le proprietà specifiche del provider JMS vengono associate ai campi MQMD al momento di send () o publish (),

Per i campi contrassegnati da Oggetto messaggio, il valore trasmesso è il valore contenuto nel messaggio JMS immediatamente prima dell'operazione send () o publish (). Il valore nel messaggio JMS viene lasciato invariato dall'operazione.

Per i campi contrassegnati come impostati dal metodo di invio, viene assegnato un valore quando si esegue send () o publish () (qualsiasi valore contenuto nel messaggio JMS viene ignorato). Il valore nel messaggio JMS viene aggiornato per mostrare il valore utilizzato.

I campi contrassegnati come Solo ricezione non vengono trasmessi e vengono lasciati invariati nel messaggio da send () o publish ().

Tabella 25. Mappatura del campo del messaggio in uscita			
Nome campo intestazione JMS	**Campo MQMD utilizzato per la trasmissione**	**Intestazione**	**Impostato da**
JMSDestination		MQRFH2	Metodo di invio
JMSDeliveryMode	Persistenza	MQRFH2	Metodo di invio
JMSExpiration	Scadenza	MQRFH2	Metodo di invio
JMSPriority	Priorità	MQRFH2	Metodo di invio
JMSMessageID	MsgID		Metodo di invio

Tabella 25. Mappatura del campo del messaggio in uscita (Continua)			
Nome campo intestazione JMS	**Campo MQMD utilizzato per la trasmissione**	**Intestazione**	**Impostato da**
JMSTimestamp	PutDate/PutTime		Metodo di invio
JMSCorrelationID	CorrelId	MQRFH2	Oggetto messaggio
JMSReplyTo	Gestore code ReplyToQ/ ReplyTo	MQRFH2	Oggetto messaggio
JMSType		MQRFH2	Oggetto messaggio
JMSRedelivered			Solo ricezione

Nota:

1. Il campo MQMD CodedCharacterSetId è un valore numerico che contiene l'equivalente della stringa della serie di caratteri Java specificata dalla proprietà JMS_IBM_Character_Set.

Tabella 26. Mappatura della proprietà JMS del messaggio in uscita			
JMS nome proprietà.	**Campo MQMD utilizzato per la trasmissione**	**Intestazione**	**Impostato da**
JMSXUserID	UserIdentifier		Metodo di invio
JMSXAppID	PutApplName		Metodo di invio
JMSXDeliveryCount			Solo ricezione
JMSXGroupID	GroupId	MQRFH2	Oggetto messaggio
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Oggetto messaggio

Nota:

Queste proprietà sono definite come di sola lettura dalla specifica JMS e sono impostate (in alcuni casi facoltativamente) dal fornitore JMS.

In IBM MQ classes for JMS due di queste proprietà possono essere sovrascritte dall'applicazione. A tale scopo, verificare che la destinazione sia stata configurata in modo appropriato impostando le seguenti proprietà:

1. Impostare la proprietà `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` su `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`.
2. Impostare la proprietà `WMQConstants.WMQ_MQMD_WRITE_ENABLED` su `true`.

Le seguenti proprietà possono essere sovrascritte dall'applicazione:

JMSXAppID
Questa proprietà può essere sovrascritta impostando la proprietà `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` sul messaggio - il valore deve essere una stringa Java .

JMSXGroupID
Questa proprietà può essere sovrascritta impostando la proprietà `WMQConstants.JMS_IBM_MQMD_GROUPID` sul messaggio - il valore deve essere un array di byte.

Tabella 27. Mappatura proprietà specifica del provider JMS del messaggio in uscita			
Nome della proprietà specifica del fornitore JMS	**Campo MQMD utilizzato per la trasmissione**	**Intestazione**	**Impostato da**
Eccezione JMS_IBM_Report_Exception	Prospetto		Oggetto messaggio
Scadenza report IBM JMS	Prospetto		Oggetto messaggio
COA/COD report IBM JMS	Prospetto		Oggetto messaggio
NAN/PAN report IBM JMS	Prospetto		Oggetto messaggio
ID_ms_ms_IBM_Report_Pass_Msg_ID	Prospetto		Oggetto messaggio
ID correlazione password report IBM JMS	Prospetto		Oggetto messaggio
JMS_IBM_Report_Discard_Msg	Prospetto		Oggetto messaggio
JMS_IBM_MsgType	MsgType		Oggetto messaggio
Feedback IBM JMS	Feedback		Oggetto messaggio
Formato_IBM_JMS	Formato		Oggetto messaggio
JMS_IBM_PutApplTipo	PutApplType		Metodo di invio
JMS_IBM_Encoding	Codifica		Oggetto messaggio
Serie di caratteristiche IBM JMS	CodedCharacterSetId		Oggetto messaggio
JMS_IBM_PutDate	PutDate		Metodo di invio
JMS_IBM_PutTime	PutTime		Metodo di invio
JMS_IBM_Last_Msg_In_Group	MsgFlags		Oggetto messaggio

Associazione dei campi di intestazione JMS a send () o publish ()
Queste note si riferiscono all'associazione dei campi JMS in send () o publish ().

JMSDestination a MQRFH2
Viene memorizzata come una stringa che serializza le caratteristiche salienti dell'oggetto di destinazione in modo che un JMS di ricezione possa ricostituire un oggetto di destinazione equivalente. Il campo MQRFH2 è codificato come URI (consultare "URI (Uniform Resource Identifier)" a pagina 225 per i dettagli della notazione URI).

Da JMSReplyTo a MQMD.ReplyToQ, ReplyTo, MQRFH2
Il nome della coda viene copiato in MQMD.ReplyToQ e il nome del gestore code viene copiato nei campi del gestore code ReplyTo. Le informazioni sull'estensione di destinazione (altri dettagli utili conservati nell'oggetto di destinazione) vengono copiati nel campo MQRFH2 . Il campo MQRFH2 è codificato come URI (consultare "URI (Uniform Resource Identifier)" a pagina 225 per i dettagli della notazione URI).

JMSDeliveryMode su MQMD.Persistence

Il valore JMSDeliveryMode viene impostato dal metodo send () o publish () o MessageProducer, a meno che l'oggetto di destinazione non lo sovrascriva. Il valore JMSDeliveryMode è associato a MQMD.Persistence come segue:

- Il valore JMS PERSISTENT è equivalente a MQPER_PERSISTENT
- Il valore JMS NON_PERSISTENT è equivalente a MQPER_NOT_PERSISTENT

Se la proprietà di persistenza di MQQueue non è impostata su WMQConstants.WMQ_PER_QDEF, il valore della modalità di consegna è codificato anche in MQRFH2.

JMSExpiration a/da MQMD.Expiry, MQRFH2

JMSExpiration memorizza l'ora di scadenza (la somma dell'ora corrente e dell'ora di scadenza), mentre MQMD memorizza l'ora di scadenza. Inoltre, JMSExpiration è espresso in millisecondi, ma MQMD.Expiry è in decimi di secondo.

- Se il metodo send () imposta una durata illimitata, MQMD.Expiry è impostata su MQEI_UNLIMITED e nessun JMSExpiration è codificato in MQRFH2.
- Se il metodo send () imposta una durata inferiore a 214748364.7 secondi (circa 7 anni), la durata viene memorizzata in MQMD.Expirye l'ora di scadenza (in millisecondi) vengono codificati come un valore i8 in MQRFH2.
- Se il metodo send () imposta una durata maggiore di 214748364.7 secondi, MQMD.Expiry è impostata su MQEI_UNLIMITED. Il tempo di scadenza reale in millisecondi è codificato come un valore i8 in MQRFH2.

JMSPriority per MQMD.Priority

Associare direttamente il valore JMSPriority (0-9) al valore della priorità MQMD (0-9). Se JMSPriority è impostato su un valore non predefinito, anche il livello di priorità viene codificato in MQRFH2.

JMSMessageID da MQMD.MessageID

Tutti i messaggi inviati da JMS hanno identificatori di messaggi univoci assegnati da IBM MQ. Il valore assegnato viene restituito in MQMD.MessageId dopo la chiamata MQPUT e viene passato nuovamente all'applicazione nel campo JMSMessageID . IBM MQ messageId è un valore binario a 24 byte, mentre JMSMessageID è una stringa. Il JMSMessageID è composto dal valore binario messageId convertito in una sequenza di 48 caratteri esadecimali, preceduti dal carattere ID:. JMS fornisce un suggerimento che può essere impostato per disabilitare la produzione di identificativi messaggio. Questo suggerimento viene ignorato e viene assegnato un identificativo univoco in tutti i casi. Qualsiasi valore impostato nel campo JMSMessageID prima di send () viene sovrascritto.

Se si richiede la capacità di specificare l'MQMD MQMD.MessageID, puoi farlo con una delle estensioni IBM MQ JMS descritte in "Lettura e scrittura del descrittore del messaggio da una applicazione IBM MQ classes for JMS" a pagina 247.

JMSTimestamp su MQRFH2

Durante un invio, il campo JMSTimestamp viene impostato in base all'orologio della JVM. Questo valore è impostato in MQRFH2. Qualsiasi valore impostato nel campo JMSTimestamp prima di send () viene sovrascritto. Consultare anche le proprietà JMS_IBM_PutDate e JMS_IBMPutTime .

Da JMSType a MQRFH2

Questa stringa è impostata nel campo MQRFH2 mcd.Type . Se è in formato URI, può interessare anche i campi mcd.Set e mcd.Fmt .

JMSCorrelationID in MQMD.CorrelId, MQRFH2

Il JMSCorrelationID può contenere uno dei seguenti:

Un ID messaggio specifico del fornitore

Si tratta di un identificativo di messaggio da un messaggio precedentemente inviato o ricevuto, e quindi dovrebbe essere una stringa di 48 cifre esadecimali minuscole con prefisso `ID:` Il prefisso viene rimosso, i restanti caratteri vengono convertiti in binario e quindi impostati in MQMD.CorrelId .

Un valore byte nativo del provider []

Il valore viene copiato in MQMD.CorrelId field - riempito con valori null o troncato a 24 byte, se necessario. Nessun valore CorrelId è codificato in MQRFH2.

Una stringa specifica dell'applicazione
Il valore viene copiato in MQRFH2. I primi 24 byte della stringa, in formato UTF8 , vengono scritti nell'MQMD MQMD.CorrelID.

Associazione dei campi della proprietà JMS
Queste note fanno riferimento all'associazione dei campi delle proprietà JMS nei messaggi IBM MQ .

JMSXUserID da MQMD UserIdentifier
JMSXUserID è impostato al ritorno dalla chiamata di invio.

JMSXAppID dal nome MQMD PutAppl
JSMXAppID è impostato al ritorno dalla chiamata di invio.

JMSXGroupID a MQRFH2 (point - to - point)
Per i messaggi point-to-point, JMSXGroupID viene copiato nel campo MQMD GroupID . Se JMSXGroupID inizia con l'ID prefisso:, viene convertito in binario. Altrimenti, viene codificato come una stringa UTF8 . Il valore viene riempito o troncato se necessario ad una lunghezza di 24 byte. L'indicatore MQMF_MSG_IN_GROUP è impostato.

JMSXGroupID a MQRFH2 (pubblicazione / sottoscrizione)
Per i messaggi di pubblicazione / sottoscrizione, JMSXGroupID viene copiato in MQRFH2 come stringa.

JMSXGroupSeq MQMD MsgSeqNumero (point - to - point)
Per i messaggi point-to-point, JMSXGroupSeq viene copiato nel campo MQMD MsgSeqNumber. L'indicatore MQMF_MSG_IN_GROUP è impostato.

JMSXGroupSeq MQMD MsgSeqNumero (pubblicazione/sottoscrizione)
Per i messaggi di pubblicazione / sottoscrizione, JMSXGroupSeq viene copiato in MQRFH2 come i4.

Associazione di campi specifici del fornitore JMS
Le seguenti note fanno riferimento all'associazione dei campi specifici del provider JMS nei messaggi IBM MQ .

Report da JMS_IBM_Report_*XXX* a MQMD
Un'applicazione JMS può impostare le opzioni del report MQMD utilizzando le seguenti proprietà JMS_IBM_Report_*XXX* . Il singolo MQMD è associato a diverse proprietà JMS_IBM_Report_*XXX* .

Le costanti JMS_IBM_Report_*XXX* si trovano in `com.ibm.msg.client.jakarta.wmq.WMQConstants` o `com.ibm.msg.client.wmq.WMQConstants`.

Eccezione JMS_IBM_Report_Exception

MQRO_EXCEPTION o
MQRO_EXCEPTION_WITH_DATA o
MQRO_EXCEPTION_WITH_FULL_DATA

Scadenza report IBM JMS

MQRO_EXPIRATION o
MQRO_EXPIRATION_WITH_DATA o
MQRO_EXPIRATION_WITH_FULL_DATA

COA report IBM JMS

MQRO_COA o
MQRO_COA_WITH_DATA o
DATI_COA_WITH_MQRO_FULL_DATA

COD report IBM JMS

MQRO_COD o
MQRO_COD_WITH_DATA o
DAD_COD MQRO_WITH_FULL_DATA

PAN report IBM JMS
MQRO_PAN

NAN report IBM JMS
MQRO_NAN

ID_ms_ms_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

ID correlazione password report IBM JMS
ID CORREL_PASS_MQRO_

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

I valori MQRO sono in `com.ibm.mq.constants.CMQC`.

JMS_IBM_MsgType in MQMD MsgType
Il valore viene associato direttamente a MQMD MsgType. Se l'applicazione non ha impostato un valore esplicito di JMS_IBM_MsgType, viene utilizzato un valore predefinito. Questo valore predefinito è determinato come segue:

- Se JMSReplyTo è impostato su una destinazione coda IBM MQ , MSGType è impostato sul valore MQMT_REQUEST
- Se JMSReplyTo non è impostato o è impostato su un valore diverso da una destinazione coda IBM MQ , MsgType è impostato sul valore MQMT_DATAGRAM

Feedback da JMS_IBM_Feedback a MQMD
Il valore viene associato direttamente a MQMD Feedback.

JMS_IBM_Format in formato MQMD
Il valore si associa direttamente al formato MQMD.

Da JMS_IBM_Encoding a MQMD Encoding
Se impostata, questa proprietà sovrascrive la codifica numerica della coda di destinazione o dell'argomento.

JMS_IBM_Character_Set su MQMD CodedCharacterSetId
Se impostata, questa proprietà sovrascrive la proprietà della serie di caratteri codificati della coda di destinazione o dell'argomento.

JMS_IBM_PutDate da MQMD PutDate
Il valore di questa proprietà viene impostato, durante l'invio, direttamente dal campo PutDate in MQMD. Qualsiasi valore impostato nella proprietà JMS_IBM_PutDate prima di un invio viene sovrascritto. Questo campo è una stringa di otto caratteri, nel formato data IBM MQ AAAAMMGG. Questa proprietà può essere utilizzata con la proprietà JMS_IBMPutTime per determinare l'ora in cui il messaggio è stato inserito in base al gestore code.

JMS_IBM_PutTime da MQMD PutTime
Il valore di questa proprietà viene impostato, durante l'invio, direttamente dal campo PutTime in MQMD. Qualsiasi valore impostato nella proprietà JMS_IBM_PutTime prima di un invio viene sovrascritto. Questo campo è una stringa di otto caratteri, nel formato ora IBM MQ di HHMMSSTH. Questa proprietà può essere utilizzata con la proprietà JMS_IBMPutDate per determinare l'ora in cui il messaggio è stato inserito in base al gestore code.

JMS_IBM_Last_Msg_In_Group a MQMD MsgFlags
Per la messaggistica point - to - point, questo valore booleano si associa all'indicatore MQMF_LAST_MSG_IN_GROUP nel campo MQMD MsgFlags . Di solito viene utilizzato con le proprietà JMSXGroupID e JMSXGroupSeq per indicare a un'applicazione IBM MQ legacy che questo messaggio è l'ultimo di un gruppo. Questa proprietà viene ignorata per la messaggistica di pubblicazione / sottoscrizione.

Mappatura di campi IBM MQ su campi JMS (messaggi in entrata)
Queste tabelle mostrano in che modo i campi di intestazione e proprietà JMS vengono mappati nei campi MQMD e MQRFH2 al momento di get () o receive ().

Tabella 28 a pagina 164 mostra come JMS i campi di intestazione vengono associati ai campi MQMD/MQRFH2 al momento di get () o receive (). La Tabella 29 a pagina 164 mostra come JMS i campi di proprietà sono associati ai campi MQMD/MQRFH2 al momento di get () o receive (). Tabella 30 a pagina 165 mostra come vengono associate le proprietà specifiche del fornitore JMS .

Tabella 28. Mappatura del campo di intestazione JMS del messaggio in entrata

Nome campo intestazione JMS	**Campo MQMD richiamato da**	**Campo MQRFH2 richiamato da**
JMSDestination		jms.Dst o mqps.Top "1" a pagina 164
JMSDeliveryMode	Persistenza"2" a pagina 164	jms.Dlv "2" a pagina 164
JMSExpiration		jms.Exp
JMSPriority	Priorità	
JMSMessageID	MsgID	
JMSTimestamp	PutDate "2" a pagina 164 PutTime "2" a pagina 164	jms.Tms "2" a pagina 164
JMSCorrelationID	CorrelId "2" a pagina 164	jms.Cid "2" a pagina 164
JMSReplyTo	ReplyToQ "2" a pagina 164 ReplyToGestore code "2" a pagina 164	jms.Rto "2" a pagina 164
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Nota:

1. Se sono impostati sia jms.Dst che mqps.Top , viene utilizzato il valore in jms.Dst .
2. Per le proprietà che possono avere valori richiamati da MQRFH2 o da MQMD, se entrambi sono disponibili, viene utilizzata l'impostazione in MQRFH2 .
3. Il valore della proprietà JMS_IBM_Character_Set è un valore stringa che contiene l'equivalente della serie di caratteri Java per il valore numerico CodedCharacterSetId .

Tabella 29. Mappatura delle proprietà dei messaggi in entrata

JMS nome proprietà.	**Campo MQMD richiamato da**	**Campo MQRFH2 richiamato da**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId "1" a pagina 164	jms.Gid "1" a pagina 164
JMSXGroupSeq	MsgSeqNumero "1" a pagina 164	jms.Seq "1" a pagina 164

Nota:

1. Per le proprietà che possono avere valori richiamati da MQRFH2 o da MQMD, se entrambi sono disponibili, viene utilizzata l'impostazione in MQRFH2 . Le proprietà vengono impostate dai valori MQMD solo se sono impostati gli indicatori di messaggio MQMF_MSG_IN_GROUP o MQMF_LAST_MSG_IN_GROUP.

Tabella 30. Mappatura della proprietà JMS specifica del fornitore di messaggi in entrata		
JMS nome proprietà.	**Campo MQMD richiamato da**	**Campo MQRFH2 richiamato da**
Eccezione JMS_IBM_Report_Exception	Prospetto	
Scadenza report IBM JMS	Prospetto	
COA report IBM JMS	Prospetto	
COD report IBM JMS	Prospetto	
PAN report IBM JMS	Prospetto	
NAN report IBM JMS	Prospetto	
JMS_IBM_Report_ Pass_Msg_ID	Prospetto	
ID correlazione password report IBM JMS	Prospetto	
JMS_IBM_Report_Discard_Msg	Prospetto	
JMS_IBM_MsgType	MsgType	
Feedback IBM JMS	Feedback	
Formato_IBM_JMS	Formato	
JMS_IBM_PutApplTipo	PutApplType	
Codifica IBM JMS "1" a pagina 165	Codifica	
Serie di caratteristiche IBM JMS "1" a pagina 165	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Impostato solo se il messaggio in entrata è un messaggio in byte.

Scambio di messaggi tra un'applicazione JMS e un'applicazione IBM MQ tradizionale
Questo argomento descrive cosa accade quando un'applicazione JMS scambia i messaggi con un'applicazione IBM MQ tradizionale che non può elaborare l'intestazione MQRFH2 .

Figura 11 a pagina 166 mostra l'associazione.

L'amministratore indica che l'applicazione JMS sta comunicando con un'applicazione IBM MQ tradizionale impostando la proprietà TARGCLIENT della destinazione su *MQ*. Ciò indica che non deve essere prodotta alcuna intestazione MQRFH2 . Se questa operazione non viene eseguita, l'applicazione ricevente deve essere in grado di gestire l'intestazione MQRFH2 .

L'associazione da JMS a MQMD indirizzata a un'applicazione IBM MQ tradizionale è uguale all'associazione da JMS a MQMD indirizzata a un'applicazione JMS . Se IBM MQ classes for JMS riceve un messaggio IBM MQ con il campo MQMD *Format* impostato su un valore diverso da MQFMT_RFH2, i dati vengono ricevuti da un'applicazione nonJMS . Se il formato è MQFMT_STRING, il messaggio viene ricevuto come messaggio di testo JMS . Altrimenti, viene ricevuto come un messaggio di JMS byte. Poiché non esiste alcun MQRFH2, solo le proprietà JMS trasmesse in MQMD possono essere ripristinate.

Se IBM MQ classes for JMS riceve un messaggio che non dispone di un'intestazione MQRFH2 , la proprietà TARGCLIENT dell'oggetto Coda o Argomento derivato dal campo di intestazione JMSReplyTo del messaggio è impostata su MQ per impostazione predefinita. Ciò significa che anche un messaggio di

risposta inviato alla coda o all'argomento non dispone di un'intestazione MQRFH2 . È possibile disattivare questo comportamento includendo un'intestazione MQRFH2 in un messaggio di risposta solo se il messaggio originale ha un'intestazione MQRFH2 , impostando la proprietà TARGCLIENTMATCHING del factory di connessione su NO.

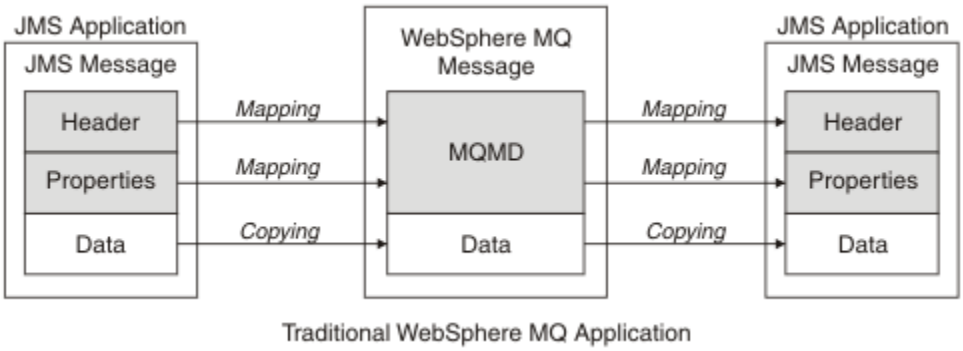


Figura 11. Come i messaggi JMS vengono trasformati in messaggi IBM MQ senza intestazione MQRFH2

Il corpo del messaggio JMS
Questo argomento contiene informazioni sulla codifica del corpo del messaggio stesso. La codifica dipende dal tipo di messaggio JMS .

ObjectMessage
Un ObjectMessage è un oggetto serializzato da Java Runtime nel modo normale.

TextMessage
Un TextMessage è una stringa codificata. Per un messaggio in uscita, la stringa è codificata nella serie di caratteri fornita dall'oggetto di destinazione. Il valore predefinito è la codifica UTF8 (la codificazione UTF8 inizia con il primo carattere del messaggio; non esiste alcun campo di lunghezza all'inizio). Tuttavia, è possibile specificare qualsiasi altra serie di caratteri supportata da IBM MQ classes for JMS. Tali serie di caratteri vengono utilizzate principalmente quando si invia un messaggio a un'applicazione nonJMS .

Se la serie di caratteri è una serie a doppio byte (incluso UTF16), la specifica di codifica del numero intero dell'oggetto di destinazione determina l'ordine dei byte.

Un messaggio in entrata viene interpretato utilizzando la serie di caratteri e la codifica specificati nel messaggio stesso. Queste specifiche si trovano nell'ultima intestazione IBM MQ (o MQMD se non ci sono intestazioni). Per i messaggi JMS , l'ultima intestazione è di solito MQRFH2.

BytesMessage
Un BytesMessage è, per impostazione predefinita, una sequenza di byte come definito nella specifica JMS 1.0.2 e nella documentazione Java associata.

Per un messaggio in uscita assemblato dall'applicazione stessa, la proprietà di codifica dell'oggetto di destinazione può essere utilizzata per sovrascrivere le codifiche dei campi integer e floating point contenuti nel messaggio. Ad esempio, è possibile richiedere che i valori a virgola mobile siano memorizzati in formato S/390 anziché IEEE).

Un messaggio in entrata viene interpretato utilizzando la codifica numerica specificata nel messaggio stesso. Questa specifica si trova nell'ultima intestazione IBM MQ (o MQMD se non ci sono intestazioni). Per i messaggi JMS , l'ultima intestazione è di solito MQRFH2.

Se viene ricevuto un BytesMessage e viene inviato nuovamente senza modifiche, il relativo corpo viene trasmesso byte per byte, come è stato ricevuto. La proprietà di codifica dell'oggetto di destinazione non ha alcun effetto sul corpo. L'unica entità di tipo stringa che può essere inviata esplicitamente in un BytesMessage è una stringa UTF8 . Viene codificato in formato Java UTF8 e inizia con un campo di lunghezza di 2 byte. La proprietà della serie di caratteri dell'oggetto di destinazione non ha alcun effetto sulla codifica di un BytesMessagein uscita. Il valore della serie di caratteri in un messaggio IBM MQ in entrata non ha alcun effetto sull'interpretazione di tale messaggio come JMS BytesMessage.

Le applicazioni nonJava non sono in grado di riconoscere la codifica Java UTF8 . Pertanto, affinché un'applicazione JMS invii un BytesMessage che contiene dati di testo, l'applicazione stessa deve convertire le proprie stringhe in array di byte e scrivere tali array di byte in BytesMessage.

MapMessage
Un MapMessage è una stringa contenente triplette nome / tipo/valore XML codificate come:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

dove `datatype` è uno dei tipi di dati elencati in Tabella 21 a pagina 156. Il tipo di dati predefinito è `string`e quindi l'attributo `dt="string"` viene omesso per gli elementi stringa.

La serie di caratteri utilizzata per codificare o interpretare la stringa XML che forma il corpo di un messaggio di associazione è determinata in base alle regole che si applicano a un messaggio di testo.

Le versioni di IBM MQ classes for JMS precedenti a 5.3 codificano il contenuto di un messaggio di mappa nel seguente formato:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

IBM MQ classes for JMS 5.3 e versioni successive possono interpretare entrambi i formati, ma le versioni di IBM MQ classes for JMS precedenti a 5.3 non possono interpretare il formato corrente.

Se un'applicazione deve inviare messaggi di associazione a un'altra applicazione che utilizza una versione di IBM MQ classes for JMS precedente a 5.3, l'applicazione di invio deve chiamare il metodo di produzione connessioni `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)` per specificare che i messaggi di associazione vengono inviati nel formato precedente. Per impostazione predefinita, tutti i messaggi di associazione vengono inviati nel formato corrente.

StreamMessage
Un StreamMessage è come un messaggio di associazione, ma senza nomi elemento:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

dove `datatype` è uno dei tipi di dati elencati in Tabella 21 a pagina 156. Il tipo di dati predefinito è `string`e quindi l'attributo `dt="string"` viene omesso per gli elementi stringa.

La serie di caratteri utilizzata per codificare o interpretare la stringa XML che costituisce il corpo di StreamMessage viene determinata in base alle regole che si applicano a TextMessage.

Il campo MQRFH2.format è impostato come segue:

MQFMT_NONE
per ObjectMessage, BytesMessageo messaggi senza corpo.

MQFMT_STRING
per TextMessage, StreamMessageo MapMessage.

JMS Conversione del messaggio
La conversione dei dati del messaggio in JMS viene eseguita durante l'invio e la ricezione dei messaggi. IBM MQ esegue automaticamente la maggior parte della conversione dei dati. Converte il testo e i dati

numerici durante il trasferimento di un messaggio tra applicazioni JMS . Il testo viene convertito quando si scambiano JMSTextMessage tra un'applicazione JMS e un'applicazione IBM MQ .

Se si sta pianificando di effettuare scambi di messaggi più complessi, i seguenti argomenti sono di interesse. Gli scambi di messaggi complessi includono:

- Trasferimento di messaggi non di testo tra un'applicazione IBM MQ e un'applicazione JMS .
- Scambio di dati di testo in formato byte.
- Conversione del testo nella tua applicazione.

JMS Dati del messaggio

La conversione dei dati è necessaria per scambiare dati di testo e numerici tra le applicazioni, anche tra due applicazioni JMS . La rappresentazione interna del testo e dei numeri deve essere codificata in modo che possano essere trasferiti in un messaggio. La codifica forza una decisione su come vengono rappresentati i numeri e il testo. IBM MQ gestisce la codifica di testo e numeri nei messaggi JMS , ad eccezione di JMSObjectMessage, consultare “JMSObjectMessage” a pagina 174. Utilizza tre attributi del messaggio. I tre attributi sono CodedCharacterSetId, Encodinge Format.

Questi tre attributi del messaggio sono normalmente memorizzati nei campi di intestazione JMS , MQRFH2, di un messaggio JMS . Se il tipo di messaggio è un tipo di messaggio MQ, piuttosto che JMS , gli attributi vengono memorizzati nel descrittore del messaggio, MQMD. Gli attributi vengono utilizzati per convertire i dati del messaggio JMS . I dati del messaggio JMS vengono trasferiti nella parte dei dati del messaggio di un messaggio IBM MQ .

JMS Proprietà messaggio

Le proprietà del messaggio JMS , come JMS_IBM_CHARACTER_SET, vengono scambiate nella parte di intestazione MQRFH2 di un messaggio JMS , a meno che il messaggio non sia stato inviato senza un MQRFH2. Solo JMSTextMessage e JMSBytesMessage possono essere inviati senza MQRFH2. Se una proprietà JMS viene memorizzata come una proprietà del messaggio IBM MQ nel descrittore del messaggio, MQMD, viene convertita come parte della conversione MQMD . Se una proprietà JMS è memorizzata in MQRFH2, viene memorizzata nella serie di caratteri specificata da MQRFH2.NameValueCCSID. Quando un messaggio viene inviato o ricevuto, le proprietà del messaggio vengono convertite nella relativa rappresentazione interna nella JVM. La conversione è da e verso la serie di caratteri del descrittore del messaggio o MQRFH2.NameValueCCSID. I dati numerici vengono convertiti in testo.

JMS Conversione del messaggio

I seguenti argomenti contengono esempi e attività utili se si prevede di scambiare messaggi più complessi che richiedono la conversione.

Approcci di conversione dei messaggi JMS
Una serie di approcci di conversione dati sono aperti ai progettisti di applicazioni JMS . Questi approcci non sono esclusivi; alcune applicazioni probabilmente utilizzeranno una combinazione di questi approcci. Se l'applicazione sta scambiando solo testo o messaggi solo con altre applicazioni JMS , normalmente non si considera la conversione dei dati. La conversione dei dati viene eseguita automaticamente da IBM MQ.

È possibile porre una serie di domande su come avvicinarsi alla conversione dei messaggi:

È necessario pensare alla conversione dei messaggi?
In alcuni casi, come ad esempio i trasferimenti di messaggi da JMS a JMS e lo scambio di messaggi di testo con programmi IBM MQ , IBM MQ esegue automaticamente le conversioni necessarie. È possibile che si desideri controllare la conversione dei dati per motivi di prestazioni o che si stia scambiando messaggi complessi con un formato predefinito. In casi come questi, è necessario comprendere la conversione dei messaggi e leggere i seguenti argomenti.

Che tipo di conversione ci sono?

Ci sono quattro tipi principali di conversione, che sono spiegati nelle sezioni seguenti:

1. “Conversione dei dati client JMS” a pagina 169
2. “Conversione dati applicazione” a pagina 170
3. “Conversione dati del gestore code” a pagina 170
4. “Conversione dati del canale dei messaggi” a pagina 171

Dove deve essere eseguita la conversione?

La sezione, “Scelta di un approccio alla conversione del messaggio: il ricevitore è corretto” a pagina 171, descrive il solito approccio di "ricezione". "Il ricevitore è valido" anche per la conversione dati JMS .

Conversione dei dati client JMS

JMS client[1]la conversione dei dati è la conversione di oggetti e primitive Java in byte in un messaggio JMS quando viene inviato a una destinazione e di nuovo la conversione quando viene ricevuto. La conversione dei dati del client JMS utilizza i metodi delle classi `JMSMessage` . I metodi sono elencati per il tipo di classe `JMSMessage` in Tabella 31 a pagina 172.

La conversione da e verso la rappresentazione JVM interna di numeri e testo viene eseguita per i metodi read, get, set e write. La conversione viene eseguita quando il messaggio viene inviato e quando uno dei metodi read o get viene richiamato su un messaggio ricevuto.

La codepage e la codifica numerica utilizzate per scrivere o impostare il contenuto di un messaggio sono definite come attributi della destinazione. La codepage di destinazione e la codifica numerica possono essere modificate in modo amministrativo. Un'applicazione può anche sovrascrivere la codepage di destinazione e la codifica impostando le proprietà del messaggio che controllano la scrittura o l'impostazione del contenuto del messaggio.

Se si desidera convertire la codifica dei numeri quando un messaggio `JMSBytesMessage` viene inviato a una destinazione non definita come `Native` codifica, è necessario impostare la proprietà del messaggio `JMS_IBM_ENCODING` prima di inviare il messaggio. Se si sta seguendo il modello "destinatario fa bene" o se si stanno scambiando messaggi tra applicazioni JMS , l'applicazione non deve impostare `JMS_IBM_ENCODING`. Nella maggior parte dei casi, è possibile lasciare la proprietà `Encoding` come `Native`.

Per i messaggi `JMSStreamMessage`, `JMSMapMessage`e `JMSTextMessage` , vengono utilizzate le proprietà dell'identificativo della serie di caratteri della destinazione. La codifica viene ignorata durante l'invio poiché i numeri vengono scritti in formato testo. Il programma applicativo client JMS non deve impostare `JMS_IBM_CHARACTER_SET` prima di inviare il messaggio se la proprietà della serie di caratteri di destinazione da applicare.

Per ottenere i dati in un messaggio, un'applicazione richiama i metodi read o get del messaggio JMS . I metodi fanno riferimento alla codepage e alla codifica definiti nell'intestazione del messaggio precedente per creare correttamente gli oggetti e le primitive Java .

La conversione dei dati del client JMS soddisfa le esigenze della maggior parte delle applicazioni JMS che scambiano messaggi tra un client JMS e un altro. Non si codifica alcuna conversione dati esplicita. Non si utilizza la classe `java.nio.charset.Charset` , generalmente utilizzata quando si scrive il testo in un file. I metodi `writeString` e `setString` fanno la conversione per conto dell'utente.

Per ulteriori dettagli sulla conversione dati del client JMS , consultare “Conversione e codifica dei messaggi del client JMS” a pagina 181.

[1] "JMS Client" fa riferimento al IBM MQ classes for JMS che implementa l'interfaccia JMS , che viene eseguita in modalità client o bind.

Conversione dati applicazione

Un'applicazione client JMS può eseguire una conversione dei dati carattere esplicita utilizzando la classe `java.nio.charset.Charset` ; consultare gli esempi in Figura 14 a pagina 174 e Figura 15 a pagina 174. I dati stringa vengono convertiti in byte, utilizzando il metodo `getBytes` e inviati come byte. I byte vengono riconvertiti in testo utilizzando un costruttore `String` che utilizza un array di byte e un `Charset`. I dati carattere vengono convertiti utilizzando i metodi `encode` e `decode` `Charset` . Generalmente il messaggio viene inviato o ricevuto come `JMSBytesMessage`, poiché la parte del messaggio di un `JMSBytesMessage` non contiene altro che i dati scritti dall'applicazione[2]. È anche possibile inviare e ricevere byte utilizzando `JMSStreamMessage`, `JMSMapMessage`o `JMSObjectMessage`.

Non esistono metodi Java per codificare e decodificare i byte che contengono dati numerici rappresentati in formati di codifica differenti. I dati numerici vengono codificati e decodificati automaticamente utilizzando i metodi di lettura e scrittura `JMSMessage` numerici. I metodi di lettura e scrittura utilizzano il valore dell'attributo `JMS_IBM_ENCODING` dei dati del messaggio.

Un utilizzo tipico per la conversione dei dati dell'applicazione è quando il client JMS invia o riceve un messaggio formattato da un'applicazione nonJMS . Un messaggio formattato contiene dati di testo, numerici e byte organizzati in base alla lunghezza dei campi di dati. A meno che l'applicazione nonJMS non abbia specificato il formato del messaggio come "MQSTR ", il messaggio viene costruito come `JMSBytesMessage`. Per ricevere i dati dei messaggi formattati in `JMSBytesMessage` è necessario richiamare una sequenza di metodi. I metodi devono essere richiamati nello stesso ordine in cui i campi sono stati scritti nel messaggio. Se i campi sono numerici, è necessario conoscere la codifica e la lunghezza dei dati numerici. Se uno dei campi contiene dati di byte o di testo, è necessario conoscere la lunghezza dei dati di byte nel messaggio. Esistono due metodi per convertire un messaggio formattato in un oggetto Java facile da usare.

1. Creare una classe Java corrispondente al record per incapsulare la lettura e la scrittura del messaggio. L'accesso ai dati nel record è con i metodi get e set della classe.
2. Costruire una classe Java corrispondente al record estendendo la classe `com.ibm.mq.headers` . L'accesso ai dati nella classe è con gli accessor specifici del tipo del modulo, `getStringValue(`*`fieldname`*`);`

Consultare “Scambio di un record formattato con un'applicazione nonJMS” a pagina 189.

Conversione dati del gestore code

La conversione della codepage può essere eseguita dal gestore code quando un programma client JMS riceve un messaggio. La conversione è uguale alla conversione eseguita per un programma C. Un programma C imposta `MQGMO_CONVERT` come opzione di parametro `MQGET` `GetMsgOpts` ; consultare Figura 13 a pagina 173. Un gestore code esegue la conversione per un programma client JMS che sta ricevendo un messaggio, se la proprietà di destinazione `WMQ_RECEIVE_CONVERSION` è impostata su `WMQ_RECEIVE_CONVERSION_QMGR`, il programma client JMS può impostare anche la proprietà di destinazione; consultare Figura 12 a pagina 170.

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Oppure,

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 12. Abilita conversione dati gestore code

[2] Un'eccezione: i dati scritti utilizzando `writeUTF` iniziano con un campo di lunghezza di 2 byte

Il vantaggio principale della conversione del gestore code si ha quando si scambiano messaggi con applicazioni nonJMS . Se il campo `Format` nel messaggio è definito e la serie di caratteri di destinazione, o la codifica, è diversa dal messaggio, il gestore code esegue la conversione dei dati per l'applicazione di destinazione, se l'applicazione lo richiede. Il gestore code converte i dati del messaggio formattati in base a uno dei tipi di messaggio IBM MQ predefiniti, come un'intestazione CICS bridge (`MQCIH`). Se il campo `Format` è definito dall'utente, il gestore code cerca un'uscita di conversione dati con il nome fornito nel campo `Format` .

La conversione dei dati del gestore code viene utilizzata al meglio con il modello di progettazione "Il destinatario fa bene" . Un client JMS di invio non deve eseguire la conversione. Un programma di ricezione nonJMS si basa sull'uscita di conversione per garantire che il messaggio venga consegnato nella codepage e nella codifica richieste. Con un client JMS di invio e un destinatario nonJMS , l'esempio si applica a IBM MQ.

È possibile creare un'uscita di conversione dati utilizzando il programma di utilità di uscita di conversione dati, **crtmqcvx**, per consentire al gestore code di convertire i propri dati formattati del record. È possibile creare il proprio formato record, utilizzare `com.ibm.mq.headers` per accedervi come classe Java e utilizzare la propria uscita di conversione per convertirlo. Su z/OS il programma di utilità è denominato **CSQUCVX**e su IBM i, **CVTMQMDTA**. Consultare “Scambio di un record formattato con un'applicazione nonJMS” a pagina 189.

Conversione dati del canale dei messaggi

IBM MQ I canali mittente, Server, Ricevente cluster e Mittente cluster hanno un'opzione di conversione del messaggio, `CONVERT`. Il contenuto di un messaggio può essere facoltativamente convertito quando viene inviato un messaggio. La conversione avviene all'estremità di invio del canale. La definizione ricevente del cluster viene utilizzata per definire automaticamente il corrispondente canale mittente del cluster.

La conversione dei dati da parte dei canali dei messaggi viene generalmente utilizzata se non è possibile utilizzare altre forme di conversione.

Scelta di un approccio alla conversione del messaggio: "il ricevitore è corretto"

L'approccio usuale nella progettazione dell'applicazione IBM MQ per la conversione del codice è "receiver rende bene". "Destinatario corretto" riduce il numero di conversioni di messaggi. Evita inoltre il problema di errori di canale imprevisti se la conversione del messaggio non riesce su un gestore code intermediario durante il trasferimento del messaggio. La regola "receiver make good" viene interrotta solo se esiste un motivo per cui il ricevitore non può essere corretto. Ad esempio, la piattaforma ricevente potrebbe non avere la serie di caratteri corretta.

"Il ricevitore fa bene" è anche una buona guida generale per applicazioni client JMS . Ma in casi specifici, la conversione al set di caratteri corretto all'origine può essere più efficiente. La conversione dalla rappresentazione interna JVM deve essere eseguita quando viene inviato un messaggio contenente tipi di testo o numerici. La conversione alla serie di caratteri richiesta dal destinatario, se il destinatario non è un client JMS , potrebbe eliminare la necessità per il destinatario nonJMS di eseguire la conversione. Se il destinatario è un client JMS , eseguirà nuovamente la conversione, comunque, per decodificare i dati del messaggio e creare oggetti e primitive Java .

La differenza tra le applicazioni client JMS e le applicazioni scritte in un linguaggio come C, è che Java deve eseguire la conversione dati. Un'applicazione Java deve convertire numeri e testo dalla loro rappresentazione interna in un formato codificato utilizzato nei messaggi.

Impostando la destinazione o le proprietà del messaggio, è possibile impostare la serie di caratteri e la codifica utilizzate da IBM MQ per codificare i numeri e il testo nei messaggi. Normalmente, la serie di caratteri viene lasciata come `1208` e la codifica come `Native`.

IBM MQ non converte array di byte. Per codificare stringhe e array di caratteri in array di byte, utilizzare il pacchetto `java.nio.charset` . `Charset` specifica la serie di caratteri utilizzata per convertire una stringa o una schiera di caratteri in una schiera di byte. È anche possibile decodificare una schiera di byte in una stringa o una schiera di caratteri utilizzando un `Charset`. Non è consigliabile affidarsi a `java.nio.charset.Charset.defaultCodePage` quando si codificano stringhe e array di caratteri.

Il valore predefinito `Charset` è generalmente `windows-1252` su Windowse `UTF-8` su AIX and Linux. `windows-1252` è una serie di caratteri a singolo byte e `UTF-8` è una serie di caratteri a più byte.

Generalmente, lasciare la serie di caratteri di destinazione e le proprietà di codifica sui valori predefiniti di `UTF-8` e `Native` quando si scambiano messaggi con altre applicazioni JMS . Se si scambiano messaggi contenenti numeri o testo con un'applicazione JMS , scegliere uno dei tipi di messaggi `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage`o `JMSObjectMessage` che si adattano al proprio scopo. Non ci sono altre attività di conversione da eseguire.

Se si scambiano messaggi con applicazioni nonJMS che utilizzano un formato record, è più complicato. A meno che l'intero record non contenga testo e non possa essere trasferito come `JMSTextMessage`, è necessario codificare e decodificare il testo nell'applicazione. Impostare il tipo di messaggio di destinazione su `MQ`e utilizzare `JMSBytesMessage` per evitare che IBM MQ classes for JMS aggiungano ulteriori informazioni di intestazione e tag ai dati del messaggio. Utilizzare i metodi `JMSBytesMessage` per scrivere numeri e byte e la classe `Charset` converte esplicitamente il testo in schiere di byte. Una serie di fattori potrebbe influenzare la scelta della serie di caratteri:

- Prestazioni: È possibile ridurre il numero di conversioni trasformando il testo in un set di caratteri utilizzato sul maggior numero di server?
- Uniformità: trasferire tutti i messaggi nella stessa serie di caratteri.
- Ricchezza: Quali set di caratteri hanno tutti i punti di codice che le applicazioni devono utilizzare?
- Semplicità: le serie di caratteri a byte singolo sono più semplici da utilizzare rispetto alle serie di caratteri a lunghezza variabile e multibyte.

Consultare “Scambio di un record formattato con un'applicazione nonJMS” a pagina 189. per esempi di conversione di messaggi scambiati con applicazioni nonJMS .

Esempi

Tabella dei tipi di messaggio e dei tipi di conversione

Tabella 31. Tipi di messaggi e tipi di conversione

	Tipo di conversione			
Tipo messaggio	**Testo**	**Numerico**	**Altro**	**Nessuna**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

`JMSBytesMessage`	`readUTF`
	`writeUTF`
		`readDouble`
		`readFloat`
		`readInt`
		`readLong`
		`readShort`
		`readUnsignedShort`
		`writeDouble`
		`writeFloat`
		`writeInt`
		`writeLong`
		`writeShort`
			`readBoolean`
			`readObject`
			`writeBoolean`
			`writeObject`
				`readByte`
				`readUnsignedByte`
				`readBytes`
				`readChar`
				`writeByte`
				`writeBytes`
				`writeChar`

Tabella 31. Tipi di messaggi e tipi di conversione (Continua)				
	Tipo di conversione			
Tipo messaggio	**Testo**	**Numerico**	**Altro**	**Nessuna**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Richiamo della conversione dati da un programma C

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                 */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Figura 13. Frammento di codice da `amqsget0.c`

Invio e ricezione di testo in un JMSBytesMessage

Il codice in Figura 14 a pagina 174 invia una stringa in un BytesMessage. Per semplicità, l'esempio invia una singola stringa, per cui un `JMSTextMessage` è più appropriato. Per ricevere un messaggio di testo in byte contenente una combinazione di tipi, è necessario conoscere la lunghezza della stringa in byte, denominata *TEXT_LENGTH* in Figura 15 a pagina 174. Anche per una stringa con un numero fisso di caratteri, la lunghezza della rappresentazione in byte potrebbe essere maggiore.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Figura 14. Invio di un String in un JMSBytesMessage

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Figura 15. Ricezione di un String da un JMSBytesMessage

Concetti correlati

Conversione e codifica dei messaggi del client JMS
Vengono elencati i metodi utilizzati per eseguire la conversione e la codifica dei messaggi del client JMS , con esempi di codice di ogni tipo di conversione.

Conversione dati del gestore code
La conversione dei dati del gestore code è sempre stata disponibile per le applicazioni nonJMS che ricevono i messaggi dai client JMS . I client JMS che ricevono i messaggi utilizzano anche la conversione dei dati del gestore code, che è facoltativa.

Attività correlate

Scambio di un record formattato con un'applicazione nonJMS
Seguire i passi suggeriti in questa attività per progettare e creare un'uscita di conversione dati e un'applicazione client JMS che può scambiare messaggi con un'applicazione nonJMS utilizzando JMSBytesMessage. Lo scambio di un messaggio formattato con un'applicazione nonJMS può avvenire con o senza la chiamata di un'uscita di conversione dati.

Riferimenti correlati

Conversione e tipi di messaggi JMS
La scelta del tipo di messaggio influisce sull'approccio alla conversione del messaggio. L'interazione tra la conversione del messaggio e il tipo di messaggio è descritta per i tipi di messaggio JMS JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagee JMSBytesMessage.

Conversione e tipi di messaggi JMS
La scelta del tipo di messaggio influisce sull'approccio alla conversione del messaggio. L'interazione tra la conversione del messaggio e il tipo di messaggio è descritta per i tipi di messaggio JMS JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagee JMSBytesMessage.

JMSObjectMessage

JMSObjectMessage contiene un oggetto e tutti gli oggetti a cui fa riferimento, serializzati in un flusso di byte dalla JVM. Il testo viene serializzato in UTF-8e limitato a stringhe o schiere di caratteri di non più di 65534 byte. Un vantaggio di JMSObjectMessage è che le applicazioni non sono coinvolte in alcun problema di conversione dati purché utilizzino solo i metodi e gli attributi dell'oggetto. JMSObjectMessage fornisce la conversione dei dati per oggetti complessi senza che il programmatore dell'applicazione consideri come codificare un oggetto in un messaggio. Lo svantaggio di utilizzare JMSObjectMessage è che può essere scambiato solo con altre applicazioni JMS . Scegliendo uno degli altri tipi di messaggi JMS , è possibile scambiare JMS messaggi con applicazioni nonJMS .

“Invio e ricezione di un JMSObjectMessage” a pagina 177 mostra un `String` oggetto scambiato in un messaggio.

Un'applicazione client JMS può ricevere un `JMSObjectMessage` solo in un messaggio con un corpo in stile JMS. La destinazione deve specificare un corpo di stile JMS .

JMSTextMessage

`JMSTextMessage` contiene una singola stringa di testo. Quando viene inviato un messaggio di testo, il testo `Format` è impostato su `"MQSTR   "`, `WMQConstants.MQFMT_STRING`. Il `CodedCharacterSetId` del testo è impostato sull'identificativo della serie di caratteri codificati definito per la destinazione. Il testo viene codificato in `CodedCharacterSetId` da IBM MQ. I campi `CodedCharacterSetId` e `Format` sono impostati nel descrittore del messaggio, `MQMD`, o nei campi JMS in un `MQRFH2`. Se il messaggio è definito come avente uno stile del corpo del messaggio `WMQ_MESSAGE_BODY_MQ` o lo stile del contenuto non è specificato, ma la destinazione è `WMQ_TARGET_DEST_MQ`, vengono impostati i campi del descrittore del messaggio. In alternativa, il messaggio ha un JMS `RFH2` e i campi sono impostati nella parte fissa di `MQRFH2`.

Un'applicazione può sostituire il CCSID (coded character set identifier) definito per una destinazione. Deve impostare la proprietà del messaggio `JMS_IBM_CHARACTER_SET` su un CCSID (coded character set identifier); consultare l'esempio in “Invio e ricezione di un JMSTextmessage” a pagina 177.

Quando il client JMS richiama la conversione del gestore code del metodo `consumer.receive` è facoltativa. La conversione del gestore code è abilitata impostando la proprietà di destinazione `WMQ_RECEIVE_CONVERSION` su `WMQ_RECEIVE_CONVERSION_QMGR`. Il gestore code converte il messaggio di testo dal `JMS_IBM_CHARACTER_SET` specificato per il messaggio prima di trasferire il messaggio sul client JMS . La serie di caratteri del messaggio convertito è `1208`, `UTF-8`, a meno che la destinazione non abbia un `WMQ_RECEIVE_CCSID`diverso. Il `CodedCharacterSetId` nel messaggio che fa riferimento a `JMSTextMessage` viene aggiornato all'ID della serie di caratteri di destinazione. Il testo viene decodificato dalla serie di caratteri di destinazione in Unicode mediante il metodo `getText` ; consultare l'esempio in “Invio e ricezione di un JMSTextmessage” a pagina 177.

Un `JMSTextMessage` può essere inviato in un corpo del messaggio in stile MQ, senza un'intestazione JMS `MQRFH2` . Il valore degli attributi di destinazione, `WMQ_MESSAGE_BODY` e `WMQ_TARGET_DEST` determinano lo stile del messaggio, a meno che non venga sovrascritto dall'applicazione. L'applicazione può sovrascrivere i valori impostati sulla destinazione richiamando `destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ)` o `destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)`.

Se si invia un `JMSTextMessage` con un corpo di stile MQ inviandolo a una destinazione con `WMQ_MESSAGE_BODY` impostato su `WMQ_MESSAGE_BODY_MQ`, non è possibile riceverlo come `JMSTextMessage` dalla stessa destinazione. Tutti i messaggi ricevuti da una destinazione con `WMQ_MESSAGE_BODY` impostato su `WMQ_MESSAGE_BODY_MQ` vengono ricevuti come `JMSBytesMessage`. Se si tenta di ricevere il messaggio come `JMSTextMessage` , viene generata un'eccezione, `ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage`.

Nota: Il testo in `JMSBytesMessage` non viene convertito dal client JMS. Il client può ricevere solo il testo nel messaggio come schiera di byte. Se la conversione del gestore code è abilitata, il testo viene convertito dal gestore code, ma il JMS client deve ancora riceverlo come un array di byte in un `JMSBytesMessage`.

È generalmente preferibile utilizzare la proprietà `WMQ_TARGET_DEST` per controllare se un `JMSTextMessage` viene inviato con uno stile del corpo MQ o JMS. È quindi possibile ricevere il messaggio da una destinazione che ha `WMQ_TARGET_DEST` impostato su `WMQ_TARGET_DEST_MQ` o `WMQ_TARGET_DEST_JMS`. `WMQ_TARGET_DEST` non ha effetto sul ricevitore.

JMSMapMessage e JMSStreamMessage

Questi due tipi di messaggio JMS sono simili. È possibile leggere e scrivere tipi primitivi nei messaggi utilizzando metodi basati sulle interfacce `DataInputStream` e `DataOutputStream` ; consultare

“Tabella dei tipi di messaggio e dei tipi di conversione” a pagina 180. I dettagli sono descritti in “Conversione e codifica dei messaggi del client JMS” a pagina 181. Ogni primitiva è contrassegnata; consultare “Il corpo del messaggio JMS” a pagina 166.

I dati numerici vengono letti e scritti nel messaggio codificato come testo XML. Non viene fatto riferimento alla proprietà di destinazione, JMS_IBM_ENCODING. I dati di testo vengono trattati come il testo in un JMSTextMessage. Se si dovesse esaminare il contenuto del messaggio creato dall'esempio in Figura 20 a pagina 178, tutti i dati del messaggio sarebbero in EBCDIC come sono stati inviati con un valore della serie di caratteri 37.

È possibile inviare più elementi in un JMSMapMessage o JMSStreamMessage.

È possibile richiamare i singoli elementi di dati per nome da un JMSMapMessageo per posizione da un JMSStreamMessage. Ogni elemento viene decodificato quando viene richiamato un metodo get o read utilizzando il valore di CodedCharacterSetId memorizzato nel messaggio. Se il metodo utilizzato per richiamare l'elemento restituisce un tipo diverso dal tipo inviato, il tipo viene convertito. Se il tipo non può essere convertito, viene generata un'eccezione. Per dettagli, vedere Classe JMSStreamMessage . L'esempio riportato in “Invio di dati in un JMSStreamMessage e JMSMapMessage” a pagina 178 illustra la conversione del tipo e il richiamo del contenuto JMSMapMessage fuori sequenza.

Il campo MQRFH2.format per JMSMapMessage e JMSStreamMessage è impostato su "MQSTR ". Se la proprietà di destinazione WMQ_RECEIVE_CONVERSION è impostata su WMQ_RECEIVE_CONVERSION_QMGR, i dati del messaggio vengono convertiti dal gestore code prima di essere inviati al client JMS . MQRFH2.CodedCharacterSetId del messaggio è il WMQ_RECEIVE_CCSID della destinazione. MQRFH2.Encoding è Native. Se WMQ_RECEIVE_CONVERSION è WMQ_RECEIVE_CONVERSION_CLIENT_MSG il CodedCharacterSetId e Encoding di MQRFH2 è il valore impostato dal mittente.

Un'applicazione client JMS può ricevere un JMSMapMessage o JMSStreamMessage solo in un messaggio che ha un corpo in stile JMSe da una destinazione che non specifica un corpo in stile MQ .

JMSBytesMessage

Un JMSBytesMessage può contenere più tipi primitivi. È possibile leggere e scrivere tipi primitivi nei messaggi utilizzando metodi basati sulle interfacce DataInputStream e DataOutputStream ; consultare “Tabella dei tipi di messaggio e dei tipi di conversione” a pagina 180. I dettagli sono descritti in “Conversione e tipi di messaggi JMS” a pagina 174.

La codifica dei dati numerici nel messaggio è controllata dal valore di JMS_IBM_ENCODING impostato prima della scrittura di dati numerici in JMSBytesMessage. Un'applicazione può sovrascrivere la codifica Native predefinita definita per JMSBytesMessage impostando la proprietà del messaggio JMS_IBM_ENCODING.

I dati di testo possono essere letti e scritti in UTF-8 utilizzando readUTF e writeUTFoppure in Unicode utilizzando i metodi readChar e writeChar . Non ci sono metodi che utilizzano CodedCharacterSetId. In alternativa, il client JMS può codificare e decodificare il testo in byte utilizzando la classe Charset . Trasferisce i byte tra la JVM e il messaggio senza che IBM MQ classes for JMS esegua alcuna conversione; consultare “Invio e ricezione di testo in un JMSBytesMessage” a pagina 178.

Un JMSBytesMessage inviato a un'applicazione MQ viene generalmente inviato in un corpo del messaggio in stile MQ, senza un'intestazione JMS MQRFH2 . Se viene inviato a un'applicazione JMS , lo stile del contenuto del messaggio è di solito JMS. Il valore degli attributi di destinazione, WMQ_MESSAGE_BODY e WMQ_TARGET_DEST determinano lo stile del messaggio, a meno che non venga sovrascritto dall'applicazione. L'applicazione può sovrascrivere i valori impostati sulla destinazione richiamando destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) o destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Se si invia un JMSBytesMessage con un corpo di stile MQ , è possibile ricevere il messaggio da una destinazione che definisce uno stile di contenuto del messaggio MQ o JMS . Se si invia un JMSBytesMessage con un corpo di stile JMS, è necessario ricevere il messaggio da una destinazione

che definisce uno stile di contenuto del messaggio JMS . In caso contrario, il MQRFH2 viene considerato come parte dei dati del messaggio utente, il che potrebbe non essere quello previsto.

Se un messaggio ha uno stile di corpo MQ o JMS , il modo in cui viene ricevuto non è influenzato dall'impostazione WMQ_TARGET_DEST.

Il messaggio potrebbe essere trasformato in un secondo momento, dal gestore code, se viene fornito un Format per i dati del messaggio e la conversione dei dati del gestore code è abilitata. Non utilizzare il campo formato se non per specificare il formato dei dati del messaggio o lasciarlo vuoto, MQConstants.MQFMT_NONE

È possibile inviare più elementi in JMSBytesMessage. Ogni elemento numerico viene convertito quando il messaggio viene inviato utilizzando la codifica definita per il messaggio.

È possibile richiamare i singoli elementi di dati da JMSBytesMessage. Richiamare i metodi di lettura nello stesso ordine in cui vengono richiamati i metodi di scrittura per creare il messaggio. Ogni elemento numerico viene convertito quando il messaggio viene richiamato utilizzando il valore Encoding memorizzato nel messaggio.

A differenza di JMSMapMessage e JMSStreamMessage, JMSBytesMessage contiene solo dati scritti dall'applicazione. Non vengono memorizzati ulteriori dati nei dati del messaggio, come ad esempio i tag XML utilizzati per definire gli elementi in JMSMapMessage e JMSStreamMessage. Per questo motivo, utilizzare JMSBytesMessage per trasferire i messaggi formattati per altre applicazioni.

La conversione tra JMSBytesMessage e DataInputStream e DataOutputStream è utile in alcune applicazioni. Il codice basato sull'esempio, “Lettura e scrittura di messaggi utilizzando DataInputStream e DataOutputStream” a pagina 179, è necessario per utilizzare il package com.ibm.mq.header con JMS.

Esempi

Invio e ricezione di un JMSObjectMessage

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

Figura 16. Invio e ricezione di un JMSObjectMessage

Invio e ricezione di un JMSTextmessage

Un messaggio di testo non può contenere testo in serie di caratteri differenti. L'esempio mostra il testo in diverse serie di caratteri, inviato in due diversi messaggi.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figura 17. Invia messaggio di testo nella serie di caratteri definita dalla destinazione

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figura 18. Invia messaggio di testo in `ccsid 37`

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figura 19. Ricevi messaggio di testo

Invio di dati in un JMSStreamMessage e JMSMapMessage

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                         " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                       " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

Figura 20. Invio di dati in `JMSStreamMessage` e `JMSMapMessage`

Invio e ricezione di testo in un JMSBytesMessage

Il codice in Figura 21 a pagina 179 invia una stringa in un BytesMessage. Per semplicità, l'esempio invia una singola stringa, per cui un `JMSTextMessage` è più appropriato. Per ricevere un messaggio di testo in byte contenente una combinazione di tipi, è necessario conoscere la lunghezza della stringa in byte, denominata *`TEXT_LENGTH`* in Figura 22 a pagina 179. Anche per una stringa con un numero fisso di caratteri, la lunghezza della rappresentazione in byte potrebbe essere maggiore.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Figura 21. Invio di un String in un JMSBytesMessage

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Figura 22. Ricezione di un String da un JMSBytesMessage

Lettura e scrittura di messaggi utilizzando DataInputStream e DataOutputStream

Il codice in Figura 23 a pagina 179 crea un JMSBytesMessage utilizzando un DataOutputStream.

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

Figura 23. Inviare un JMSBytesMessage utilizzando un DataOutputStream

L'istruzione che imposta la proprietà JMS_IBM_ENCODING è commentata. L'istruzione è valida se si scrive direttamente in un JMSBytesMessage, ma non ha alcun effetto quando si scrive in DataOutputStream. I numeri scritti in DataOutputStream sono codificati nella codifica Native . L'impostazione JMS_IBM_ENCODING non ha alcun effetto.

Il codice in Figura 24 a pagina 180 riceve un JMSBytesMessage utilizzando un DataInputStream.

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

Figura 24. Ricevere un JMSBytesMessage utilizzando un DataInputStream

La codepage viene stampata utilizzando la proprietà codepage dei dati del messaggio di input, JMS_IBM_CHARACTER_SET. Nell'immissione JMS_IBM_CHARACTER_SET è una codepage Java e non un CCSID (coded character set identifier) numerico.

Tabella dei tipi di messaggio e dei tipi di conversione

Tabella 32. Tipi di messaggi e tipi di conversione

	Tipo di conversione			
Tipo messaggio	**Testo**	**Numerico**	**Altro**	**Nessuna**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

Tabella 32. Tipi di messaggi e tipi di conversione (Continua)				
	Tipo di conversione			
Tipo messaggio	**Testo**	**Numerico**	**Altro**	**Nessuna**

`JMSStreamMessage`	`readString`
	`writeString`
		`readDouble`
		`readFloat`
		`readInt`
		`readLong`
		`readShort`
		`writeDouble`
		`writeFloat`
		`writeInt`
		`writeLong`
		`writeShort`
			`readBoolean`
			`readObject`
			`writeBoolean`
			`writeObject`
				`readByte`
				`readBytes`
				`readChar`
				`writeByte`
				`writeBytes`
				`writeChar`

`JMSMapMessage`	`getString`
	`setString`
		`getDouble`
		`getFloat`
		`getInt`
		`getLong`
		`getShort`
		`setDouble`
		`setFloat`
		`setInt`
		`setLong`
		`setShort`
			`getBoolean`
			`getObject`
			`setBoolean`
			`setObject`
				`getByte`
				`getBytes`
				`readChar`
				`setByte`
				`setBytes`
				`setChar`

Concetti correlati

Approcci di conversione dei messaggi JMS

Una serie di approcci di conversione dati sono aperti ai progettisti di applicazioni JMS . Questi approcci non sono esclusivi; alcune applicazioni probabilmente utilizzeranno una combinazione di questi approcci. Se l'applicazione sta scambiando solo testo o messaggi solo con altre applicazioni JMS , normalmente non si considera la conversione dei dati. La conversione dei dati viene eseguita automaticamente da IBM MQ.

Conversione e codifica dei messaggi del client JMS

Vengono elencati i metodi utilizzati per eseguire la conversione e la codifica dei messaggi del client JMS , con esempi di codice di ogni tipo di conversione.

Conversione dati del gestore code

La conversione dei dati del gestore code è sempre stata disponibile per le applicazioni nonJMS che ricevono i messaggi dai client JMS . I client JMS che ricevono i messaggi utilizzano anche la conversione dei dati del gestore code, che è facoltativa.

Attività correlate

Scambio di un record formattato con un'applicazione nonJMS

Seguire i passi suggeriti in questa attività per progettare e creare un'uscita di conversione dati e un'applicazione client JMS che può scambiare messaggi con un'applicazione nonJMS utilizzando `JMSBytesMessage`. Lo scambio di un messaggio formattato con un'applicazione nonJMS può avvenire con o senza la chiamata di un'uscita di conversione dati.

Conversione e codifica dei messaggi del client JMS

Vengono elencati i metodi utilizzati per eseguire la conversione e la codifica dei messaggi del client JMS , con esempi di codice di ogni tipo di conversione.

La conversione e la codifica si verificano quando le primitive o gli oggetti Java vengono letti o scritti in e da messaggi JMS . La conversione è denominata conversione dei dati client JMS per distinguerla dalla conversione dei dati del gestore code e dalla conversione dei dati dell'applicazione. La conversione

avviene esclusivamente quando i dati vengono letti o scritti in un messaggio JMS . Il testo viene convertito in e dalla rappresentazione interna Unicode a 16 bit[3]alla serie di caratteri utilizzata per il testo nei messaggi. I dati numerici vengono convertiti in tipi numerici primitivi Java nella codifica definita per il messaggio. Se viene eseguita la conversione e quale tipo di conversione viene eseguito, dipende dal tipo di messaggio JMS e dall'operazione di lettura o scrittura.

Tabella 33 a pagina 182 categorizza i metodi di lettura e scrittura per diversi JMS tipi di messaggio in base al tipo di conversione eseguita. I tipi di conversione sono descritti nel testo che segue la tabella.

Tabella 33. Tipi di messaggi e tipi di conversione

	Tipo di conversione			
Tipo messaggio	**Testo**	**Numerico**	**Altro**	**Nessuna**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar

[3] Alcune rappresentazioni Unicode richiedono più di 16 bit. Consultare un riferimento Java SE.

Tabella 33. Tipi di messaggi e tipi di conversione (Continua)				
	Tipo di conversione			
Tipo messaggio	**Testo**	**Numerico**	**Altro**	**Nessuna**
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Testo

Il valore predefinito CodedCharacterSetId per una destinazione è 1208, UTF-8. Per impostazione predefinita, il testo viene convertito da Unicode e inviato come stringa di testo UTF-8 . In ricezione, il testo viene convertito dal set di caratteri codificati nel messaggio ricevuto dal client in Unicode.

I metodi setText e writeString convertono il testo da Unicode nella serie di caratteri definita per la destinazione. Un'applicazione può sostituire la serie di caratteri di destinazione impostando la proprietà del messaggio JMS_IBM_CHARACTER_SET. JMS_IBM_CHARACTER_SET, quando si invia un messaggio deve essere un CCSID (coded character set identifier) numerico[4].

I frammenti di codice in "Invio e ricezione di un JMSTextmessage" a pagina 185 inviano due messaggi. Uno viene inviato nella serie di caratteri definita per la destinazione e l'altro nella serie di caratteri 37, definita dall'applicazione.

I metodi getText e readString convertono il testo nel messaggio dalla serie di caratteri definita nel messaggio in Unicode. I metodi utilizzano la codepage definita nella proprietà del messaggio JMS_IBM_CHARACTER_SET. La codepage è associata da MQRFH2.CodedCharacterSetId a meno che il messaggio non sia un messaggio di MQe non abbia MQRFH2. Se il messaggio è un messaggio di tipo MQ, senza MQRFH2, la codepage viene associata da MQMD.CodedCharacterSetId.

Il frammento di codice in Figura 29 a pagina 186 riceve il messaggio che è stato inviato alla destinazione. Il testo nel messaggio viene riconvertito dalla codepage IBM037 in Unicode.

Nota: Un modo semplice per controllare che il testo venga convertito nella serie di caratteri codificati 37 consiste nell'utilizzare Esplora risorse di IBM MQ . Sfoglia la coda e mostra le proprietà del messaggio prima che venga richiamato.

Contrasto del frammento di codice in Figura 28 a pagina 186 con il frammento di codice non corretto in Figura 25 a pagina 183. Nel frammento non corretto la stringa di testo viene convertita due volte, una dall'applicazione e un'altra da IBM MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Figura 25. Conversione codepage non corretta

Il metodo writeUTF converte il testo da Unicode a 1208, UTF-8. La stringa di testo ha come prefisso una lunghezza di 2 byte. La lunghezza massima della stringa di testo è 65534 byte. Il metodo

[4] Quando si riceve un messaggio, JMS_IBM_CHARACTER_SET è un nome di codepage Java Charset .

readUTF legge un elemento in un messaggio scritto dal metodo writeUTF . Legge esattamente il numero di byte scritti dal metodo writeUTF .

Numerico

La codifica numerica predefinita per una destinazione è Native. La costante di codifica Native per Java ha il valore 273, x'00000111', che è lo stesso per tutte le piattaforme. Alla ricezione, i numeri nel messaggio vengono trasformati correttamente in primitive Java numeriche. La trasformazione usa la codifica definita nel messaggio e il tipo restituito dal metodo read.

Il metodo di invio converte i numeri aggiunti a un messaggio da set e write nella codifica numerica definita per la destinazione. La codifica di destinazione può essere sovrascritta per un messaggio da un'applicazione che imposta la proprietà del messaggio, JMS_IBM_ENCODING ; ad esempio:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

I metodi numerici get e read convertono i numeri nel messaggio dalla codifica numerica definita nel messaggio. Convertono i numeri nel tipo specificato dal metodo read o get ; consultare La proprietà ENCODING. I metodi utilizzano la codifica definita in JMS_IBM_ENCODING. La codifica viene associata da MQRFH2.Encoding a meno che il messaggio non sia un messaggio di tipo MQe non abbia MQRFH2. Se il messaggio è un messaggio di MQdi tipo, senza MQRFH2, i metodi utilizzano la codifica definita in MQMD.Encoding.

L'esempio in Figura 30 a pagina 186 mostra un'applicazione che codifica un numero nel formato di destinazione e lo invia in un JMSStreamMessage. Confrontare l'esempio in Figura 30 a pagina 186 con quello in Figura 31 a pagina 186. La differenza è che JMS_IBM_ENCODING deve essere impostato in un JMSBytesMessage.

Nota: Un modo semplice per controllare che il numero sia codificato correttamente consiste nell'utilizzare Esplora risorse di IBM MQ . Sfoglia la coda e mostra le proprietà del messaggio prima che venga utilizzato.

Altro

I metodi boolean codificano true e false come x'01' e x'00' in un JMSByteMessage, JMSStreamMessagee JMSMapMessage.

I metodi UTF codificano e decodificano Unicode in stringhe di testo UTF-8 . Le stringhe sono limitate a meno di 65536 caratteri e sono precedute dal campo di lunghezza di 2 byte.

I metodi Object avvolgono i tipi primitivi come oggetti. I tipi numerici e di testo vengono codificati o convertiti come se i tipi primitivi fossero stati letti o scritti utilizzando i metodi numerici e di testo.

Nessuna

I metodi readByte, readBytes, readUnsignedByte, writeBytee writeBytes ottengono o inserendo singoli byte o array di byte, tra l'applicazione e il messaggio senza conversione. I metodi readChar e writeChar ottengono e inserendo caratteri Unicode a 2 byte tra l'applicazione e il messaggio senza conversione.

Utilizzando i metodi readBytes e writeBytes , l'applicazione può eseguire la propria conversione del punto di codice, come in “Invio e ricezione di testo in un JMSBytesMessage” a pagina 186.

IBM MQ non esegue alcuna conversione di codepage nel client poiché il messaggio è un JMSBytesMessagee perché vengono utilizzati i metodi readBytes e writeBytes . Tuttavia, se i byte rappresentano il testo, assicurarsi che la codepage utilizzata dall'applicazione corrisponda alla serie di caratteri codificati della destinazione. Il messaggio potrebbe essere riconvertito da un'uscita di conversione del gestore code. Un'altra possibilità è che il programma client JMS ricevente possa seguire la convenzione di conversione di qualsiasi array di byte che rappresenta il testo nel messaggio in stringhe o caratteri utilizzando la proprietà JMS_IBM_CHARACTER_SET nel messaggio.

In questo esempio il client utilizza la serie di caratteri codificati di destinazione per la propria conversione:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

In alternativa, il client potrebbe aver scelto una codepage e quindi impostare la serie di caratteri codificati corrispondente nella proprietà JMS_IBM_CHARACTER_SET del messaggio. IBM MQ classes for Java utilizzare JMS_IBM_CHARACTER_SET per impostare il campo CodedCharacterSetId nelle proprietà JMS in MQRFH2o nel descrittore del messaggio, MQMD:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

Se una schiera di byte viene scritta in un JMSStringMessage o in un JMSMapMessage, IBM MQ classes for JMS non esegue la conversione dei dati, poiché i byte vengono immessi come dati esadecimali e non come testo in JMSStringMessage e JMSMapMessage.

Se i byte rappresentano i caratteri nell'applicazione, è necessario considerare quali punti di codice leggere e scrivere nel messaggio. Il codice in Figura 26 a pagina 185 segue la convenzione di utilizzo della serie di caratteri codificati di destinazione. Se si crea la stringa utilizzando la serie di caratteri predefinita per la JVM, il contenuto dei byte dipende dalla piattaforma. Una JVM su Windows generalmente ha un Charset predefinito di windows-1252e AIX and Linux ha UTF-8. Per lo scambio tra Windowse AIX and Linux, è necessario selezionare una codepage esplicita per lo scambio di testo come byte.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Figura 26. Scrittura di byte che rappresentano una stringa in JMSStreamMessage utilizzando la serie di caratteri di destinazione

Esempi

Invio e ricezione di un JMSTextmessage

Un messaggio di testo non può contenere testo in serie di caratteri differenti. L'esempio mostra il testo in diverse serie di caratteri, inviato in due diversi messaggi.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figura 27. Invia messaggio di testo nella serie di caratteri definita dalla destinazione

[5] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figura 28. Invia messaggio di testo in `ccsid 37`

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figura 29. Ricevi messaggio di testo

Esempi di codifica

Esempi che mostrano un numero inviato nella codifica definisce una destinazione. Notare che è necessario impostare la proprietà `JMS_IBM_ENCODING` di un `JMSBytesMessage` sul valore specificato per la destinazione.

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

Figura 30. Invio di un numero utilizzando la codifica di destinazione in un `JMSStreamMessage`

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

Figura 31. Invio di un numero utilizzando la codifica di destinazione in un `JMSBytesMessage`

Invio e ricezione di testo in un JMSBytesMessage

Il codice in Figura 32 a pagina 187 invia una stringa in un BytesMessage. Per semplicità, l'esempio invia una singola stringa, per cui un `JMSTextMessage` è più appropriato. Per ricevere un messaggio di testo in byte contenente una combinazione di tipi, è necessario conoscere la lunghezza della stringa in byte, denominata *TEXT_LENGTH* in Figura 33 a pagina 187. Anche per una stringa con un numero fisso di caratteri, la lunghezza della rappresentazione in byte potrebbe essere maggiore.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Figura 32. Invio di un `String` in un `JMSBytesMessage`

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Figura 33. Ricezione di un `String` da un `JMSBytesMessage`

Concetti correlati

Approcci di conversione dei messaggi JMS
Una serie di approcci di conversione dati sono aperti ai progettisti di applicazioni JMS . Questi approcci non sono esclusivi; alcune applicazioni probabilmente utilizzeranno una combinazione di questi approcci. Se l'applicazione sta scambiando solo testo o messaggi solo con altre applicazioni JMS , normalmente non si considera la conversione dei dati. La conversione dei dati viene eseguita automaticamente da IBM MQ.

Conversione dati del gestore code
La conversione dei dati del gestore code è sempre stata disponibile per le applicazioni nonJMS che ricevono i messaggi dai client JMS . I client JMS che ricevono i messaggi utilizzano anche la conversione dei dati del gestore code, che è facoltativa.

Attività correlate

Scambio di un record formattato con un'applicazione nonJMS
Seguire i passi suggeriti in questa attività per progettare e creare un'uscita di conversione dati e un'applicazione client JMS che può scambiare messaggi con un'applicazione nonJMS utilizzando `JMSBytesMessage`. Lo scambio di un messaggio formattato con un'applicazione nonJMS può avvenire con o senza la chiamata di un'uscita di conversione dati.

Riferimenti correlati

Conversione e tipi di messaggi JMS
La scelta del tipo di messaggio influisce sull'approccio alla conversione del messaggio. L'interazione tra la conversione del messaggio e il tipo di messaggio è descritta per i tipi di messaggio JMS `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`e `JMSBytesMessage`.

Conversione dati del gestore code
La conversione dei dati del gestore code è sempre stata disponibile per le applicazioni nonJMS che ricevono i messaggi dai client JMS . I client JMS che ricevono i messaggi utilizzano anche la conversione dei dati del gestore code, che è facoltativa.

Il gestore code può convertire i dati carattere e numerici nei dati del messaggio utilizzando i valori di `CodedCharacterSetId`, `Encoding`e `Format` impostati per i dati del messaggio. Per le applicazioni nonJMS la funzione di conversione è sempre stata disponibile impostando l'opzione `GetMessage`, `GMO_CONVERT`.

Il gestore code è in grado di convertire i messaggi inviati ai client JMS . La conversione del gestore code è controllata impostando la proprietà di destinazione `WMQ_RECEIVE_CONVERSION`su `WMQ_RECEIVE_CONVERSION_QMGR`o `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`. L'applicazione può modificare l'impostazione della destinazione:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Oppure,

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 34. Abilita conversione dati gestore code

La conversione dei dati del gestore code per un client JMS si verifica quando il client richiama un metodo `consumer.receive` . I dati di testo vengono trasformati in UTF-8 (1208) per impostazione predefinita. I metodi read e get successivi decodificano il testo nei dati ricevuti da UTF-8, creando le primitive di testo Java nella loro codifica Unicode interna. UTF-8 non è l'unica serie di caratteri di destinazione dalla conversione dei dati del gestore code. È possibile scegliere un CCSID differente impostando la proprietà destinazione `WMQ_RECEIVE_CCSID` .

Un'applicazione può anche modificare l'impostazione della destinazione, ad esempio impostandola su 437, DOS - US:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

Oppure,

```
((MQDestination)destination).setReceiveCCSID(437);
```

Figura 35. Imposta set di caratteri codificati di destinazione per la conversione del gestore code

Il motivo per cui si modifica `WMQ_RECEIVE_CCSID` è specializzato; il CCSID scelto non fa differenza per gli oggetti di testo creati nella JVM. Tuttavia, alcune JVM, su alcune piattaforme, potrebbero non essere in grado di gestire la conversione dal CCSID del testo nel messaggio in Unicode. L'opzione fornisce una scelta di CCSID per qualsiasi testo consegnato al client nel messaggio. Alcune piattaforme client JMS hanno avuto problemi con il testo del messaggio consegnato in UTF-8.

Il codice JMS è equivalente al testo in grassetto nel codice C in Figura 36 a pagina 189,

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Figura 36. Frammento di codice da `amqsget0.c`

Nota:

La conversione del gestore code viene eseguita solo sui dati del messaggio che hanno un formato IBM MQ noto. MQSTR, o MQCIH sono esempi di formati noti predefiniti. Un formato noto può anche essere definito dall'utente, purché sia stata fornita un'uscita di conversione dati.

I messaggi creati come JMSTextMessage, JMSMapMessage e JMSStreamMessage, hanno un formato MQSTR e possono essere convertiti dal gestore code.

Concetti correlati

Approcci di conversione dei messaggi JMS
Una serie di approcci di conversione dati sono aperti ai progettisti di applicazioni JMS . Questi approcci non sono esclusivi; alcune applicazioni probabilmente utilizzeranno una combinazione di questi approcci. Se l'applicazione sta scambiando solo testo o messaggi solo con altre applicazioni JMS , normalmente non si considera la conversione dei dati. La conversione dei dati viene eseguita automaticamente da IBM MQ.

Conversione e codifica dei messaggi del client JMS
Vengono elencati i metodi utilizzati per eseguire la conversione e la codifica dei messaggi del client JMS , con esempi di codice di ogni tipo di conversione.

“Richiamo dell'uscita di conversione dati” a pagina 986
Un'uscita di conversione dati è un'uscita scritta dall'utente che riceve il controllo durante l'elaborazione di una chiamata MQGET.

Attività correlate

Scambio di un record formattato con un'applicazione nonJMS
Seguire i passi suggeriti in questa attività per progettare e creare un'uscita di conversione dati e un'applicazione client JMS che può scambiare messaggi con un'applicazione nonJMS utilizzando JMSBytesMessage. Lo scambio di un messaggio formattato con un'applicazione nonJMS può avvenire con o senza la chiamata di un'uscita di conversione dati.

Riferimenti correlati

Conversione e tipi di messaggi JMS
La scelta del tipo di messaggio influisce sull'approccio alla conversione del messaggio. L'interazione tra la conversione del messaggio e il tipo di messaggio è descritta per i tipi di messaggio JMS JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagee JMSBytesMessage.

Scambio di un record formattato con un'applicazione nonJMS
Seguire i passi suggeriti in questa attività per progettare e creare un'uscita di conversione dati e un'applicazione client JMS che può scambiare messaggi con un'applicazione nonJMS utilizzando

JMSBytesMessage. Lo scambio di un messaggio formattato con un'applicazione nonJMS può avvenire con o senza la chiamata di un'uscita di conversione dati.

Prima di iniziare

È possibile progettare una soluzione più semplice per lo scambio di messaggi con un'applicazione nonJMS utilizzando un JMSTextMessage. Eliminare questa possibilità prima di seguire i passi in questa attività.

Informazioni su questa attività

Un client JMS è più semplice da scrivere se non è coinvolto nei dettagli di formattazione dei messaggi JMS scambiati con altri client JMS . Finché il tipo di messaggio è JMSTextMessage, JMSMapMessage, JMSStreamMessageo JMSObjectMessage, IBM MQ si occupa dei dettagli di formattazione del messaggio. IBM MQ si occupa delle differenze nelle codepage e nella codifica numerica su piattaforme differenti.

È possibile utilizzare questi tipi di messaggi per scambiare messaggi con applicazioni nonJMS . Per fare ciò, è necessario comprendere come questi messaggi vengono creati da IBM MQ classes for JMS. È possibile modificare l'applicazione nonJMS per interpretare i messaggi; consultare “Mappatura dei messaggi JMS sui messaggi IBM MQ” a pagina 150.

Un vantaggio dell'utilizzo di uno di questi tipi di messaggi è che la programmazione del client JMS non dipende dal tipo di applicazione con cui sta scambiando i messaggi. Uno svantaggio è che potrebbe richiedere una modifica ad un altro programma e potrebbe non essere possibile modificare l'altro programma.

Un approccio alternativo consiste nel scrivere un'applicazione client JMS che possa gestire i formati di messaggio esistenti. Spesso i messaggi esistenti sono in formato fisso e contengono una combinazione di dati, testo e numeri non formattati. Utilizzare i passi di questa attività e il client JMS di esempio in “Scrittura di classi per incapsulare un layout di record in un JMSBytesMessage” a pagina 193come punto di partenza per la creazione di un client JMS che può scambiare record formattati con applicazioni nonJMS .

Procedura

1. Definire il layout del record oppure utilizzare una delle classi di intestazione IBM MQ predefinite.

   Per la gestione delle intestazioni IBM MQ predefinite, consultare Gestione delle intestazioni del messaggio IBM MQ.

   Figura 37 a pagina 191 è un esempio di layout di record a lunghezza fissa definito dall'utente, che può essere elaborato dal programma di utilità di conversione dati.
2. Creare l'uscita di conversione dati.

   Seguire le istruzioni in Scrittura di un programma di uscita di conversione dati per scrivere un'uscita di conversione dati.

   Per provare l'esempio in “Scrittura di classi per incapsulare un layout di record in un JMSBytesMessage” a pagina 193, denominare l'uscita di conversione dati MYRECORD.
3. Scrivere le classi Java per incapsulare il layout del record e inviare e ricevere il record. Due approcci possibili sono:

   - Scrivere una classe in cui leggere e scrivere il file JMSBytesMessage che contiene il record; consultare “Scrittura di classi per incapsulare un layout di record in un JMSBytesMessage” a pagina 193.
   - Scrivere una classe che estenda com.ibm.mq.header.Header per definire la struttura dati del record; consultare Creazione di classi per nuovi tipi di intestazione.
4. Decidere in quale serie di caratteri codificati scambiare i messaggi.

   Consultare la sezione Scelta di un approccio alla conversione del messaggio: il destinatario fa bene.

5. Configurare la destinazione per lo scambio di messaggi di tipo MQ, senza un'intestazione JMS MQRFH2 .

   Sia la destinazione di invio che quella di ricezione devono essere configurate per lo scambio di messaggi di tipo MQ. È possibile utilizzare la stessa destinazione sia per l'invio che per la ricezione.

   L'applicazione può sovrascrivere la proprietà del corpo del messaggio di destinazione:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   L'esempio in “Scrittura di classi per incapsulare un layout di record in un JMSBytesMessage” a pagina 193 sovrascrive la proprietà del corpo del messaggio di destinazione, verificando che venga inviato un messaggio in stile MQ.
6. Test della soluzione con applicazioni JMS e nonJMS

   Strumenti utili per testare un'uscita di conversione dati sono:

   - Il programma di esempio `amqsgetc0.c` è utile per verificare la ricezione di un messaggio inviato da un client JMS . Consultare le modifiche suggerite per utilizzare l'intestazione di esempio, `RECORD.h`, in Figura 38 a pagina 192. Con le modifiche, `amqsgetc0.c` riceve un messaggio inviato dal client JMS di esempio, `TryMyRecord.java` ; consultare “Scrittura di classi per incapsulare un layout di record in un JMSBytesMessage” a pagina 193.
   - Il programma di esempio IBM MQ browse, `amqsbcg0.c`, è utile per esaminare il contenuto dell'intestazione del messaggio, l'intestazione JMS , `MQRFH2`e il contenuto del messaggio.
   - Il programma **`rfhutil`** , precedentemente disponibile in SupportPac IH03, consente di catturare e memorizzare i messaggi di test nei file e quindi utilizzarli per gestire i flussi di messaggi. I messaggi di output possono anche essere letti e visualizzati in diversi formati. I formati includono due tipi di XML e la corrispondenza con un copybook COBOL. I dati possono essere in EBCDIC o ASCII. È possibile aggiungere un'intestazione RFH2 al messaggio prima che venga inviato.

   Se si tenta di ricevere i messaggi utilizzando il programma di esempio `amqsgetc0.c` modificato e si riceve un errore con codice di errore `2080`, verificare se il messaggio ha un `MQRFH2`. Le modifiche presuppongono che il messaggio sia stato inviato a una destinazione che non specifica `MQRFH2`.

Esempi

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                                MQLONG StructLength;
                                MQLONG Encoding;
                                MQLONG CodeCharSetId;
                                MQCHAR Format[8];
                                MQLONG Flags;
                                MQCHAR RecordData[32];
        };
```

Figura 37. RECORD.h

- Dichiarare la struttura dati RECORD.h

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- Modificare la chiamata MQGET per utilizzare RECORD,

  1. Prima della modifica:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. Dopo la modifica:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- Modificare l'istruzione di stampa,

  1. Da:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. A:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

Figura 38. Modificare amqsget0.c

Concetti correlati

Approcci di conversione dei messaggi JMS

Una serie di approcci di conversione dati sono aperti ai progettisti di applicazioni JMS . Questi approcci non sono esclusivi; alcune applicazioni probabilmente utilizzeranno una combinazione di questi approcci. Se l'applicazione sta scambiando solo testo o messaggi solo con altre applicazioni JMS , normalmente non si considera la conversione dei dati. La conversione dei dati viene eseguita automaticamente da IBM MQ.

Conversione e codifica dei messaggi del client JMS
Vengono elencati i metodi utilizzati per eseguire la conversione e la codifica dei messaggi del client JMS , con esempi di codice di ogni tipo di conversione.

Conversione dati del gestore code
La conversione dei dati del gestore code è sempre stata disponibile per le applicazioni nonJMS che ricevono i messaggi dai client JMS . I client JMS che ricevono i messaggi utilizzano anche la conversione dei dati del gestore code, che è facoltativa.

Programma di utilità per la creazione del codice di uscita di conversione

Riferimenti correlati

Conversione e tipi di messaggi JMS
La scelta del tipo di messaggio influisce sull'approccio alla conversione del messaggio. L'interazione tra la conversione del messaggio e il tipo di messaggio è descritta per i tipi di messaggio JMS `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`e `JMSBytesMessage`.

Scrittura di classi per incapsulare un layout di record in un `JMSBytesMessage`
Lo scopo di questa attività è di esplorare, per esempio, come combinare la conversione dei dati e un layout di record fisso in un `JMSBytesMessage`. Nell'attività, si creano alcune classi Java per scambiare una struttura record di esempio in un `JMSBytesMessage`. È possibile modificare l'esempio per scrivere classi per scambiare altre strutture di record.

Un `JMSBytesMessage` è la scelta migliore del tipo di messaggio JMS per lo scambio di record di tipi di dati misti con programmi nonJMS . Non dispone di ulteriori dati inseriti nel corpo del messaggio dal provider JMS . È quindi la scelta migliore del tipo di messaggio da utilizzare se un programma client JMS interagisce con un programma IBM MQ esistente. La sfida principale nell'utilizzo di un `JMSBytesMessage` viene fornita con la corrispondenza della codifica e della serie di caratteri previsti dall'altro programma. Una soluzione è creare una classe che incapsula il record. Una classe che incapsula la lettura e la scrittura di un `JMSBytesMessage`, per un tipo di record specifico, rende più semplice l'invio e la ricezione di record in formato fisso in un programma JMS . Catturando gli aspetti generali dell'interfaccia in una classe astratta, gran parte della soluzione può essere riutilizzata per formati di record diversi. Diversi formati di record possono essere implementati in classi che estendono la classe generica astratta.

Un approccio alternativo consiste nell'estendere la classe `com.ibm.mq.headers.Header` . La classe `Header` dispone di metodi, come `addMQLONG`, per creare un formato record in modo più dichiarativo. Uno svantaggio dell'utilizzo della classe `Header` è ottenere e l'impostazione degli attributi utilizza un'interfaccia interpretativa più complicata. Entrambi gli approcci risultano nella stessa quantità di codice dell'applicazione.

Un `JMSBytesMessage` può incapsulare solo un formato singolo, oltre a un `MQRFH2`, in un messaggio, a meno che ogni record non utilizzi lo stesso formato, serie di caratteri codificati e codifica. Il formato, la codifica e la serie di caratteri di un `JMSBytesMessage` sono proprietà di tutti i messaggi che seguono `MQRFH2`. L'esempio è scritto supponendo che un `JMSBytesMessage` contenga solo un record utente.

Prima di iniziare

1. Il tuo livello di competenza: devi avere familiarità con la programmazione Java e JMS. Non viene fornita alcuna istruzione sull'impostazione dell'ambiente di sviluppo Java . È vantaggioso aver scritto un programma per scambiare un `JMSTextMessage`, `JMSStreamMessage`o `JMSMapMessage`. È possibile quindi visualizzare le differenze nello scambio di un messaggio utilizzando un `JMSBytesMessage`.
2. L'esempio richiede IBM WebSphere MQ 7.0.
3. L'esempio è stato creato utilizzando la vista Java del workbench Eclipse . Richiede JRE 6.0 o superiore. È possibile utilizzare la prospettiva Java in Esplora risorse di IBM MQ per sviluppare ed eseguire classi Java . In alternativa, utilizzare il proprio ambiente di sviluppo Java .
4. L'utilizzo di Esplora risorse di IBM MQ rende più semplice l'impostazione dell'ambiente di test e il debug rispetto all'utilizzo dei programmi di utilità della riga comandi.

Informazioni su questa attività

Si è guidati attraverso la creazione di due classi: RECORD e MyRecord. Insieme queste due classi racchiudono un record a formato fisso. Hanno metodi per ottenere e impostare gli attributi. Il metodo get legge il record da un JMSBytesMessage e il metodo put scrive un record in JMSBytesMessage.

Lo scopo dell'attività non è creare una classe di qualità di produzione che è possibile riutilizzare. È possibile scegliere di utilizzare gli esempi nell'attività per iniziare a utilizzare le proprie classi. Lo scopo dell'attività è quello di fornire note di guida, principalmente sull'utilizzo di serie di caratteri, formati e codifica, quando si utilizza un JMSBytesMessage. Ogni fase nella creazione delle classi viene spiegata e vengono descritti gli aspetti dell'utilizzo di JMSBytesMessage, a volte trascurati.

La classe RECORD è astratta e definisce alcuni campi comuni per un record utente. I campi comuni sono modellati sul layout di intestazione IBM MQ standard con un campo di tipo eye catcher, una versione e una lunghezza. I campi di codifica, set di caratteri e formato, trovati in molte intestazioni IBM MQ , vengono omessi. Un'altra intestazione non può seguire un formato definito dall'utente. La classe MyRecord , che estende la classe RECORD , lo fa estendendo letteralmente il record con ulteriori campi utente. Un JMSBytesMessage, creato dalle classi, può essere elaborato dall'uscita di conversione dati del gestore code.

“Classi utilizzate per eseguire l'esempio” a pagina 200 include un elenco completo di RECORD e MyRecord. Include anche elenchi delle classi "scaffolding" aggiuntive per verificare RECORD e MyRecord. Le classi extra sono:

TryMyRecord
Il programma principale per testare RECORD e MyRecord.

EndPoint
Una classe astratta che incapsula la sessione, la destinazione e la connessione JMS in una sola classe. La sua interfaccia soddisfa le esigenze di verifica delle classi RECORD e MyRecord . non è un modello di progettazione stabilito per la scrittura di applicazioni JMS .

Nota: La classe Endpoint include questa riga di codice dopo la creazione di una destinazione:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

In V7.0, da V7.0.1.5, è necessario attivare la conversione del gestore code. Per impostazione predefinita, questa funzione è disabilitata. In V7.0, la conversione del gestore code V7.0.1.4 è abilitata per impostazione predefinita e questa riga di codice causa un errore.

MyProducer e MyConsumer
Classi che estendono EndPointe creano un MessageConsumer e un MessageProducer, connessi e pronti ad accettare richieste.

Insieme, tutte le classi costituiscono un'applicazione completa che è possibile creare e sperimentare, per comprendere come utilizzare la conversione dati in un JMSBytesMessage.

Procedura

1. Creare una classe astratta per incapsulare i campi standard in un'intestazione IBM MQ , con un costruttore predefinito. Successivamente, si estende la classe per adattare l'intestazione ai propri requisiti.

```
public abstract class RECORD implements Serializable {
   private static final long serialVersionUID = -1616617232750561712L;
   protected final static int UTF8 = 1208;
   protected final static int MQLONG_LENGTH = 4;
   protected final static int RECORD_STRUCT_ID_LENGTH = 4;
   protected final static int RECORD_VERSION_1 = 1;
   protected final String RECORD_STRUCT_ID = "BLNK";
   protected final String RECORD_TYPE = "BLANK   ";
   private String structID = RECORD_STRUCT_ID;
   private int version = RECORD_VERSION_1;
   private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
```

```
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
```

Nota:

a. Gli attributi, da `structID` a `nextFormat`, sono elencati nell'ordine in cui sono disposti in un'intestazione di messaggio IBM MQ standard.

b. Gli attributi `format`, `messageEncoding`e `messageCharset`, descrivono l'intestazione stessa e non fanno parte dell'intestazione.

c. È necessario decidere se memorizzare il CCSID (coded character set identifier) o la serie di caratteri del record. Java utilizza serie di caratteri e i messaggi IBM MQ utilizzano identificativi di serie di caratteri codificati. Il codice di esempio utilizza serie di caratteri.

d. `int` viene serializzato in `MQLONG` da IBM MQ. `MQLONG` è di 4 byte.

2. Creare i getter e i setter per gli attributi privati.

a) Creare o generare i getter:

```
     public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }
```

b) Creare o generare i setter:

```
     public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

3. Crea un costruttore per creare un'istanza `RECORD` da una `JMSBytesMessage`.

```
  public RECORD(BytesMessage message) throws JMSException, IOException,
         MQDataException {
     super();
     setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
     setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
     byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
     message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
     setStructID(new String(structID, getMessageCharset()));
     setVersion(message.readInt());
     setStructLength(message.readInt());
  }
```

Nota:

a. `messageCharset` e `messageEncoding`vengono catturati dalle proprietà del messaggio, poiché sovrascrivono i valori impostati per la destinazione. `format` non è aggiornato. L'esempio non effettua alcuna verifica degli errori. Se viene richiamato il costruttore

Record(BytesMessage) , si presume che JMSBytesMessage sia un messaggio di tipo RECORD . La linea "setStructID(new String(structID, getMessageCharset()))" imposta l'eye catcher.

b. Le righe di codice che completano i campi deserializzati del metodo nel messaggio, in ordine, aggiornando i valori predefiniti impostati nell'istanza RECORD.

4. Creare un metodo put per scrivere i campi di intestazione in un JMSBytesMessage.

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
       JMSException, UnsupportedEncodingException {
   setHeaderEncoding(myProducer.getEncoding());
   setHeaderCharset(myProducer.getCharset());
   myProducer.setMQClient(true);
   BytesMessage bytes = myProducer.session.createBytesMessage();
   bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
   bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
   bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                        myProducer.getCCSID());
   bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                   + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                   .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
   bytes.writeInt(getVersion());
   bytes.writeInt(getStructLength());
   return bytes;
}
```

Nota:

a. MyProducer incapsula JMS Connection, Destination, Sessione MessageProducer in una singola classe. MyConsumer, utilizzato successivamente, incapsula JMS Connection, Destination, Sessione MessageConsumer in una classe singola.

b. Per un JMSBytesMessage, se la codifica è diversa da Native, la codifica deve essere impostata nel messaggio. La codifica di destinazione viene copiata nell'attributo di codifica del messaggio, JMS_IBM_CHARACTER_SET, e salvata come attributo della classe RECORD .

i) "setMessageEncoding(myProducer.getEncoding());" chiama "(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));" per ottenere la codifica di destinazione.

ii) "Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());" imposta la codifica del messaggio.

c. La serie di caratteri utilizzata per trasformare il testo in byte si ottiene dalla destinazione e viene salvata come attributo della classe RECORD . Non viene impostato nel messaggio, perché non viene utilizzato da IBM MQ classes for JMS quando si scrive un JMSBytesMessage.

Chiamate "messageCharset = myProducer.getCharset();"

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

Richiama la serie di caratteri Java da un CCSID (coded character set identifier).

" CCSID.getCodepage(ccsid) " si trova nel pacchetto com.ibm.mq.headers. ccsid si ottiene da un altro metodo in MyProducer, che interroga la destinazione:

```
public int getCCSID() throws JMSException {
    return (((MQDestination) destination)
             .getIntProperty(WMQConstants.WMQ_CCSID));
}
```

d. "myProducer.setMQClient(true);" sovrascrive l'impostazione di destinazione per il tipo client, forzandola a un IBM MQ MQI client. È possibile che si preferisca omettere questa riga di codice, poiché oscura un errore di configurazione di gestione.

"myProducer.setMQClient(true);" chiama:

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

Il codice ha l'effetto collaterale di impostare lo stile del corpo IBM MQ su non specificato, se deve sovrascrivere un'impostazione di JMS.

Nota:

IBM MQ classes for JMS scrive il formato, la codifica e l'identificativo della serie di caratteri del messaggio nel descrittore del messaggio, MQMD, o nell'intestazione JMS , MQRFH2. Dipende dal fatto che il messaggio abbia un corpo di stile IBM MQ . Non impostare i campi MQMD manualmente.

Esiste un metodo per impostare manualmente le proprietà del descrittore del messaggio. Utilizza le proprietà JMS_IBM_MQMD_* . È necessario impostare la proprietà di destinazione WMQ_MQMD_WRITE_ENABLED per impostare le proprietà JMS_IBM_MQMD_* :

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

È necessario impostare la proprietà di destinazione, WMQ_MQMD_READ_ENABLED, per leggere le proprietà.

Utilizzare JMS_IBM_MQMD_* solo se si assume il controllo completo sull'intero payload del messaggio. A differenza delle proprietà JMS_IBM_* , le proprietà JMS_IBM_MQMD_* non controllano il modo in cui IBM MQ classes for JMS crea un messaggio JMS . È possibile creare le proprietà del descrittore del messaggio che sono in conflitto con le proprietà del messaggio JMS .

e. Le righe di codice che completano il metodo serializzano gli attributi nella classe come campi nel messaggio.

   Gli attributi stringa vengono riempiti con spazi. Le stringhe vengono convertite in byte utilizzando la serie di caratteri definita per il record e troncate alla lunghezza dei campi del messaggio.

5. Completare la classe aggiungendo le importazioni.

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. Creare una classe per estendere la classe RECORD per includere ulteriori campi. Includere un costruttore predefinito.

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

   public MyRecord() {
       super();
       super.setStructID(STRUCT_ID);
       super.setHeaderFormat(FORMAT);
       super.setStructLength(super.getStructLength() + MQLONG_LENGTH
               + DATA_LENGTH);
```

```
    }
```

Nota:

a. La sottoclasse RECORD , MyRecord, personalizza l'elemento eye catcher, il formato e la lunghezza dell'intestazione.

7. Creare o generare i getter e i setter.

a) Creare i getter:

```
     public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .
```

b) Creare i setter:

```
     public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

8. Crea un costruttore per creare un'istanza MyRecord da una JMSBytesMessage.

```
  public MyRecord(BytesMessage message) throws JMSException, IOException,
         MQDataException {
     super(message);
     setFlags(message.readInt());
     byte[] recordData = new byte[DATA_LENGTH];
     message.readBytes(recordData, DATA_LENGTH);
     setRecordData(new String(recordData, super.getMessageCharset()));
  }
```

Nota:

a. I campi che costituiscono il modello di messaggio standard vengono letti per primi dalla classe RECORD .

b. Il testo recordData viene convertito in String utilizzando la proprietà della serie di caratteri del messaggio.

9. Creare un metodo statico per ottenere un messaggio da un consumer e creare una nuova istanza MyRecord .

```
  public static MyRecord get(MyConsumer myConsumer) throws JMSException,
         MQDataException, IOException {
     BytesMessage message = (BytesMessage) myConsumer.receive();
     return new MyRecord(message);
  }
```

Nota:

a. Nell'esempio, per brevità, il costruttore MyRecord(BytesMessage) viene richiamato dal metodo get statico. Generalmente, è possibile separare la ricezione del messaggio dalla creazione di una nuova istanza MyRecord .

10. Creare un metodo put per accodare i campi cliente a JMSBytesMessage contenente un'intestazione del messaggio.

```
  public BytesMessage put(MyProducer myProducer) throws JMSException,
         IOException {
     BytesMessage bytes = super.put(myProducer);
     bytes.writeInt(getFlags());
     bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
```

```
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
```

Nota:

a. Il metodo richiama nel codice la serializzazione degli attributi nella classe MyRecord come campi nel messaggio.

- L'attributo `recordData` `String` viene riempito con spazi vuoti, convertito in byte utilizzando la serie di caratteri definita per il record e troncato alla lunghezza dei campi `RecordData` .

11. Completare la classe aggiungendo le istruzioni include.

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Risultati

- I risultati dell'esecuzione della classe `TryMyRecord` :
  - Invio di un messaggio nella serie di caratteri codificata 37 e utilizzo di un'exit di conversione del gestore code:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - Invio di un messaggio nella serie di caratteri codificati 37 e non utilizzando un'uscita di conversione del gestore code:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- I risultati della modifica della classe `TryMyRecord` per non ricevere il messaggio e per riceverlo invece utilizzando l'esempio `amqsget0.c` modificato. L'esempio modificato accetta un record formattato; consultare Figura 38 a pagina 192 in “Scambio di un record formattato con un'applicazione nonJMS” a pagina 189.
  - Invio di un messaggio nella serie di caratteri codificata 37 e utilizzo di un'exit di conversione del gestore code:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - Invio di un messaggio nella serie di caratteri codificati 37 e non utilizzando un'uscita di conversione del gestore code:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

Per provare l'esempio e provare diverse codepage e un'uscita di conversione dati. Creare le classi Java , configurare IBM MQed eseguire il programma principale, `TryMyRecord` ; consultare “#unique_196/unique_196_Connect_42_Try” a pagina 200.

1. Configurare IBM MQ e JMS per eseguire l'esempio. Le istruzioni sono per eseguire l'esempio su Windows.
   a. Creare un gestore code

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. Creare una coda

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. Crea una directory JNDI

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. Passare alla directory JMS bin.

   Il programma di amministrazione JMS deve essere eseguito da qui. Il percorso è *MQ_INSTALLATION_PATH*`\java\bin`.

   e. Creare le seguenti definizioni JMS in un file denominato `JMSQM1Q1.txt`

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. Eseguire il programma JMSAdmin per creare le risorse JMS

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. È possibile creare, modificare e sfogliare le definizioni create utilizzando Esplora risorse di IBM MQ .
3. Eseguire `TryMyRecord`.

Classi utilizzate per eseguire l'esempio

Le classi elencate nei seguenti blocchi di codice sono disponibili anche in un file compresso. Scaricare jm25529_.zip o jm25529_.tar.gz.

`TryMyRecord`

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
```

```
    }
}
```

RECORD

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
```

```
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
```

```
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
```

```
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

Creazione e configurazione di factory di connessione e destinazioni

Un'applicazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging può creare factory di connessione e destinazioni richiamandoli come oggetti gestiti da uno spazio dei nomi JNDI (Java Naming and Directory Interface), utilizzando le estensioni IBM JMS o utilizzando le estensioni IBM MQ JMS . Un'applicazione può anche utilizzare le estensioni IBM JMS o IBM MQ JMS per impostare le proprietà delle factory di connessione e delle destinazioni.

Le factory di connessione e le destinazioni sono punti di partenza nel flusso della logica di un'applicazione JMS o Jakarta Messaging . Un'applicazione utilizza un oggetto ConnectionFactory per creare una

connessione a un server di messaggistica e utilizza un oggetto coda o argomento come destinazione per inviare messaggi o un'origine da cui ricevere messaggi. Un'applicazione deve quindi creare almeno una factory di connessione e una o più destinazioni. Dopo aver creato una factory di connessione o una destinazione, l'applicazione potrebbe dover configurare l'oggetto impostando una o più proprietà.

In sintesi, un'applicazione può creare e configurare factory di connessione e destinazioni nei modi seguenti:

Utilizzo di JNDI per richiamare gli oggetti gestiti
Un amministratore può utilizzare lo strumento di amministrazione IBM MQ JMS come descritto in Configurazione degli oggetti JMS e Jakarta Messaging utilizzando gli strumenti di gestione, o IBM MQ Explorer come descritto in Configurazione degli oggetti JMS 2.0 utilizzando IBM MQ Explorer, per creare e configurare factory di connessione e destinazioni come oggetti amministrati in uno spazio dei nomi JNDI. Un'applicazione può quindi richiamare gli oggetti gestiti dallo spazio nomi JNDI. Dopo aver richiamato un oggetto gestito, l'applicazione può, se necessario, impostare o modificare una o più delle sue proprietà utilizzando le estensioni IBM JMS o IBM MQ JMS .

Nota: V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, non è possibile gestire JNDI utilizzando IBM MQ Explorer. La gestione JNDI è supportata dalla variante Jakarta Messaging 3.0 di **JMSAdmin**, che è **JMS30Admin**.

Utilizzo delle estensioni IBM JMS
Un'applicazione può utilizzare le estensioni IBM JMS per creare factory di connessione e destinazioni in modo dinamico al runtime. L'applicazione crea innanzitutto un oggetto factory JmsFactory, quindi utilizza i metodi di questo oggetto per creare factory di connessione e destinazioni. Dopo aver creato una factory di connessione o una destinazione, l'applicazione può utilizzare i metodi ereditati dall'interfaccia di contesto JmsPropertyper impostare le relative proprietà. In alternativa, l'applicazione può utilizzare un URI (uniform resource identifier) per specificare una o più proprietà di una destinazione quando crea la destinazione.

Utilizzo delle estensioni IBM MQ JMS
Un'applicazione può anche utilizzare le estensioni IBM MQ JMS per creare in modo dinamico factory di connessione e destinazioni al runtime. L'applicazione utilizza i costruttori forniti per creare factory di connessione e destinazioni. Dopo aver creato un factory di connessione o una destinazione, l'applicazione può utilizzare i metodi dell'oggetto per impostare le relative proprietà. In alternativa, l'applicazione può utilizzare un URI per specificare una o più proprietà di una destinazione quando crea la destinazione.

Attività correlate
Configurazione delle risorse JMS e Jakarta Messaging

Utilizzo di JNDI per richiamare oggetti gestiti in un'applicazione JMS o Jakarta Messaging
Per richiamare gli oggetti gestiti da uno spazio nomi JNDI (Java Naming and Directory Interface), un'applicazione JMS o Jakarta Messaging deve creare un contesto iniziale e quindi utilizzare il metodo lookup () per richiamare gli oggetti.

Prima che un'applicazione possa richiamare gli oggetti gestiti da uno spazio nomi JNDI, un amministratore deve prima creare gli oggetti gestiti.

JMS 2.0 Per JMS 2.0, l'amministratore può utilizzare lo strumento di amministrazione IBM MQ JMS , **JMSAmin**o IBM MQ Explorer per creare e gestire gli oggetti gestiti in uno spazio dei nomi JNDI. Per ulteriori informazioni, consultare Configurazione di factory di connessione e destinazioni in uno spazio dei nomi JNDI.

V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, non è possibile gestire JNDI utilizzando IBM MQ Explorer. La gestione JNDI è supportata dalla variante Jakarta Messaging 3.0 di **JMSAdmin**, che è **JMS30Admin**.

Un server delle applicazioni, generalmente fornisce il proprio repository per gli oggetti gestiti e i propri strumenti per la creazione e la gestione degli oggetti.

Per richiamare gli oggetti gestiti da un namespace JNDI, un'applicazione deve prima creare un contesto iniziale, come mostrato nel seguente esempio:

V 9.3.0 JM 3.0 V 9.3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

In questo codice, le variabili stringa `url` e `icf` hanno i seguenti significati:

URL
L'URL (Uniform Resource Locator) (URL) del servizio directory. L' URL può avere uno dei formati seguenti:

- `ldap://`*`hostname`*`/`*`contextName`* , per un servizio di directory basato su un server LDAP
- `file:/`*`directoryPath`* , per un servizio di directory basato sul file system locale

ICF
Il nome classe del factory di contesto iniziale, che può essere uno dei seguenti valori:

- `com.sun.jndi.ldap.LdapCtxFactory`, per un servizio di directory basato su un server LDAP
- `com.sun.jndi.fscontext.RefFSContextFactory`, per un servizio di directory basato sul file system locale

Notare che alcune combinazioni di un package JNDI e di un fornitore di servizi LDAP (Lightweight Directory Access Protocol) possono causare l'errore LDAP 84. Per risolvere questo problema, inserire la seguente riga di codice prima della chiamata a InitialDirContext ():

```
environment.put(Context.REFERRAL, "throw");
```

Dopo aver ottenuto un contesto iniziale, l'applicazione può recuperare gli oggetti gestiti dallo spazio nomi JNDI utilizzando il metodo lookup (), come mostrato nel seguente esempio:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

Questo codice richiama i seguenti oggetti da un namespace basato su LDAP:

- Un oggetto ConnectionFactory collegato con il nome myCF
- Un oggetto Queue collegato con il nome myQ
- Un oggetto Argomento collegato con nome myT

Per ulteriori informazioni sull'utilizzo di JNDI, consultare la documentazione JNDI fornita da Oracle Corporation.

Attività correlate

Configurazione degli oggetti JMS 2.0 utilizzando IBM MQ Explorer

Configurazione di oggetti JMS e Jakarta Messaging mediante gli strumenti di amministrazione

Configurazione delle risorse JMS 2.0 in WebSphere Application Server

Utilizzo delle estensioni IBM JMS

IBM MQ classes for JMS (JMS 2.0) e IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) contengono ciascuno una serie di estensioni funzionalmente identiche all'API JMS denominata estensioni IBM JMS . Un'applicazione può utilizzare queste estensioni per creare factory di connessione e destinazioni in modo dinamico al runtime e per impostare le proprietà degli oggetti IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Le estensioni possono essere utilizzate con qualsiasi provider di messaggistica.

Le estensioni IBM JMS sono una serie di interfacce e classi nei seguenti package:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Per Jakarta Messaging 3.0, questi pacchetti sono in `com.ibm.jakarta.client.jar`.

JMS 2.0 Per JMS 2.0, questi pacchetti si trovano in `com.ibm.mqjms.jar` o `com.ibm.mq.allclient.jar`.

Queste estensioni forniscono la seguente funzione:

- Un meccanismo basato su factory per la creazione dinamica di factory di connessione e destinazioni in fase di runtime, invece di richiamarli come oggetti gestiti da uno spazio nomi JNDI (Java Naming and Directory Interface)
- Una serie di metodi per l'impostazione delle proprietà degli oggetti IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging
- Una serie di classi di eccezioni con metodi per ottenere informazioni dettagliate su un problema
- Una serie di metodi per il controllo della traccia
- Una serie di metodi per ottenere informazioni sulla versione relative a IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging

Per la creazione dinamica di factory di connessione e destinazioni in fase di runtime e l'impostazione e il richiamo delle relative proprietà, le estensioni IBM JMS forniscono una serie alternativa di interfacce alle estensioni IBM MQ JMS . Tuttavia, mentre le estensioni IBM MQ JMS sono specifiche del provider di messaggistica IBM MQ , le estensioni IBM JMS non sono specifiche di IBM MQ e possono essere utilizzate con qualsiasi provider di messaggistica all'interno dell'architettura a livelli descritta in IBM MQ classes for JMS architecture.

L'interfaccia com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) o com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) contiene le definizioni delle costanti che un'applicazione può utilizzare quando imposta le proprietà degli oggetti IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging utilizzando le estensioni IBM JMS . L'interfaccia contiene costanti per il provider di messaggistica IBM MQ e le costanti JMS che sono indipendenti da qualsiasi provider di messaggistica.

Gli esempi di codice riportati di seguito presuppongono che le seguenti istruzioni di importazione siano incluse nella classe Java :

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0

```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Creazione di factory di connessione e destinazioni

Prima che un'applicazione possa creare factory di connessione e destinazioni utilizzando le estensioni IBM JMS , è necessario creare un oggetto factory JmsFactory. Per creare un oggetto Factory JmsFactory, l'applicazione richiama il metodo getInstance() della classe Factory JmsFactory, come mostrato nel seguente esempio:

V 9.3.0 JM 3.0 V 9.3.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

Il parametro sulla chiamata getInstance() è una costante che identifica il provider di messaggistica IBM MQ come provider di messaggistica scelto. L'applicazione può quindi utilizzare l'oggetto factory JmsFactoryper creare factory di connessione e destinazioni.

Per creare una factory di connessione, l'applicazione richiama il metodo createConnectionFactory () dell'oggetto Factory JmsFactory, come mostrato nel seguente esempio:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

Questa istruzione crea un oggetto JmsConnectionFactory con i valori predefiniti per tutte le relative proprietà, il che significa che l'applicazione si connette al gestore code predefinito in modalità bind. Se si desidera che un'applicazione si connetta in modalità client o a un gestore code diverso dal gestore code predefinito, l'applicazione deve impostare le proprietà appropriate dell'oggetto factory JmsConnectionprima di creare la connessione. Per informazioni su come svolgere questa procedura, consultare “Impostazione delle proprietà degli oggetti IBM MQ classes for JMS” a pagina 212.

La classe factory JmsFactorycontiene anche metodi per creare factory di connessione dei tipi seguenti:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- Factory JmsXAConnection
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

Per creare un oggetto Queue, l'applicazione richiama il metodo createQueue() dell'oggetto Factory JmsFactory, come mostrato nel seguente esempio:

```
JmsQueue q1 = ff.createQueue("Q1");
```

Questa istruzione crea un oggetto JmsQueue con i valori predefiniti per tutte le relative proprietà. L'oggetto rappresenta una coda IBM MQ denominata Q1 appartenente al gestore code locale. Questa coda può essere una coda locale, una coda alias o una definizione di coda remota.

Il metodo createQueue() può anche accettare un URI (uniform resource identifier) della coda come parametro. Un URI della coda è una stringa che specifica il nome di una coda IBM MQ e, facoltativamente, il nome del gestore code proprietario della coda e una o più proprietà dell'oggetto JmsQueue . La seguente istruzione contiene un esempio di URI della coda:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

L'oggetto JmsQueue creato da questa istruzione rappresenta una coda IBM MQ denominata Q2 di proprietà del gestore code QM2e tutti i messaggi inviati a questa destinazione sono persistenti e hanno una priorità di 5. Per ulteriori informazioni sugli URI della coda, consultare “URI (Uniform Resource Identifier)” a pagina 225. Per una modalità alternativa di impostazione delle proprietà di un oggetto JmsQueue , consultare “Impostazione delle proprietà degli oggetti IBM MQ classes for JMS” a pagina 212.

Per creare un oggetto Topic, un'applicazione può utilizzare il metodo createTopic() dell'oggetto Factory JmsFactory, come mostrato nel seguente esempio:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

Questa istruzione crea un oggetto JmsTopic con i valori predefiniti per tutte le sue proprietà. L'oggetto rappresenta un argomento chiamato Sport / Calcio/Risultati.

Il metodo createTopic() può accettare anche un URI argomento come parametro. Un URI dell'argomento è una stringa che specifica il nome di un argomento e, facoltativamente, una o più proprietà dell'oggetto JmsTopic . Le seguenti istruzioni contengono un esempio di URI argomento:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

L'oggetto JmsTopic creato da queste istruzioni rappresenta un argomento denominato Sport / Tennis/ Results e tutti i messaggi inviati a questa destinazione sono non persistenti e hanno una priorità 0. Per ulteriori informazioni sugli URI argomento, consultare “URI (Uniform Resource Identifier)” a pagina 225. Per un modo alternativo di impostare le proprietà di un oggetto JmsTopic , consultare “Impostazione delle proprietà degli oggetti IBM MQ classes for JMS” a pagina 212.

Dopo che un'applicazione ha creato una factory di connessione o una destinazione, tale oggetto può essere utilizzato solo con il fornitore di messaggistica selezionato.

Impostazione delle proprietà degli oggetti IBM MQ classes for JMS

Per impostare le propriet ... degli oggetti IBM MQ classes for JMS utilizzando l'estensione IBM JMS , un'applicazione utilizza i metodi dell'interfaccia com.ibm.msg.client.JmsPropertyContext . Allo stesso modo, per impostare le propriet ... degli oggetti IBM MQ classes for Jakarta Messaging utilizzando le estensioni IBM JMS , un'applicazione utilizza i metodi dell'interfaccia com.ibm.msg.jakarta.client.JmsPropertyContext .

Per ciascun tipo di dati Java , l'interfaccia di contesto JmsPropertycontiene un metodo per impostare il valore di una proprietà con quel tipo di dati e un metodo per ottenere il valore di una proprietà con quel tipo di dati. Ad esempio, un'applicazione richiama il metodo setIntProperty () per impostare una proprietà con un valore intero e richiama il metodo getIntProperty () per ottenere una proprietà con un valore intero.

Le istanze delle classi nei package com.ibm.mq.jms e com.ibm.mq.jakarta.jms ereditano i metodi delle corrispondenti interfacce di contesto JmsProperty. Un'applicazione può quindi utilizzare questi metodi per impostare le proprietà di oggetti MQConnectionFactory, MQQueue e MQTopic.

Quando un'applicazione crea un oggetto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , tutte le proprietà con valori predefiniti vengono impostate automaticamente. Quando un'applicazione imposta una proprietà, il nuovo valore sostituisce qualsiasi valore precedente della proprietà. Una volta impostata una proprietà, non è possibile eliminarla, ma è possibile modificarne il valore.

Se un'applicazione tenta di impostare una proprietà su un valore non valido per la proprietà, IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging genera un'eccezione JMSException. Se un'applicazione tenta di ottenere una proprietà che non è stata impostata, il comportamento è quello descritto nella specifica JMS . IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging generano un'eccezione NumberFormatper i tipi di dati primitivi e restituiscono un valore null per i tipi di dati di riferimento.

Oltre alle proprietà predefinite di un oggetto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , un'applicazione può impostare le proprie proprietà. Queste proprietà definite dall'applicazione vengono ignorate da IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Per ulteriori informazioni relative alle proprietà degli oggetti IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging , consultare Proprietà degli oggetti IBM MQ classes for JMS.

Il seguente codice è un esempio di come impostare le proprietà utilizzando le estensioni IBM JMS . Il codice imposta cinque proprietà di una factory di connessione.

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

L'effetto dell'impostazione di queste proprietà è che l'applicazione si connette al gestore code QM1 in modalità client, utilizzando un canale MQI denominato QM1.SVR. Il gestore code è in esecuzione su un sistema con il nome host HOST1e il listener per il gestore code è in attesa nella porta numero 1415. Questa connessione e altre connessioni del gestore code associate alle sessioni al di sotto di essa, hanno il nome dell'applicazione "Applicazione personale" associato ad esse.

Nota: I gestori code in esecuzione su piattaforme z/OS non supportano l'impostazione dei nomi delle applicazioni, pertanto questa impostazione viene ignorata.

L'interfaccia di contesto JmsPropertycontiene anche il metodo setObjectProperty (), che un'applicazione può utilizzare per impostare le proprietà. Il secondo parametro del metodo è un oggetto che incapsula il valore della proprietà. Ad esempio, il seguente codice crea un oggetto Integer che incapsula il numero intero 1415 e quindi richiama setObjectProperty () per impostare la proprietà PORT di una factory di connessione sul valore 1415:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

Questo codice è quindi equivalente alla seguente istruzione:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

Al contrario, il metodo getObjectProperty () restituisce un oggetto che incapsula il valore di una proprietà.

Conversione implicita di un valore di proprietà da un tipo di dati a un altro

Quando un'applicazione utilizza un metodo dell'interfaccia di contesto JmsPropertyper impostare o richiamare la proprietà di un oggetto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , il valore della proprietà può essere implicitamente convertito da un tipo di dati ad un altro.

Ad esempio, la seguente istruzione imposta la proprietà PRIORITY dell'oggetto JmsQueue q1:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

La proprietà PRIORITY ha un valore intero, quindi la chiamata setStringProperty () converte implicitamente la stringa "5" (il valore di origine) nel numero intero 5 (il valore di destinazione), che diventa il valore della proprietà PRIORITY.

Al contrario, la seguente istruzione richiama la proprietà PRIORITY dell'oggetto JmsQueue q1:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

Il numero intero 5 (il valore di origine), che è il valore della proprietà PRIORITY, viene implicitamente convertito nella stringa "5" (il valore di destinazione) dalla chiamata getStringProperty ().

Le conversioni supportate da IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono riportate in Tabella 34 a pagina 214.

Tabella 34. Conversioni supportate da un tipo di dati a un altro

Tipo di dati di origine	**Tipi di dati di destinazione supportati**
booleano	Stringa
byte	int, long, short, String
char	Stringa
doppio	Stringa
mobile	double, Stringa
int	long, stringa
lungo	Stringa
breve	int, long, Stringa
Stringa	booleano, byte, double, float, int, long, short

Le regole generali che disciplinano le conversioni supportate sono le seguenti:

- I valori numerici possono essere convertiti da un tipo di dati ad un altro purché non vengano persi dati durante la conversione. Ad esempio, un valore con tipo di dati `int` può essere convertito in un valore con tipo di dati `long`, ma non può essere convertito in un valore con tipo di dati `short`.
- Un valore di qualsiasi tipo di dati può essere convertito in una stringa.
- Una stringa può essere convertita in un valore di qualsiasi altro tipo di dati (tranne `char`) purché la stringa sia nel formato corretto per la conversione. Se un'applicazione tenta di convertire una stringa che non è nel formato corretto, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging generano un'eccezione NumberFormat.
- Se un'applicazione tenta una conversione non supportata, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging generano un'eccezione MessageFormat.

Le regole specifiche per convertire un valore da un tipo di dati ad un altro sono le seguenti:

- Quando si converte un valore booleano in una stringa, il valore `true` viene convertito nella stringa "true"e il valore `false` viene convertito nella stringa "false".
- Quando si converte una stringa in un valore booleano, la stringa "true" (non sensibile al maiuscolo / minuscolo) viene convertita in `true`e la stringa "false" (non sensibile al maiuscolo / minuscolo) viene convertita in `false`. Qualsiasi altra stringa viene convertita in `false`.
- Quando si converte una stringa in un valore con tipo di dati `byte`, `int`, `long`o `short`, la stringa deve avere il seguente formato:

  [*spazi*] [*segno*] *cifre*

  I significati dei componenti della stringa sono i seguenti:

  spazi vuoti
  Caratteri vuoti iniziali facoltativi.

  firmare
  Un segno più (+) o meno (-) facoltativo.

cifre
Una sequenza contigua di cifre (0-9). Deve essere presente almeno una cifra.

Dopo la sequenza di cifre, la stringa può contenere altri caratteri che non sono cifre, ma la conversione si arresta non appena viene raggiunto il primo di questi caratteri. Si presuppone che la stringa rappresenti un numero intero decimale.

Se la stringa non è nel formato corretto, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging generano un'eccezione NumberFormat.

- Quando si converte una stringa in un valore con tipo di dati `double` o `float`, la stringa deve avere il formato seguente:

  [*blank*] [*sign*] *cifre* [*e_char* [*e_sign*] *e_digits*]

  I significati dei componenti della stringa sono i seguenti:

  spazi vuoti
  Caratteri vuoti iniziali facoltativi.

  firmare
  Un segno più (+) o meno (-) facoltativo.

  cifre
  Una sequenza contigua di cifre (0-9). Deve essere presente almeno una cifra.

  e_car
  Un carattere esponente, che è *E* o *e*.

  e_firma
  Un segno più (+) o meno (-) facoltativo per l'esponente.

  e_cifre
  Una sequenza contigua di cifre (0-9) per l'esponente. Deve essere presente almeno una cifra se la stringa contiene un carattere esponente.

  Dopo la sequenza di cifre o i caratteri facoltativi che rappresentano un esponente, la stringa può contenere altri caratteri che non sono cifre, ma la conversione si interrompe non appena viene raggiunto il primo di questi caratteri. Si presuppone che la stringa rappresenti un numero a virgola mobile decimale con un esponente che sia una potenza di 10.

  Se la stringa non è nel formato corretto, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging generano un'eccezione NumberFormat.

- Durante la conversione di un valore numerico (incluso un valore con tipo di dati `byte`) in una stringa, il valore viene convertito nella rappresentazione della stringa del valore come numero decimale, non la stringa contenente il carattere ASCII per tale valore. Ad esempio, il numero intero 65 viene convertito nella stringa "65", non nella stringa "A".

Impostazione di più di una proprietà in una singola chiamata

L'interfaccia di contesto JmsPropertycontiene anche il metodo setBatchProperties (), che un'applicazione può utilizzare per impostare più di una proprietà in una singola chiamata. Il parametro del metodo è un oggetto mappa che incapsula una serie di coppie nome - valore della proprietà.

Ad esempio, il codice riportato di seguito utilizza il metodo setBatchProperties () per impostare le stesse cinque proprietà di un factory di connessione come mostrato in “Impostazione delle proprietà degli oggetti IBM MQ classes for JMS” a pagina 212. Il codice crea un'istanza della classe HashMap , che implementa l'interfaccia Map.

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

Il secondo parametro del metodo Map.put() deve essere un oggetto. Pertanto, un valore di proprietà con un tipo di dati primitivo deve essere incapsulato all'interno di un oggetto o rappresentato da una stringa, come mostrato nell'esempio.

Il metodo setBatchProperties () convalida ciascuna proprietà. Se il metodo setBatchProperties () non può impostare una proprietà perché, ad esempio, il suo valore non è valido, nessuna delle proprietà specificate viene impostata.

Nomi e valori delle proprietà

Se un'applicazione utilizza i metodi dell'interfaccia di contesto JmsPropertyappropriata per impostare e richiamare le proprietà degli oggetti IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , l'applicazione può specificare i nomi e i valori delle proprietà in uno dei seguenti modi. Ognuno degli esempi di accompagnamento mostra come impostare la proprietà PRIORITY dell'oggetto JmsQueue q1 in modo che un messaggio inviato alla coda abbia la priorità specificata nella chiamata send ().

Utilizzo dei nomi e valori delle proprietà definiti come costanti nell'interfaccia com.ibm.msg.client.wmq.WMQConstants

La seguente istruzione è un esempio di come specificare i nomi e valori delle proprietà in questo modo:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

Utilizzo dei nomi e dei valori delle proprietà che possono essere utilizzati negli URI (uniform resource identifier) della coda e dell'argomento

La seguente istruzione è un esempio di come specificare i nomi e valori delle proprietà in questo modo:

```
q1.setIntProperty("priority", -2);
```

Solo i nomi e valori delle proprietà delle destinazioni possono essere specificati in questo modo.

Utilizzo dei valori e dei nomi proprietà riconosciuti dallo strumento di amministrazione IBM MQ JMS

La seguente istruzione è un esempio di come specificare i nomi e valori delle proprietà in questo modo:

```
q1.setStringProperty("PRIORITY", "APP");
```

Anche la forma breve del nome della proprietà è accettabile, come mostrato nella seguente istruzione:

```
q1.setStringProperty("PRI", "APP");
```

Quando un'applicazione ottiene una proprietà, il valore restituito dipende dal modo in cui l'applicazione specifica il nome della proprietà. Ad esempio, se un'applicazione specifica la costante WMQConstants.WMQ_PRIORITY come nome della proprietà, il valore restituito è il numero intero -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

Lo stesso valore viene restituito se l'applicazione specifica la stringa "priority" come nome della proprietà:

```
int n2 = getIntProperty("priority");
```

Tuttavia, se l'applicazione specifica la stringa "PRIORITY" o "PRI" come nome della proprietà, il valore restituito è la stringa "APP":

```
String s1 = getStringProperty("PRI");
```

Internamente, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging memorizzano i valori e i nomi delle proprietà come valori letterali definiti nell'interfaccia WMQConstants corrispondente. Questo è il formato canonico definito per i nomi e i valori delle proprietà. Come regola generale, se un'applicazione imposta le proprietà utilizzando uno degli altri due modi per specificare i nomi e i valori delle proprietà, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging devono convertire i nomi e i valori dal formato di input specificato nel formato canonico. Allo stesso modo, se un'applicazione ottiene le proprietà utilizzando uno degli altri due modi per specificare i nomi e i valori delle proprietà, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging devono convertire i nomi dal formato di immissione specificato nel formato canonico e convertire i valori dal formato canonico nel formato di emissione richiesto. L'esecuzione di queste conversioni potrebbe avere implicazioni per le prestazioni.

I nomi e i valori delle proprietà restituiti dalle eccezioni, nei file di traccia o nel log di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , sono sempre in formato canonico.

Utilizzo dell'interfaccia Mappa

L'interfaccia di contesto JmsPropertyestende l'interfaccia java.util.Map . Un'applicazione può quindi utilizzare i metodi dell'interfaccia Mappa per accedere alle proprietà di un oggetto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

Ad esempio, il seguente codice stampa i nomi e i valori di tutte le proprietà di una factory di connessione. Il codice utilizza solo i metodi dell'interfaccia Mappa per ottenere nomi e valori delle proprietà.

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

L'utilizzo dei metodi dell'interfaccia Mappa non ignora le conversioni o le convalide delle proprietà.

Utilizzo delle estensioni IBM MQ JMS
IBM MQ classes for JMS contiene una serie di estensioni all'API JMS denominate estensioni IBM MQ JMS . Un'applicazione può utilizzare queste estensioni per creare factory di connessione e destinazioni in modo dinamico al runtime e per impostare le proprietà delle factory di connessione e delle destinazioni.

IBM MQ classes for JMS contiene una serie di classi nei package com.ibm.jms e com.ibm.mq.jms. Queste classi implementano le interfacce JMS e contengono le estensioni IBM MQ JMS . Gli esempi di codice che seguono presuppongono che questi pacchetti siano stati importati dalle seguenti istruzioni:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Un'applicazione può utilizzare le estensioni IBM MQ JMS per eseguire le seguenti funzioni:

- Creare le factory di connessione e le destinazioni in modo dinamico al runtime, invece di richiamarle come oggetti gestiti da uno spazio dei nomi JNDI (Java Naming and Directory Interface)
- Impostare le proprietà delle factory di connessione e delle destinazioni

Creazione factory di connessione

Per creare un factory di connessioni, un'applicazione può utilizzare il costruttore MQConnectionFactory , come mostrato nel seguente esempio:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

Questa istruzione crea un oggetto MQConnectionFactory con i valori predefiniti per tutte le sue proprietà, il che significa che l'applicazione si connette al gestore code predefinito in modalità di bind. Se si desidera che un'applicazione si connetta in modalità client o a un gestore code diverso dal gestore code predefinito, l'applicazione deve impostare le proprietà appropriate dell'oggetto MQConnectionFactory prima di creare la connessione. Per informazioni su come svolgere questa procedura, consultare “Impostazione delle proprietà delle factory di connessione” a pagina 218.

Un'applicazione può creare factory di connessione dei seguenti tipi in modo simile:

- Factory di MQQueueConnection
- Factory MQTopicConnection
- MQXAConnectionFactory
- Factory MQXAQueueConnection
- Factory MQXATopicConnection

Impostazione delle proprietà delle factory di connessione

Un'applicazione pu impostare le propriet ... di una produzione connessioni richiamando i metodi appropriati della produzione connessioni. Il factory di connessione può essere un oggetto gestito o un oggetto creato dinamicamente in fase di runtime.

Si consideri il seguente codice, ad esempio:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

Questo codice crea un oggetto MQConnectionFactory e imposta cinque proprietà dell'oggetto. L'effetto dell'impostazione di queste proprietà è che l'applicazione si connette al gestore code QM1 in modalità client utilizzando un canale MQI denominato QM1.SVR. Il gestore code è in esecuzione su un sistema con il nome host HOST1e il listener per il gestore code è in attesa nella porta numero 1415.

Un'applicazione che utilizza una connessione in tempo reale a un broker può utilizzare solo lo stile di pubblicazione / sottoscrizione della messaggistica. Non può utilizzare lo stile point-to-point della messaggistica.

Sono valide solo alcune combinazioni di proprietà di una produzione connessioni. Per informazioni sulle combinazioni valide, consultare Dipendenze tra proprietà di oggetti IBM MQ classes for JMS.

Per ulteriori informazioni sulle proprietà di una factory di connessione e sui metodi utilizzati per impostarne le proprietà, vedere Proprietà degli oggetti IBM MQ classes for JMS.

Creazione di destinazioni

Per creare un oggetto Queue, un'applicazione può utilizzare il costruttore MQQueue, come mostrato nel seguente esempio:

```
MQQueue q1 = new MQQueue("Q1");
```

Questa istruzione crea un oggetto MQQueue con i valori predefiniti per tutte le relative proprietà. L'oggetto rappresenta una coda IBM MQ denominata Q1 appartenente al gestore code locale. Questa coda può essere una coda locale, una coda alias o una definizione di coda remota.

Un formato alternativo del costruttore MQQueue ha due parametri, come mostrato nel seguente esempio:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

L'oggetto MQQueue creato da questa istruzione rappresenta una coda IBM MQ denominata Q2 di proprietà del gestore code QM2. Il gestore code identificato in questo modo può essere il gestore code locale o un gestore code remoto. Se si tratta di un gestore code remoto, IBM MQ deve essere configurato in modo che, quando l'applicazione invia un messaggio a questa destinazione, WebSphere MQ possa instradare il messaggio dal gestore code locale al gestore code remoto.

Il costruttore MQQueue può anche accettare un URI (uniform resource identifier) della coda come singolo parametro. Un URI della coda è una stringa che specifica il nome di una coda IBM MQ e, facoltativamente, il nome del gestore code proprietario della coda e una o più proprietà dell'oggetto MQQueue. La seguente istruzione contiene un esempio di URI della coda:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

L'oggetto MQQueue creato da questa istruzione rappresenta una coda IBM MQ denominata Q3 di proprietà del gestore code QM3e tutti i messaggi inviati a questa destinazione sono persistenti e hanno una priorità di 5. Per ulteriori informazioni sugli URI della coda, consultare “URI (Uniform Resource Identifier)” a pagina 225. Per un modo alternativo di impostare le proprietà di un oggetto MQQueue, consultare “Impostazione delle proprietà delle destinazioni” a pagina 219.

Per creare un oggetto Argomento, un'applicazione può utilizzare il costruttore MQTopic, come mostrato nel seguente esempio:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

Questa istruzione crea un oggetto MQTopic con i valori predefiniti per tutte le relative proprietà. L'oggetto rappresenta un argomento chiamato Sport / Calcio/Risultati.

Il costruttore MQTopic può anche accettare un URI argomento come parametro. Un URI argomento è una stringa che specifica il nome di un argomento e, facoltativamente, una o più proprietà dell'oggetto MQTopic. La seguente istruzione contiene un esempio di un URI argomento:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

L'oggetto MQTopic creato da questa istruzione rappresenta un argomento denominato Sport / Tennis/ Results e tutti i messaggi inviati a questa destinazione sono non persistenti e hanno una priorità 0. Per ulteriori informazioni sugli URI argomento, consultare “URI (Uniform Resource Identifier)” a pagina 225. Per un modo alternativo di impostare le proprietà di un oggetto MQTopic, consultare “Impostazione delle proprietà delle destinazioni” a pagina 219.

Impostazione delle proprietà delle destinazioni

Un'applicazione può impostare le proprietà di una destinazione richiamando i metodi appropriati della destinazione. La destinazione può essere un oggetto gestito o un oggetto creato dinamicamente al runtime.

Si consideri il seguente codice, ad esempio:

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

Questo codice crea un oggetto MQQueue e imposta due proprietà dell'oggetto. L'effetto dell'impostazione di queste proprietà è che tutti i messaggi inviati alla destinazione sono persistenti e hanno una priorità di 5.

Un'applicazione può impostare le proprietà dell'oggetto MQTopic in modo simile, come mostrato nel seguente esempio:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

```
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

Questo codice crea un oggetto MQTopic e quindi imposta due proprietà dell'oggetto. L'effetto dell'impostazione di queste proprietà è che tutti i messaggi inviati alla destinazione sono non persistenti e hanno una priorità pari a 0.

Per ulteriori informazioni relative alle proprietà di una destinazione e ai metodi utilizzati per impostarne le proprietà, consultare Proprietà degli oggetti IBM MQ classes for JMS.

Linux AIX *Connessione a IBM MQ da una applicazione JMS*

Per creare un collegamento, un'applicazione JMS utilizza un oggetto **ConnectionFactory** per creare un oggetto **Connection** , quindi avvia il collegamento.

Per JMS 2.0 e versioni successive, le applicazioni generalmente si collegano a un provider di messaggistica utilizzando un oggetto **ConnectionFactory** e il metodo `createContext()` .

Nelle versioni precedenti di JMS, era necessario utilizzare prima `createConnection` per creare un oggetto **Connection** , quindi avviare la chiamata di connessione `getSession()` per creare un oggetto **Session** che potesse eseguire operazioni di messaggistica.

Un oggetto **JMSContext** incapsula in modo efficace sia gli oggetti **Connection** che **Session** . Se si desidera utilizzare l'approccio tradizionale e creare direttamente la connessione e gli oggetti sessione, consultare “Creazione di una connessione in una applicazione JMS” a pagina 220 e “Creazione di una sessione in una applicazione JMS” a pagina 221.

Per creare un oggetto **JMSContext** , un'applicazione utilizza un metodo `createContext()` di oggetto **ConnectionFactory** , come mostrato nel seguente esempio:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

Quando viene creata una connessione JMS , IBM MQ classes for JMS crea un handle di connessione (Hconn) e avvia una conversazione con il gestore code.

Nota: Tenere presente che l'ID processo dell'applicazione viene utilizzato come identità utente predefinita da trasmettere al gestore code. Se l'applicazione è in esecuzione in modalità di trasporto client, questo ID processo deve esistere, con le autorizzazioni pertinenti, sul server. Se si desidera utilizzare un'identità diversa, utilizzare il metodo `createConnection(username,password)` .

V 9.3.5 Questo meccanismo può essere utilizzato anche per fornire un token di autenticazione, consultare Ottenimento di un token di autenticazione dall'emittente del token scelto.

JMS 1.0 *Creazione di una connessione in una applicazione JMS*

Per creare una connessione in JMS 1.0, un'applicazione JMS utilizza un oggetto ConnectionFactory per creare un oggetto Connection e quindi avvia la connessione.

Per creare un oggetto Connection, un'applicazione utilizza il metodo createConnection() di un oggetto ConnectionFactory , come mostrato nel seguente esempio:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

Quando viene creata una connessione JMS , IBM MQ classes for JMS crea un handle di connessione (Hconn) e avvia una conversazione con il gestore code.

L'interfaccia factory QueueConnectione l'interfaccia factory TopicConnectionereditano il metodo createConnection() dall'interfaccia di ConnectionFactory . È quindi possibile utilizzare il metodo createConnection() per creare un oggetto specifico del dominio, come mostrato nel seguente esempio:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

Questo frammento di codice crea un oggetto QueueConnection . Un'applicazione può ora eseguire un'operazione indipendente dal dominio su questo oggetto o un'operazione applicabile solo al dominio point - to - point. Tuttavia, se l'applicazione tenta di eseguire un'operazione applicabile solo al dominio di pubblicazione / sottoscrizione, viene generata un'eccezione IllegalStatecon il seguente messaggio:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

Ciò si verifica perché la connessione è stata creata da un factory di connessione specifico del dominio.

Nota: Tenere presente che l'ID processo dell'applicazione viene utilizzato come identità utente predefinita da trasmettere al gestore code. Se l'applicazione è in esecuzione in modalità di trasporto client, questo ID processo deve esistere, con le autorizzazioni pertinenti, sul server. Se si desidera utilizzare un'identità differente, utilizzare il metodo createConnection(nome utente, password).

La specifica JMS indica che una connessione viene creata nello stato `stopped` . Fino a quando non viene avviata una connessione, un utente di messaggi associato alla connessione non può ricevere alcun messaggio. Per avviare una connessione, un'applicazione utilizza il metodo start () di un oggetto Connection, come mostrato nel seguente esempio:

```
connection.start();
```

V 9.3.5 Questo meccanismo può essere utilizzato anche per fornire un token di autenticazione, consultare Ottenimento di un token di autenticazione dall'emittente del token scelto.

JMS 1.0 *Creazione di una sessione in una applicazione JMS*

Per creare una sessione in JMS 1.0, un'applicazione JMS utilizza il metodo createSession() di un oggetto Connection.

Il metodo createSession() ha due parametri:

1. Un parametro che specifica se la sessione è transattiva o meno
2. Un parametro che specifica la modalità di riconoscimento per la sessione

Ad esempio, il seguente codice crea una sessione che non è sottoposta a transazione e ha una modalità di riconoscimento AUTO_ACKNOWLEDGMENT:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Quando viene creata la sessione JMS , IBM MQ classes for JMS crea un handle di connessione (Hconn) e avvia una conversazione con il gestore code.

Un oggetto Session e qualsiasi oggetto MessageProducer o MessageConsumer da esso creato, non possono essere utilizzati contemporaneamente da thread differenti di un'applicazione a più thread. Il modo più semplice per garantire che questi oggetti non vengano utilizzati contemporaneamente è creare un oggetto Session separato per ciascun thread.

V 9.3.5 Questo meccanismo può essere utilizzato anche per fornire un token di autenticazione, consultare Ottenimento di un token di autenticazione dall'emittente del token scelto.

Sessioni transazionali in applicazioni JMS
Le applicazioni JMS possono eseguire transazioni locali creando prima una sessione transattiva. Un'applicazione può eseguire il commit o il rollback di una transazione.

Le applicazioni JMS possono eseguire transazioni locali. Una transazione locale è una transazione che implica modifiche solo alle risorse del gestore code a cui è connessa l'applicazione. Per eseguire le transazioni locali, un'applicazione deve prima creare una sessione sottoposta a transazione richiamando il metodo createSession() di un oggetto Connection, specificando come parametro che la sessione viene sottoposta a transazione. Successivamente, tutti i messaggi inviati e ricevuti all'interno della sessione vengono raggruppati in una sequenza di transazioni. Una transazione termina quando l'applicazione esegue il commit o il rollback dei messaggi inviati e ricevuti dall'inizio della transazione.

Per eseguire il commit di una transazione, un'applicazione richiama il metodo commit () dell'oggetto Session. Quando viene eseguito il commit di una transazione, tutti i messaggi inviati all'interno della transazione diventano disponibili per il recapito ad altre applicazioni e tutti i messaggi ricevuti all'interno della transazione vengono riconosciuti in modo che il server di messaggistica non tenti di consegnarli nuovamente all'applicazione. Nel dominio point - to - point, il server di messaggistica rimuove anche i messaggi ricevuti dalle code.

Per eseguire il rollback di una transazione, un'applicazione richiama il metodo rollback () dell'oggetto Session. Quando viene eseguito il rollback di una transazione, tutti i messaggi inviati all'interno della transazione vengono eliminati dal server di messaggistica e tutti i messaggi ricevuti all'interno della transazione diventano nuovamente disponibili per la consegna. Nel dominio point - to - point, i messaggi ricevuti vengono reinseriti nelle loro code e diventano nuovamente visibili ad altre applicazioni.

Una nuova transazione viene avviata automaticamente quando un'applicazione crea una sessione sottoposta a transazione o richiama il metodo commit () o rollback (). Pertanto, una sessione transazionale ha sempre una transazione attiva.

Quando un'applicazione chiude una sessione transatta, si verifica un rollback implicito. Quando un'applicazione chiude una connessione, si verifica un rollback implicito per tutte le sessioni transazionali della connessione.

Se un'applicazione termina senza chiudere una connessione, si verifica anche un rollback implicito per tutte le sessioni transazionali della connessione.

Una transazione è interamente contenuta all'interno di una sessione transatta. Una transazione non può estendere le sessioni. Ciò significa che non è possibile per un'applicazione inviare e ricevere messaggi in due o più sessioni transazionali e quindi eseguire il commit o il rollback di tutte queste azioni come una singola transazione.

Modalità di riconoscimento delle sessioni JMS
Ogni sessione non sottoposta a transazione dispone di una modalità di riconoscimento che determina il modo in cui vengono riconosciuti i messaggi ricevuti dall'applicazione. Sono disponibili tre modalità di riconoscimento e la scelta della modalità di riconoscimento influisce sulla progettazione dell'applicazione.

Se non viene eseguita la transazione di una sessione, il modo in cui vengono riconosciuti i messaggi ricevuti dall'applicazione è determinato dalla modalità di riconoscimento della sessione. Le tre modalità di riconoscimento sono descritte nei seguenti paragrafi:

AUTO_RICONOSCIMENTO

La sessione riconosce automaticamente ogni messaggio ricevuto dall'applicazione.

Se i messaggi vengono consegnati in modo sincrono all'applicazione, la sessione conferma la ricezione di un messaggio ogni volta che una chiamata di ricezione viene completata correttamente. Se i messaggi vengono consegnati in modo asincrono, la sessione conferma la ricezione di un messaggio ogni volta che una chiamata al metodo onMessage() di un listener di messaggi viene completata correttamente.

Se l'applicazione riceve correttamente un messaggio, ma un errore impedisce il verificarsi del riconoscimento, il messaggio diventa nuovamente disponibile per la consegna. L'applicazione deve quindi essere in grado di gestire un messaggio che viene riconsegnato.

DUPS_OK_RICONOSCI

La sessione riconosce i messaggi ricevuti dall'applicazione nel momento in cui viene selezionata.

L'utilizzo di questa modalità di riconoscimento riduce la quantità di lavoro che la sessione deve eseguire, ma un malfunzionamento che impedisce la conferma di ricezione del messaggio potrebbe causare la ridisponibilità di più di un messaggio per la consegna. L'applicazione deve quindi essere in grado di gestire i messaggi che vengono riconsegnati.

Limitazione: Nelle modalità AUTO_RICONOSCI e DUPS_OK_TANTO, JMS non supporta un'applicazione che genera un'eccezione non gestita in un listener di messaggi. Ciò significa che i messaggi vengono sempre riconosciuti quando il listener dei messaggi viene restituito, indipendentemente dal fatto che sia stato elaborato correttamente (a condizione che gli errori non siano irreversibili e non impediscano all'applicazione di continuare). Se si richiede un controllo più fine della conferma di ricezione del messaggio, utilizzare le modalità CLIENT_ACKNOWLEDGEMENT o transazionali, che forniscono all'applicazione il controllo completo delle funzioni di conferma di ricezione.

CLIENT_RICONOSCIMENTO

L'applicazione riconosce i messaggi ricevuti richiamando il metodo Acknowledge della classe Message.

L'applicazione può confermare la ricezione di ciascun messaggio singolarmente oppure può ricevere un batch di messaggi e richiamare il metodo Acknowledge solo per l'ultimo messaggio ricevuto. Quando il metodo di riconoscimento viene richiamato, vengono riconosciuti tutti i messaggi ricevuti dall'ultima volta che il metodo è stato richiamato.

Insieme a una di queste modalità di riconoscimento, un'applicazione può arrestare e riavviare la consegna dei messaggi in una sessione richiamando il metodo Recover della classe Session. I messaggi ricevuti ma precedentemente non riconosciuti vengono riconsegnati. Tuttavia, potrebbero non essere consegnati nella stessa sequenza in cui sono stati precedentemente consegnati. Nel frattempo, potrebbero essere arrivati messaggi con priorità più elevata e alcuni dei messaggi originali potrebbero essere scaduti. Nel dominio point - to - point, alcuni dei messaggi originali potrebbero essere stati utilizzati da un'altra applicazione.

Un'applicazione può determinare se un messaggio viene riconsegnato esaminando il contenuto del campo di intestazione JMSRefornita del messaggio. L'applicazione esegue questa operazione richiamando il metodo getJMSRedelivered() della classe Message.

Creazione di destinazioni in una applicazione JMS

Invece di richiamare le destinazioni come oggetti gestiti da un namespace JNDI (Java Naming and Directory Interface), un'applicazione JMS può utilizzare una sessione per creare le destinazioni in modo dinamico al runtime. Un'applicazione può utilizzare un URI (uniform resource identifier) per identificare una coda IBM MQ o un argomento e, facoltativamente, per specificare una o più proprietà di un oggetto Coda o Argomento.

Utilizzo di una sessione per creare oggetti Coda

Per creare un oggetto Coda, un'applicazione può utilizzare il metodo createQueue() di un oggetto Session, come mostrato nel seguente esempio:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

Questo codice crea un oggetto Coda con i valori predefiniti per tutte le relative proprietà. L'oggetto rappresenta una coda IBM MQ denominata Q1 appartenente al gestore code locale. Questa coda può essere una coda locale, una coda alias o una definizione di coda remota.

Il metodo createQueue() accetta anche un URI della coda come parametro. Un URI della coda è una stringa che specifica il nome di una coda IBM MQ e, facoltativamente, il nome del gestore code che possiede la coda e una o più proprietà dell'oggetto Coda. La seguente istruzione contiene un esempio di URI della coda:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

L'oggetto Queue creato da questa istruzione rappresenta una coda IBM MQ denominata Q2 di proprietà di un gestore code denominato QM2e tutti i messaggi inviati a questa destinazione sono persistenti e hanno una priorità di 5. Il gestore code identificato in questo modo può essere il gestore code locale o un gestore code remoto. Se si tratta di un gestore code remoto, IBM MQ deve essere configurato in modo che, quando l'applicazione invia un messaggio a questa destinazione, WebSphere MQ possa instradare il messaggio dal gestore code locale al gestore code QM2. Per ulteriori informazioni sugli URI, consultare “URI (Uniform Resource Identifier)” a pagina 225.

Si noti che il parametro nel metodo createQueue() contiene informazioni specifiche del provider. Pertanto, l'utilizzo del metodo createQueue() per creare un oggetto Queue, invece di richiamare un oggetto Queue come oggetto gestito da uno spazio dei nomi JNDI, potrebbe rendere l'applicazione meno portabile.

Un'applicazione può creare un oggetto TemporaryQueue utilizzando il metodo createTemporaryQueue () di un oggetto Session, come mostrato nel seguente esempio:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

Anche se una sessione viene utilizzata per creare una coda temporanea, l'ambito di una coda temporanea è la connessione utilizzata per creare la sessione. Qualsiasi sessione della connessione può creare produttori di messaggi e consumer di messaggi per la coda temporanea. La coda temporanea rimane fino al termine della connessione o fino a quando l'applicazione non elimina esplicitamente la coda temporanea utilizzando il metodo TemporaryQueue.delete (), a seconda di quale sia la prima.

Quando un'applicazione crea una coda temporanea, IBM MQ classes for JMS crea una coda dinamica nel gestore code a cui è connessa l'applicazione. La proprietà TEMPMODEL della factory di connessione specifica il nome della coda modello utilizzata per creare la coda dinamica e la proprietà TEMPQPREFIX della factory di connessione specifica il prefisso utilizzato per formare il nome della coda dinamica.

Utilizzo di una sessione per creare oggetti argomento

Per creare un oggetto Argomento, un'applicazione può utilizzare il metodo createTopic() di un oggetto Session, come mostrato nel seguente esempio:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

Questo codice crea un oggetto argomento con i valori predefiniti per tutte le proprietà. L'oggetto rappresenta un argomento chiamato Sport / Calcio/Risultati.

Il metodo createTopic() accetta anche un URI argomento come parametro. Un URI argomento è una stringa che specifica il nome di un argomento e, facoltativamente, una o più proprietà dell'oggetto Argomento. Il codice riportato di seguito contiene un esempio di URI argomento:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

L'oggetto Argomento creato da questo codice rappresenta un argomento denominato Sport / Tennis/ Results e tutti i messaggi inviati a questa destinazione non sono persistenti e hanno una priorità 0. Per ulteriori informazioni sugli URI argomento, consultare “URI (Uniform Resource Identifier)” a pagina 225.

Si noti che il parametro nel metodo () createTopiccontiene informazioni specifiche del provider. Pertanto, l'utilizzo del metodo createTopic() per creare un oggetto argomento, invece di richiamare un oggetto argomento come oggetto gestito da uno spazio dei nomi JNDI, potrebbe rendere l'applicazione meno portabile.

Un'applicazione può creare un oggetto TemporaryTopic utilizzando il metodo createTemporaryTopic () di un oggetto Session, come mostrato nel seguente esempio:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

Sebbene una sessione venga utilizzata per creare un argomento temporaneo, l'ambito di un argomento temporaneo è la connessione utilizzata per creare la sessione. Qualsiasi sessione della connessione può creare produttori di messaggi e consumer di messaggi per l'argomento temporaneo. L'argomento temporaneo rimane attivo fino al termine della connessione o fino a quando l'applicazione non elimina esplicitamente l'argomento temporaneo utilizzando il metodo TemporaryTopic.delete (), a seconda di quale delle due modalità si verifica prima.

Quando un'applicazione crea un argomento temporaneo, IBM MQ classes for JMS crea un argomento con un nome che inizia con i caratteri TEMP/*tempTopicPrefix*, dove *tempTopicPrefix* è il valore della proprietà TEMPTOPICPREFIX della factory di connessione.

URI (Uniform Resource Identifier)

Un URI della coda è una stringa che specifica il nome di una coda IBM MQ e, facoltativamente, il nome del gestore code proprietario della coda e una o più proprietà dell'oggetto Coda creato dall'applicazione. Un URI argomento è una stringa che specifica il nome di un argomento e, facoltativamente, una o più proprietà dell'oggetto Argomento creato dall'applicazione.

Un URI coda ha il formato seguente:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Un URI argomento ha il seguente formato:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Le variabili in questi formati hanno i significati seguenti:

QmgrName
Il nome del gestore code proprietario della coda identificata dall'URI.

Il gestore code può essere il gestore code locale o un gestore code remoto. Se si tratta di un gestore code remoto, IBM MQ deve essere configurato in modo che, quando un'applicazione invia un messaggio alla coda, WebSphere MQ possa instradare il messaggio dal gestore code locale al gestore code remoto.

Se non viene specificato alcun nome, viene utilizzato il gestore code locale.

qName
Il nome della coda IBM MQ .

La coda può essere una coda locale, una coda alias o una definizione di coda remota.

Per le regole per la creazione di nomi coda, consultare Regole per la denominazione degli oggetti IBM MQ.

topicName
Il nome dell'argomento.

Per le regole per la creazione dei nomi argomento, consultare Regole per la denominazione degli oggetti IBM MQ. Evitare l'utilizzo dei caratteri jolly +, #, * e? nei nomi argomento. I nomi degli argomenti che contengono questi caratteri possono causare risultati imprevisti quando si sottoscrivono. Consultare Combinazione di stringhe di argomenti.

***propertyName1*, *propertyName2*, ...**
I nomi delle proprietà dell'oggetto Coda o Argomento creato dall'applicazione. Tabella 35 a pagina 226 elenca i nomi di proprietà validi che possono essere utilizzati in un URI.

Se non viene specificata alcuna proprietà, l'oggetto Coda o Argomento ha i valori predefiniti per tutte le relative proprietà.

***propertyValue1*, *propertyValue2*, ...**
I valori delle proprietà dell'oggetto Coda o Argomento creato dall'applicazione. Tabella 35 a pagina 226 elenca i valori di proprietà validi che possono essere utilizzati in un URI.

Le parentesi ([]) indicano un componente facoltativo e i puntini di sospensione (...) indicano che l'elenco delle coppie nome - valore della proprietà, se presenti, può contenere una o più coppie nome - valore.

Tabella 35 a pagina 226 elenca i nomi proprietà validi e i valori validi che possono essere utilizzati negli URI della coda e dell'argomento. Anche se lo strumento di gestione IBM MQ JMS utilizza costanti simboliche per i valori delle proprietà, gli URI non possono contenere costanti simboliche.

Tabella 35. Nomi di proprietà e valori validi da utilizzare negli URI di argomenti e code

Nome proprietà	**Descrizione**	**Valori validi**
CCSID	Modalità di rappresentazione dei dati carattere nel corpo di un messaggio quando IBM MQ classes for JMS inoltra il messaggio alla destinazione	• Qualsiasi CCSID (coded character set identifier) supportato da IBM MQ.
codifica	Come vengono rappresentati i dati numerici nel corpo di un messaggio quando IBM MQ classes for JMS inoltra il messaggio alla destinazione	• Qualsiasi valore valido per il campo *Codifica* in un descrittore di messaggi IBM MQ .
scadenza	Il TTL (time to live) per i messaggi inviati alla destinazione	• -2 - Come specificato nella chiamata send () o, se non specificato nella chiamata send (), la durata predefinita del produttore del messaggio. • 0 - Un messaggio inviato alla destinazione non scade mai. • Un numero intero positivo che indica la durata in millisecondi.

Tabella 35. Nomi di proprietà e valori validi da utilizzare negli URI di argomenti e code (Continua)		
Nome proprietà	**Descrizione**	**Valori validi**
supporto multicast	L'impostazione multicast per un argomento quando si utilizza una connessione in tempo reale a un broker	Il seguente elenco contiene valori validi. Associato a ciascun valore è il valore corrispondente della proprietà MULTICAST come utilizzato nello strumento di gestione IBM MQ JMS . Per una descrizione della proprietà MULTICAST e dei relativi valori validi, vedere Proprietà degli oggetti IBM MQ classes for JMS. • -1 - ASCF • 0 - DISABILITATO • 3 - NOTR • 5 - AFFIDABILE • 7 - ABILITATO
persistenza	La persistenza dei messaggi inviati alla destinazione	• -2 - Come specificato nella chiamata send () o, se non specificato nella chiamata send (), la persistenza predefinita del produttore del messaggio. • -1 - Come specificato dall'attributo *DefPersistence* della coda o dell'argomento IBM MQ . • 1 - Non persistente. • 2 - Permanente. • 3 - Equivalente al valore HIGH per la proprietà PERSISTENCE utilizzato nello strumento di somministrazione di IBM MQ JMS . Per una spiegazione di questo valore, consultare “JMS Messaggi persistenti” a pagina 256.
priorità	La priorità dei messaggi inviati alla destinazione	• -2 - Come specificato nella chiamata send () o, se non specificato nella chiamata send (), la priorità predefinita del produttore del messaggio. • -1 - Come specificato dall'attributo *DefPriority* della coda o dell'argomento IBM MQ . • Un numero intero compreso tra 0 e 9 che specifica la priorità dei messaggi inviati alla destinazione.
targetClient	Se i messaggi inviati alla destinazione contengono un'intestazione MQRFH2	• 0 - I messaggi contengono un'intestazione MQRFH2 . • 1 - I messaggi non contengono un'intestazione MQRFH2 .

Ad esempio, il seguente URI identifica una coda IBM MQ denominata Q1 appartenente al gestore code locale. Un oggetto coda creato utilizzando questo URI ha i valori predefiniti per tutte le proprie proprietà.

```
queue:///Q1
```

Il seguente URI identifica una coda IBM MQ denominata Q2 di proprietà di un gestore code denominato QM2. Tutti i messaggi inviati a questa destinazione hanno una priorità di 6. Le restanti proprietà dell'oggetto Coda creato utilizzando questo URI hanno i valori predefiniti.

```
queue://QM2/Q2?priority=6
```

Il seguente URI identifica un argomento denominato Sport / Atletica/Risultati. Tutti i messaggi inviati a questa destinazione sono non persistenti e hanno una priorità 0. Le restanti proprietà dell'oggetto Argomento creato utilizzando questo URI hanno i propri valori predefiniti.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

Invio di messaggi in un'applicazione JMS

Prima che un'applicazione JMS possa inviare messaggi a una destinazione, deve prima creare un oggetto MessageProducer per la destinazione. Per inviare un messaggio alla destinazione, l'applicazione crea un oggetto Messaggio e richiama il metodo send () dell'oggetto MessageProducer .

Un'applicazione utilizza un oggetto MessageProducer per inviare messaggi. Un'applicazione normalmente crea un oggetto MessageProducer per una destinazione specifica, che può essere una coda o un argomento, in modo che tutti i messaggi inviati utilizzando il produttore del messaggio vengano inviati alla stessa destinazione. Di conseguenza, prima che un'applicazione possa creare un oggetto MessageProducer , deve prima creare un oggetto Coda o Argomento. Per informazioni su come creare un oggetto Coda o Argomento, consultare i seguenti argomenti:

- “Utilizzo di JNDI per richiamare oggetti gestiti in un'applicazione JMS o Jakarta Messaging” a pagina 208
- “Utilizzo delle estensioni IBM JMS” a pagina 210
- “Utilizzo delle estensioni IBM MQ JMS” a pagina 217
- “Creazione di destinazioni in una applicazione JMS” a pagina 223

Per creare un oggetto MessageProducer , un'applicazione utilizza il metodo createProducer() di un oggetto Session, come mostrato nel seguente esempio:

```
MessageProducer producer = session.createProducer(destination);
```

Il parametro `destination` è un oggetto Coda o Argomento creato in precedenza dall'applicazione.

Prima che un'applicazione possa inviare un messaggio, deve creare un oggetto Messaggio. Il corpo di un messaggio contiene i dati dell'applicazione e JMS definisce cinque tipi di corpo del messaggio:

- Byte
- Associazione
- Oggetto
- Flusso
- Testo

Ogni tipo di contenuto del messaggio dispone della propria interfaccia JMS , che è un'interfaccia secondaria dell'interfaccia del messaggio e un metodo nell'interfaccia Session per la creazione di un messaggio con quel tipo di corpo. Ad esempio, l'interfaccia per un messaggio di testo è denominata

TextMessagee un'applicazione utilizza il metodo createTextMessage () di un oggetto Session per creare un messaggio di testo, come mostrato nella seguente istruzione:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Per ulteriori informazioni sui messaggi e sul corpo del messaggio, consultare “JMS messaggi” a pagina 146.

Per inviare un messaggio, un'applicazione utilizza il metodo send () di un oggetto MessageProducer , come mostrato nel seguente esempio:

```
producer.send(outMessage);
```

Un'applicazione può utilizzare il metodo send () per inviare messaggi in uno dei domini di messaggistica. La natura della destinazione determina quale dominio di messaggistica viene utilizzato. Tuttavia, TopicPublisher, l'interfaccia secondaria di MessageProducer specifica del dominio di pubblicazione / sottoscrizione, dispone anche di un metodo publish (), che può essere utilizzato al posto del metodo send (). I due metodi sono funzionalmente gli stessi.

Un'applicazione può creare un oggetto MessageProducer senza una destinazione specificata. In questo caso, l'applicazione deve specificare la destinazione quando si richiama il metodo send ().

Se un'applicazione invia un messaggio all'interno di una transazione, il messaggio non viene consegnato alla sua destinazione fino a quando non viene eseguito il commit della transazione. Ciò significa che un'applicazione non può inviare un messaggio e ricevere una risposta al messaggio all'interno della stessa transazione.

Una destinazione può essere configurata in modo che quando un'applicazione invia messaggi, IBM MQ classes for JMS inoltra il messaggio e restituisce il controllo all'applicazione senza determinare se il gestore code ha ricevuto il messaggio in modo sicuro. A volte ci si riferisce a questo come *inserimento asincrono*. Per ulteriori informazioni, consultare “Inserimento asincrono dei messaggi in IBM MQ classes for JMS” a pagina 322.

Ricezione di messaggi in un'applicazione JMS

Un'applicazione utilizza un consumatore di messaggi per ricevere messaggi. Un sottoscrittore di argomenti durevoli è un consumatore di messaggi che riceve tutti i messaggi inviati a una destinazione, inclusi quelli inviati mentre il consumatore è inattivo. Un'applicazione può selezionare i messaggi che desidera ricevere utilizzando un selettore di messaggi e può ricevere messaggi in modo asincrono utilizzando un listener di messaggi.

Un'applicazione utilizza un oggetto MessageConsumer per ricevere messaggi. Un'applicazione crea un oggetto MessageConsumer per una destinazione specifica, che può essere una coda o un argomento, in modo che tutti i messaggi ricevuti utilizzando il consumer dei messaggi vengano ricevuti dalla stessa destinazione. Di conseguenza, prima che un'applicazione possa creare un oggetto MessageConsumer , deve prima creare un oggetto Coda o Argomento. Per informazioni su come creare un oggetto Coda o Argomento, consultare i seguenti argomenti:

- “Utilizzo di JNDI per richiamare oggetti gestiti in un'applicazione JMS o Jakarta Messaging” a pagina 208
- “Utilizzo delle estensioni IBM JMS” a pagina 210
- “Utilizzo delle estensioni IBM MQ JMS” a pagina 217
- “Creazione di destinazioni in una applicazione JMS” a pagina 223

Per creare un oggetto MessageConsumer , un'applicazione utilizza il metodo createConsumer() di un oggetto Session, come mostrato nel seguente esempio:

```
MessageConsumer consumer = session.createConsumer(destination);
```

Il parametro `destination` è un oggetto Coda o Argomento creato in precedenza dall'applicazione.

L'applicazione utilizza quindi il metodo receive () dell'oggetto MessageConsumer per ricevere un messaggio dalla destinazione, come mostrato nel seguente esempio:

```
Message inMessage = consumer.receive(1000);
```

Il parametro sulla chiamata receive () specifica per quanto tempo, in millisecondi, il metodo attende l'arrivo di un messaggio appropriato se non è disponibile alcun messaggio immediatamente. Se si omette questo parametro, la chiamata si blocca indefinitamente fino all'arrivo di un messaggio appropriato. Se non si desidera che l'applicazione attenda un messaggio, utilizzare invece il metodo receiveNoWait ().

Il metodo receive () restituisce un messaggio di un tipo specifico. Ad esempio, quando un'applicazione riceve un messaggio di testo, l'oggetto restituito dalla chiamata receive () è un oggetto TextMessage .

Tuttavia, il tipo dichiarato di oggetto restituito da una chiamata receive () è un oggetto Messaggio. Pertanto, per estrarre i dati dal corpo di un messaggio appena ricevuto, l'applicazione deve eseguire il cast dalla classe Message alla sottoclasse più specifica, ad esempio TextMessage. Se il tipo di messaggio non è noto, l'applicazione può utilizzare l'operatore `instanceof` per determinare il tipo. È sempre buona norma per un'applicazione determinare il tipo di messaggio prima di eseguire il casting, in modo che gli errori possano essere gestiti correttamente.

Il seguente codice utilizza l'operatore `instanceof` e mostra come estrarre i dati dal corpo di un messaggio di testo:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Se un'applicazione invia un messaggio all'interno di una transazione, il messaggio non viene consegnato alla sua destinazione fino a quando non viene eseguito il commit della transazione. Ciò significa che un'applicazione non può inviare un messaggio e ricevere una risposta al messaggio all'interno della stessa transazione.

Se un utente di messaggi riceve messaggi da una destinazione configurata per la lettura anticipata, tutti i messaggi non persistenti che si trovano nel buffer di lettura anticipata al termine dell'applicazione vengono eliminati.

Nel dominio di pubblicazione / sottoscrizione, JMS identifica due tipi di utente di messaggi, sottoscrittore di argomenti non durevoli e sottoscrittore di argomenti durevoli, descritti nelle due seguenti sezioni.

Sottoscrittori di argomenti non durevoli

Un sottoscrittore di argomenti non durevoli riceve solo i messaggi pubblicati mentre il sottoscrittore è attivo. Una sottoscrizione non durevole inizia quando un'applicazione crea un sottoscrittore di argomenti non durevoli e termina quando l'applicazione chiude il sottoscrittore o quando il sottoscrittore non rientra nell'ambito. Come estensione in IBM MQ classes for JMS, un sottoscrittore di argomenti non durevoli riceve anche le pubblicazioni conservate.

Per creare un sottoscrittore di argomenti non durevoli, un'applicazione può utilizzare il metodo createConsumer() indipendente dal dominio, specificando un oggetto Argomento come destinazione. In alternativa, un'applicazione può utilizzare il metodo createSubscriber() specifico del dominio, come mostrato nel seguente esempio:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

Il parametro `topic` è un oggetto Argomento creato precedentemente dall'applicazione.

Sottoscrittori di argomenti durevoli

Limitazione: Un'applicazione non è in grado di creare sottoscrittori di argomenti durevoli quando si utilizza una connessione in tempo reale a un broker.

Un sottoscrittore di argomenti durevoli riceve tutti i messaggi pubblicati durante la durata di una sottoscrizione durevole. Questi messaggi includono tutti quelli pubblicati mentre il sottoscrittore non è attivo. Come estensione in IBM MQ classes for JMS, un sottoscrittore di argomenti durevoli riceve anche le pubblicazioni conservate.

Per creare un sottoscrittore di argomenti durevoli, un'applicazione utilizza il metodo createDurableSubscriber () di un oggetto Session, come mostrato nel seguente esempio:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

Nella chiamata createDurableSubscriber (), il primo parametro è un oggetto Topic che l'applicazione ha creato in precedenza e il secondo parametro è un nome utilizzato per identificare la sottoscrizione durevole.

La sessione utilizzata per creare un sottoscrittore di argomenti durevoli deve avere associato un identificativo client. L'identificativo client associato a una sessione è uguale all'identificativo client per il collegamento utilizzato per creare la sessione. L'ID client può essere specificato impostando la proprietà CLIENTID dell'oggetto ConnectionFactory . In alternativa, un'applicazione può specificare l'identificativo del client richiamando il metodo setClientID () dell'oggetto Connection.

Il nome utilizzato per identificare una sottoscrizione duratura deve essere univoco solo all'interno dell'identificativo client e, pertanto, l'identificativo client fa parte dell'identificativo univoco completo di una sottoscrizione durevole. Per continuare a utilizzare una sottoscrizione durevole creata precedentemente, un'applicazione deve creare un sottoscrittore di argomenti durevoli utilizzando una sessione con lo stesso identificativo client di quello associato alla sottoscrizione durevole e utilizzando lo stesso nome sottoscrizione.

Una sottoscrizione durevole viene avviata quando un'applicazione crea un sottoscrittore di argomenti durevoli utilizzando un identificativo client e un nome sottoscrizione per cui attualmente non esiste alcuna sottoscrizione durevole. Tuttavia, una sottoscrizione durevole non termina quando l'applicazione chiude il sottoscrittore dell'argomento durevole. Per terminare una sottoscrizione durevole, un'applicazione deve chiamare il metodo unsubscribe () di un oggetto Session che ha lo stesso identificativo client di quello associato alla sottoscrizione durevole. Il parametro sulla chiamata unsubscribe () è il nome della sottoscrizione, come mostrato nel seguente esempio:

```
session.unsubscribe("D_SUB_000001");
```

L'ambito di una sottoscrizione durevole è un gestore code. Se esiste una sottoscrizione durevole su un gestore code e un'applicazione connessa a un altro gestore code crea una sottoscrizione durevole con lo stesso identificativo client e lo stesso nome sottoscrizione, le due sottoscrizioni durevoli sono completamente indipendenti.

Selettori di messaggi

Un'applicazione può specificare che solo i messaggi che soddisfano determinati criteri vengono restituiti da chiamate receive () successive. Quando si crea un oggetto MessageConsumer , l'applicazione può specificare un'espressione SQL (Structured Query Language) che determina quali messaggi vengono richiamati. Questa espressione SQL è denominata *selettore messaggi*. Il selettore del messaggio può contenere i nomi dei campi di intestazione del messaggio JMS e le proprietà del messaggio. Per informazioni su come creare un selettore di messaggi, consultare “Selettori di messaggi in JMS” a pagina 147.

Il seguente esempio mostra come un'applicazione può selezionare i messaggi in base a una proprietà definita dall'utente denominata myProp:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

La specifica JMS non consente a un'applicazione di modificare il selettore di messaggi di un utente di messaggi. Dopo che un'applicazione crea un consumer di messaggi con un selettore di messaggi, il selettore di messaggi rimane per la vita di tale consumer. Se un'applicazione richiede più di un selettore di messaggi, l'applicazione deve creare un consumer di messaggi per ciascun selettore di messaggi.

Tenere presente che, quando un'applicazione è connessa a un gestore code versione 7, la proprietà MSGSELECTION della factory di connessione non ha alcun effetto. Per ottimizzare le prestazioni, tutta la selezione dei messaggi viene eseguita dal gestore code.

Soppressione delle pubblicazioni locali

Un'applicazione può creare un consumer di messaggi che ignora le pubblicazioni pubblicate sulla propria connessione del consumer. L'applicazione esegue questa operazione impostando il terzo parametro su una chiamata createConsumer() a `true`, come mostrato nel seguente esempio:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

Su una chiamata createDurableSubscriber (), l'applicazione esegue questa operazione impostando il quarto parametro su `true`, come mostrato nel seguente esempio

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

Consegna asincrona dei messaggi

Un'applicazione può ricevere messaggi in modo asincrono registrando un listener di messaggi con un utente di messaggi. Il listener dei messaggi ha un metodo denominato onMessage, che viene richiamato in maniera asincrona quando è disponibile un messaggio adatto e il cui scopo è elaborare il messaggio. Il seguente codice illustra il meccanismo:

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

Un'applicazione può utilizzare una sessione per ricevere i messaggi in modo sincrono utilizzando le chiamate receive () o per ricevere i messaggi in modo asincrono utilizzando i listener di messaggi, ma non per entrambi. Se un'applicazione deve ricevere messaggi in modo sincrono e asincrono, deve creare sessioni separate.

Una volta impostata una sessione per ricevere i messaggi in modo asincrono, non è possibile richiamare i seguenti metodi su quella sessione o sugli oggetti creati da quella sessione:

- MessageConsumer.receive ()
- MessageConsumer.receive (lungo)
- MessageConsumer.receiveNoWait ()
- Session.acknowledge()
- MessageProducer.send (Destinazione, Messaggio)
- MessageProducer.send (Destinazione, Messaggio, int, int, long)
- MessageProducer.send (Messaggio)
- MessageProducer.send (Messaggio, int, int, long)
- MessageProducer.send (Destinazione, Messaggio, CompletionListener)
- MessageProducer.send (Destinazione, Messaggio, int, int, long, CompletionListener)
- MessageProducer.send (Messaggio, CompletionListener)
- MessageProducer.send (Messaggio, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(coda)
- Session.createBrowser(Coda, Stringa)
- Session.createBytesMessage()
- Session.createConsumer(Destinazione)
- Session.createConsumer(Destinazione, stringa, booleano)
- Session.createDurableSubscriber(Argomento, Stringa)
- Session.createDurableSubscriber(Argomento, stringa, stringa, booleano)
- Session.createMapMessage()
- Session.createMessage()

- Session.createObjectMessage()
- Session.createObjectMessage(Serializzabile)
- Session.createProducer(Destinazione)
- Session.createQueue(Stringa)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(Stringa)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(Stringa)

Se viene richiamato uno di questi metodi, una JMSException contenente il messaggio:

```
JMSCC0033: Una chiamata di metodo sincrono non è consentita quando una sessione viene utilizzata
in modo asincrono: 'nome metodo'
```

viene generato.

Ricezione di messaggi non elaborabili

Un'applicazione può ricevere un messaggio che non è possibile elaborare. Ci possono essere diversi motivi per cui il messaggio non può essere elaborato, ad esempio il messaggio potrebbe avere un formato non corretto. Tali messaggi sono descritti come messaggi dannosi e richiedono una gestione speciale per evitare che il messaggio venga elaborato in modo ricorsivo.

Per i dettagli su come gestire i messaggi dannosi, consultare “Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236.

Adattamento delle dimensioni del buffer in base ai messaggi ricevuti

Quando un messaggio viene ricevuto da IBM MQ da un'applicazione nonJMS , l'applicazione deve fornire un buffer di messaggi in cui scrivere il messaggio. Le applicazioni JMS non devono creare manualmente un buffer. IBM MQ classes for JMS crea automaticamente e dimensiona i buffer di messaggi per adattarli alle dimensioni dei messaggi ricevuti. Per la maggior parte delle applicazioni, i buffer gestiti automaticamente forniscono un equilibrio adeguato di prestazioni e convenienza per lo sviluppatore dell'applicazione. In determinate circostanze, potrebbe essere utile specificare manualmente la dimensione iniziale del buffer di messaggi. La dimensione iniziale predefinita di un buffer di ricezione IBM MQ JMS è 4 KB. Se un'applicazione riceverà sempre messaggi con una dimensione di 256 KB, potrebbe essere preferibile configurare la dimensione del buffer iniziale su 256 KB. Ciò può evitare la necessità per IBM MQ classes for JMS di tentare e non ricevere il messaggio in un buffer di 4 KB prima di ridimensionarlo a 256 KB e riceverlo correttamente. Per un'applicazione connessa al client, ciò può evitare la necessità di un round trip di rete potenzialmente sprecato mentre IBM MQ classes for JMS determina la corretta dimensione del buffer da utilizzare.

La dimensione iniziale del buffer può essere configurata impostando la proprietà com.ibm.mq.jmqi.defaultMaxMsgSize Java sul valore scelto, in byte. Notare che questa proprietà influisce su tutte le applicazioni IBM MQ JMS in esecuzione all'interno di Java Virtual Machine, quindi fare attenzione a non influire negativamente su altri utenti di messaggi che ricevono messaggi di dimensioni diverse.

Il IBM MQ classes for JMS tenta ancora di ridurre automaticamente la dimensione del buffer se vengono ricevuti diversi messaggi più piccoli della dimensione configurata. Per impostazione predefinita, ciò si

verifica se vengono ricevuti 10 messaggi che sono tutti più piccoli della dimensione del buffer. Ad esempio, se 10 messaggi vengono ricevuti in una riga con una dimensione di 128 KB, il buffer viene ridotto da 256 KB a 128 KB. Viene quindi aumentato di nuovo quando vengono ricevuti messaggi più grandi. È possibile configurare il numero di messaggi che devono essere ricevuti prima che la dimensione di un buffer venga ridotta. Ad esempio, ciò potrebbe essere utile se si sa che l'applicazione riceve cinque messaggi di grandi dimensioni seguiti da 10 messaggi più piccoli e da altri cinque messaggi di grandi dimensioni. Con le impostazioni predefinite, il buffer viene ridotto dopo che sono stati ricevuti i 10 messaggi più piccoli e deve essere nuovamente aumentato per i messaggi più grandi. La Java proprietà di sistema com.ibm.mq.jmqi.smallMsgBufferReductionThreshold può essere impostata sul numero di messaggi che devono essere ricevuti prima della riduzione della dimensione del buffer. In questo esempio, potrebbe essere impostato su 20 per evitare che 10 messaggi più piccoli riducano la dimensione del buffer.

Le proprietà possono essere impostate indipendentemente l'una dall'altra. Ad esempio, è possibile scegliere di lasciare la dimensione del buffer iniziale al valore predefinito di 4 KB ma aumentare il valore di com.ibm.mq.jmqi.smallMsgBufferReductionThreshold in modo che, una volta incrementata la dimensione del buffer, rimanga tale più a lungo.

Se viene visualizzato un numero elevato di codici di ritorno MQRC_TRUNCATED_MSG_FAILED (2080) per le applicazioni JMS nei record delle statistiche MQI, ciò potrebbe indicare che si trarrebbe vantaggio dalla configurazione di una dimensione del buffer iniziale più elevata per tali applicazioni o dalla riduzione della frequenza con cui vengono ridotte le dimensioni del buffer. Tuttavia, è importante notare che per un'applicazione di lunga durata è probabile che venga visualizzato solo un numero molto piccolo di codici di ritorno MQRC_TRUNCATED_MSG_FAILED. Ciò è dovuto al fatto che, di solito, il buffer viene aumentato alla dimensione corretta immediatamente dopo la ricezione del primo messaggio di grandi dimensioni e non viene ridotto a meno che non venga ricevuto un numero di messaggi più piccoli. È pertanto possibile che un numero elevato di MQRC_TRUNCATED_MSG_FAILED indichi altre pratiche di applicazione non corrette, ad esempio la connessione a IBM MQ per ricevere solo uno o due messaggi prima della disconnessione.

Richiamo dei dati utente di sottoscrizione

Se i messaggi che un'applicazione IBM MQ classes for JMS sta utilizzando da una coda vengono inseriti da una sottoscrizione duratura definita amministrativamente, l'applicazione deve accedere alle informazioni sui dati utente associate alla sottoscrizione. Queste informazioni vengono aggiunte al messaggio come proprietà.

Quando un messaggio viene utilizzato da una coda contenente un'intestazione RFH2 con la cartella MQPS, il valore associato alla chiave Sud, se presente, viene aggiunto come proprietà String all'oggetto JMS Message restituito all'applicazione IBM MQ classes for JMS . Per abilitare il richiamo di questa proprietà dal messaggio, è possibile utilizzare la costante JMS_IBM_SUBSCRIPTION_USER_DATA nell'interfaccia JmsConstants con il seguente metodo per ottenere i dati utente della sottoscrizione:

- V 9.3.0 JM 3.0 V 9.3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

Nel seguente esempio, una sottoscrizione duratura di gestione viene definita utilizzando il comando MQSC **DEFINE SUB**:

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

Le copie dei messaggi pubblicati nella stringa di argomenti PUBLIC vengono inserite nella coda, MY.SUBSCRIPTION.Q. I dati utente associati alla sottoscrizione durevole vengono quindi aggiunti come una proprietà al messaggio, memorizzato nella cartella MQPS dell'intestazione RFH2 con la chiave Sud.

L'applicazione IBM MQ classes for JMS può richiamare:

V 9.3.0 JM 3.0 V 9.3.0

```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

```
JMS 2.0 javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

Viene quindi restituita la seguente stringa:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

Concetti correlati

"L'intestazione MQRFH2 e JMS" a pagina 151

Attività correlate

Definizione di una sottoscrizione di gestione

Riferimenti correlati

DEFINE SUB

Interfaccia JmsConstants

Chiusura di un'applicazione IBM MQ classes for JMS

È importante che un'applicazione IBM MQ classes for JMS chiuda esplicitamente alcuni oggetti JMS prima dell'arresto. I finalizer potrebbero non essere richiamati, quindi non fare affidamento su di essi per liberare risorse. Non consentire l'arresto di un'applicazione con la traccia compressa attiva.

La raccolta dati inutilizzati da sola non può rilasciare tutte le risorse IBM MQ classes for JMS e IBM MQ in modo tempestivo, specialmente se un'applicazione crea molti oggetti JMS di breve durata a livello di sessione o inferiore. È quindi importante che un'applicazione chiuda un oggetto Connection, Session, MessageConsumero MessageProducer quando non è più richiesto.

Se un'applicazione termina senza chiudere una connessione, si verifica un rollback implicito per tutte le sessioni transazionali della connessione. Per assicurarsi che tutte le modifiche apportate dall'applicazione siano sottoposte a commit, chiudere esplicitamente la connessione prima di chiudere l'applicazione.

Non utilizzare i finalizer in un'applicazione per chiudere gli oggetti JMS . Poiché i finalizer potrebbero non essere richiamati, le risorse potrebbero non essere liberate. Quando una connessione viene chiusa, vengono chiuse tutte le sessioni da essa create. Allo stesso modo, i MessageConsumers e MessageProducers creati da una sessione vengono chiusi quando la sessione viene chiusa. Tuttavia, considerare la chiusura esplicita di Sessioni, MessageConsumerse MessageProducers per garantire che le risorse vengano liberate in modo tempestivo.

Se la compressione della traccia è attivata, è probabile che System.Halt() arresti e terminazioni JVM anomale e non controllate provochino un file di traccia danneggiato. Se possibile, disattivare la funzionalità di traccia una volta raccolte le informazioni di traccia necessarie. Se si sta eseguendo la traccia di un'applicazione fino a una fine anomala, utilizzare l'output di traccia non compresso.

Nota: Per disconnettersi da un gestore code, un'applicazione JMS richiama il metodo close () sull'oggetto connessione.

Gestione dei messaggi non elaborabili in IBM MQ classes for JMS

Un messaggio non elaborabile è quello che non può essere elaborato da un'applicazione ricevente. Se un messaggio non elaborabile viene consegnato a un'applicazione e ne viene eseguito il rollback un numero specificato di volte, IBM MQ classes for JMS può spostarlo in una coda di backout.

Un messaggio non elaborabile è un messaggio che non può essere elaborato da un'applicazione ricevente. Il messaggio potrebbe avere un tipo non previsto o contenere informazioni che non possono essere gestite dalla logica dell'applicazione. Se un messaggio dannoso viene consegnato a un'applicazione, quest' ultima non sarà in grado di elaborarlo e ne eseguirà il rollback nella coda da cui proviene. Per impostazione predefinita, IBM MQ classes for JMS ridistribuirà ripetutamente il messaggio all'applicazione. Ciò può causare il blocco dell'applicazione in un loop che tenta continuamente di elaborare il messaggio dannoso e di eseguirlo di nuovo.

Per impedire che ciò accada, IBM MQ classes for JMS può rilevare messaggi non elaborabili e spostarli in una destinazione alternativa. A tale scopo, IBM MQ classes for JMS utilizza le proprietà riportate di seguito:

- Il valore del campo BackoutCount all'interno di MQMD del messaggio rilevato.
- Gli IBM MQ attributi della coda **BOTHRESH** (soglia di backout) e **BOQNAME** (coda di backout) per la coda di input contenente il messaggio.

Ogni volta che un messaggio viene sottoposto a rollback da un'applicazione, il gestore code incrementa automaticamente il valore del campo BackoutCount per il messaggio.

Quando IBM MQ classes for JMS rileva un messaggio con un valore BackoutCount maggiore di zero, confronta il valore di BackoutCount con il valore dell'attributo **BOTHRESH** .

- Se BackoutCount è inferiore al valore dell'attributo **BOTHRESH** , il IBM MQ classes for JMS lo consegna all'applicazione per l'elaborazione.
- Tuttavia, se BackoutCount è maggiore o uguale a **BOTHRESH**, il messaggio viene considerato come un messaggio non elaborabile. In questa situazione, IBM MQ classes for JMS sposta il messaggio nella coda specificata dall'attributo **BOQNAME** . Se il messaggio non può essere inserito nella coda di backout, viene spostato nella coda dei messaggi non recapitabili del gestore code oppure eliminato, in base alle opzioni di report del messaggio.

Nota:

- Se l'attributo **BOTHRESH** viene lasciato al valore predefinito `0`, la gestione dei messaggi non elaborabili è disabilitata. Ciò significa che tutti i messaggi non elaborabili vengono reinseriti nella coda di input.
- L'altra cosa da notare è che IBM MQ classes for JMS interrogare gli attributi **BOTHRESH** e **BOQNAME** per la coda la prima volta che rilevano un messaggio con un BackoutCount maggiore di zero. I valori di questi attributi vengono quindi memorizzati nella cache e utilizzati ogni volta che IBM MQ classes for JMS rileva un messaggio con un BackoutCount maggiore di zero.

Configurazione del sistema per eseguire la gestione dei messaggi non elaborabili

La coda utilizzata da IBM MQ classes for JMS durante l'interrogazione degli attributi **BOTHRESH** e **BOQNAME** dipende dallo stile di messaggistica eseguito:

- Per la messaggistica point - to - point, questa è la coda locale sottostante. Ciò è importante quando un'applicazione JMS utilizza i messaggi dalle code alias o dalle code cluster.
- Per la messaggistica di pubblicazione / sottoscrizione, viene creata una coda gestita per contenere i messaggi per una applicazione. IBM MQ classes for JMS esegue la query della coda gestita per determinare i valori degli attributi **BOTHRESH** e **BOQNAME** .

  La coda gestita viene creata da una coda modello associata all'oggetto Argomento a cui l'applicazione ha eseguito la sottoscrizione ed eredita i valori degli attributi **BOTHRESH** e **BOQNAME** dalla coda modello. La coda modello utilizzata dipende dal fatto che l'applicazione ricevente abbia eliminato una sottoscrizione durevole o non durevole:

  – La coda modello utilizzata per le sottoscrizioni durevoli è specificata dall'attributo **MDURMDL** dell'argomento. Il valore predefinito di questo attributo è `SYSTEM.DURABLE.MODEL.QUEUE`.
  – Per sottoscrizioni non durevoli, la coda modello utilizzata è specificata dall'attributo **MNDURMDL** . Il valore predefinito dell'attributo **MNDURMDL** è `SYSTEM.NDURABLE.MODEL.QUEUE`.

Quando si interrogano gli attributi **BOTHRESH** e **BOQNAME** , IBM MQ classes for JMS:

- Aprire la coda locale o la coda di destinazione per una coda alias.
- Analizzare gli attributi **BOTHRESH** e **BOQNAME** .
- Chiudere la coda locale o la coda di destinazione per una coda alias.

Le opzioni di apertura utilizzate quando si apre la coda locale o la coda di destinazione per una coda alias, dipendono dalla versione di IBM MQ classes for JMS utilizzata:

- Per IBM MQ classes for JMS in IBM MQ 9.1.0 Fix Pack 1 e versioni precedenti o IBM MQ 9.1.1, se la coda locale o la coda di destinazione per una coda alias, è una coda cluster, IBM MQ classes for JMS aprire la coda con le opzioni `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` e `MQOO_FAIL_IF_QUIESCING` .

Ciò significa che l'utente che esegue l'applicazione ricevente deve disporre dell'interrogazione e dell'accesso all'istanza locale della coda cluster.

Il IBM MQ classes for JMS apre tutti gli altri tipi di coda locale con le opzioni di apertura MQOO_INQUIRE e MQOO_FAIL_IF_QUIESCING. Per fare in modo che IBM MQ classes for JMS esegua la query dei valori degli attributi, l'utente che esegue l'applicazione ricevente deve avere accesso alla richiesta sulla coda locale.

- Quando si utilizza IBM MQ classes for JMS in IBM MQ 9.1.0 Fix Pack 2 e versioni successive o per IBM MQ 9.1.2 e versioni successive, l'utente che esegue l'applicazione ricevente deve disporre dell'accesso di interrogazione sulla coda locale, indipendentemente dal tipo di coda.

Per spostare i messaggi non elaborabili in una coda di riaccodamento di backout o in una coda di messaggi non recapitabili del gestore code, è necessario concedere all'utente che esegue l'applicazione le autorizzazioni put e passall .

Elaborazione di messaggi non elaborabili per applicazioni sincrone

Se un'applicazione riceve i messaggi in modo sincrono, richiamando uno dei seguenti metodi, IBM MQ classes for JMS riaccoda un messaggio non elaborabile all'interno dell'unità di lavoro che era attiva quando l'applicazione ha tentato di richiamare il messaggio:

- JMSConsumer.receive()
- JMSConsumer.receive(timeout lungo)
- JMSConsumer.receiveBody(Classe < T> c)
- JMSConsumer.receiveBody(Classe < T> c, timeout lungo)
- Classe JMSConsumer.receiveBodyNoWait < T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive ()
- MessageConsumer.receive (timeout lungo)
- MessageConsumer.receiveNoWait ()
- QueueReceiver.receive ()
- QueueReceiver.receive (timeout lungo)
- QueueReceiver.receiveNoWait ()
- TopicSubscriber.receive ()
- TopicSubscriber.receive (timeout lungo)
- TopicSubscriber.receiveNoWait ()

Ciò significa che se l'applicazione utilizza un contesto o una sessione JMS transazionali, lo spostamento del messaggio nella coda di backout non viene eseguito fino a quando non viene eseguito il commit della transazione.

Se l'attributo **BOTHRESH** è impostato su un valore diverso da zero, è necessario impostare anche l'attributo **BOQNAME** . Se **BOTHRESH** è impostato su un valore maggiore di zero e **BOQNAME** non è stato impostato, il funzionamento è determinato dalle opzioni del report del messaggio:

- Se il messaggio ha l'opzione di report MQRO_DISCARD_MSG impostata, il messaggio viene eliminato.
- Se il messaggio ha l'opzione di report MQRO_DEAD_LETTER_Q specificata, IBM MQ classes for JMS tenta di spostare il messaggio nella coda dei messaggi non recapitabili del gestore code.
- Se il messaggio non ha impostato MQRO_DISCARD_MSG o MQRO_DEAD_LETTER_Q , IBM MQ classes for JMS tenta di inserire il messaggio nella coda dei messaggi non recapitabili per il gestore code.

Nel caso in cui il tentativo di inserire il messaggio nella coda dei messaggi non recapitabili non riesca per qualche motivo, ciò che accade al messaggio viene determinato dal fatto che l'applicazione ricevente stia utilizzando un contesto o una sessione JMS transazionali o non transazionali:

- Se l'applicazione ricevente utilizza un contesto o una sessione JMS transazionali e viene eseguito il commit della transazione, il messaggio viene eliminato.
- Se l'applicazione ricevente sta utilizzando una sessione o un contesto JMS transazionali ed esegue il rollback della transazione, il messaggio viene restituito alla coda di input.
- Se l'applicazione ricevente ha creato una sessione o un contesto JMS non transazionali, il messaggio viene eliminato.

Elaborazione dei messaggi non elaborabili per le applicazioni asincrone

Se un'applicazione riceve i messaggi in modo asincrono tramite un MessageListener, IBM MQ classes for JMS riaccoda i messaggi non elaborabili senza influire sulla consegna dei messaggi. Il processo di riaccodamento si svolge al di fuori di qualsiasi unità di lavoro associata alla consegna effettiva del messaggio all'applicazione.

Se **BOTHRESH** è impostato su un valore maggiore di zero e **BOQNAME** non è stato impostato, il funzionamento è determinato dalle opzioni del report del messaggio:

- Se il messaggio ha l'opzione di report `MQRO_DISCARD_MSG` impostata, il messaggio viene eliminato.
- Se il messaggio ha l'opzione di report `MQRO_DEAD_LETTER_Q` specificata, IBM MQ classes for JMS tenta di spostare il messaggio nella coda dei messaggi non recapitabili del gestore code.
- Se il messaggio non ha impostato `MQRO_DISCARD_MSG` o `MQRO_DEAD_LETTER_Q` , IBM MQ classes for JMS tenta di inserire il messaggio nella coda dei messaggi non recapitabili per il gestore code.

Se il tentativo di inserire il messaggio nella DLQ (dead letter queue) ha esito negativo per qualche motivo, IBM MQ classes for JMS restituisce il messaggio alla coda di input.

Per informazioni su come le specifiche di attivazione e ConnectionConsumers gestiscono i messaggi non elaborabili, consultare Rimozione dei messaggi dalla coda in ASF.

Cosa succede a un messaggio quando viene spostato nella coda di backout

Quando un messaggio non elaborabile viene riaccodato alla coda di riaccodamento di backout, IBM MQ classes for JMS aggiungere un'intestazione RFH2 (se non ne aveva già una) e aggiornare alcuni campi all'interno del descrittore del messaggio (MQMD).

Se il messaggio non elaborabile contiene un'intestazione RFH2 (perché era un messaggio JMS , ad esempio), IBM MQ classes for JMS modifica i seguenti campi all'interno di MQMD quando sposta il messaggio nella coda di riaccodamento di backout:

- Il campo BackoutCount viene reimpostato su zero.
- Il campo Scadenza del messaggio viene aggiornato per riflettere la scadenza rimanente nel momento in cui il messaggio non elaborabile è stato ricevuto dall'applicazione JMS .

Se il messaggio non elaborabile non contiene un'intestazione RFH2 , IBM MQ classes for JMS aggiungerne uno e aggiornare i seguenti campi in MQMD come parte dell'elaborazione di backout:

- Il campo BackoutCount viene reimpostato su zero.
- Il campo Scadenza del messaggio viene aggiornato per riflettere la scadenza rimanente nel momento in cui il messaggio non elaborabile è stato ricevuto dall'applicazione JMS .
- Il campo Formato del messaggio viene modificato in MQHRF2.
- Il campo CCSID viene modificato in 1208.
- Il campo Codifica viene modificato in modo da essere 273.

Inoltre, i campi CCSID e Codifica del messaggio non elaborabile vengono copiati nei campi CCSID e Codifica dell'intestazione RFH2 , per assicurare che il concatenamento dell'intestazione del messaggio sulla coda di backout sia corretto.

Concetti correlati

“Gestione dei messaggi non elaborabili in ASF” a pagina 340

All'interno delle funzioni del server delle applicazioni, la gestione dei messaggi non elaborabili viene gestita in modo leggermente diverso rispetto a quanto avviene altrove in IBM MQ classes for JMS.

Eccezioni in IBM MQ classes for JMS

Un'applicazione IBM MQ classes for JMS deve gestire le eccezioni generate dalle chiamate API JMS o che vengono consegnate a un gestore eccezioni.

IBM MQ classes for JMS riporta problemi di runtime generando eccezioni. Il tipo di eccezioni che vengono generate e il modo in cui queste eccezioni devono essere gestite dipendono dalla versione della specifica JMS utilizzata dall'applicazione:

- I metodi sulle interfacce definite in JMS 1.1 e precedenti generano eccezioni verificate. La classe di base di queste eccezioni è `JMSException`. Per ulteriori informazioni su come gestire le eccezioni controllate, vedere "Gestione delle eccezioni controllate" a pagina 240.
- I metodi sulle interfacce aggiunte in JMS 2.0 generano eccezioni non controllate. La classe di base per queste eccezioni è `JMSRuntimeException`. Per ulteriori informazioni su come gestire le eccezioni non controllate, vedere "Gestione delle eccezioni non controllate" a pagina 243.

È anche possibile registrare un `ExceptionListener` con un JMS `Connection` o un `JMSContext`. Le classi MQ per JMS inviano una notifica a `ExceptionListener` se viene rilevato un problema con una connessione al gestore code o se si verifica un problema durante il tentativo di consegnare un messaggio in modo asincrono. Per ulteriori informazioni, consultare "ExceptionListeners" a pagina 246.

Concetti correlati

Classi IBM MQ per JMS

Riferimenti correlati

ECCEZIONE asincrona

Gestione delle eccezioni controllate

I metodi sulle interfacce definite in JMS 1.1 o precedenti generano eccezioni verificate. La classe di base per queste eccezioni è `JMSException`. Pertanto, la cattura di `JMSExceptions` fornisce un modo generico di gestire questi tipi di eccezione.

Ogni `JMSException` contiene le seguenti informazioni:

- Un messaggio di eccezione specifico del provider, che l'applicazione può ottenere richiamando il metodo `Throwable.getMessage()`.
- Un codice di errore specifico del fornitore, che può essere ottenuto dall'applicazione richiamando il metodo `JMSException.getErrorCode()`.
- Un'eccezione collegata. Un'eccezione generata da una chiamata API JMS 1.1 è spesso il risultato di un problema di livello inferiore riportato da un'altra eccezione collegata a questa eccezione. L'applicazione può ottenere un'eccezione collegata richiamando il metodo `JMSException.getLinkedException()` o il metodo `Throwable.getCause()`.

Quando si utilizza l'API JMS 1.1 , la maggior parte delle eccezioni generate da IBM MQ classes for JMS sono istanze di sottoclassi di `JMSException`. Queste sottoclassi implementano l'interfaccia `com.ibm.msg.client.jms.JmsExceptionDetail` , che fornisce le seguenti informazioni aggiuntive:

- Una spiegazione del messaggio di eccezione. L'applicazione può ottenere questo messaggio richiamando il metodo `JmsExceptionDetail.getExplanation()`.
- Una risposta utente consigliata all'eccezione. L'applicazione può ottenere questo messaggio richiamando il metodo `JmsExceptionDetail.getUserAction()`.
- Le chiavi per il messaggio vengono inserite nel messaggio di eccezione. L'applicazione può ottenere un iteratore per tutte le chiavi richiamando il metodo `JmsExceptionDetail.getKeys()`.
- Il messaggio viene inserito nel messaggio di eccezione. Ad esempio, un inserimento di un messaggio potrebbe essere il nome della coda che ha causato l'eccezione e potrebbe essere utile per l'applicazione per accedere a tale nome. L'applicazione può ottenere l'inserimento del messaggio corrispondente a una chiave specificata richiamando il metodo `JmsExceptionDetail.getValue()`.

Tutti i metodi nell'interfaccia `JmsExceptionDetail` restituiscono un valore null se non sono disponibili dettagli.

Ad esempio, se un'applicazione tenta di creare un produttore di messaggi per una coda IBM MQ che non esiste, viene generata un'eccezione con le seguenti informazioni:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

L'eccezione generata, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, è una sottoclasse della seguente classe e implementa l'interfaccia di `com.ibm.msg.client.jms.JmsExceptionDetail` .

- V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Eccezioni collegate

Un'eccezione collegata fornisce ulteriori informazioni su un problema di runtime. Pertanto, per ogni `JMSException` generato, un'applicazione deve controllare l'eccezione collegata.

L'eccezione collegata stessa potrebbe avere un'altra eccezione collegata, e quindi le eccezioni collegate formano una catena che riporta al problema sottostante originale. Un'eccezione collegata viene implementata utilizzando il meccanismo di eccezione concatenato della classe `java.lang.Throwable` e l'applicazione può ottenere un'eccezione collegata richiamando il metodo `Throwable.getCause()` . Per un `JMSException`, il metodo `getLinkedException()` delega al metodo `Throwable.getCause()` .

Ad esempio, se un'applicazione specifica un numero di porta non corretto durante la connessione a un gestore code, le eccezioni formano la seguente catena:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Di solito, ogni eccezione in una catena viene generata da un livello differente nel codice. Ad esempio, le eccezioni nella catena precedente vengono generate dai seguenti livelli:

- La prima eccezione, un'istanza di una classe secondaria di `JMSException`, viene generata dal livello comune in IBM MQ classes for JMS.
- L'eccezione successiva, un'istanza di `com.ibm.mq.MQException`, viene generata dal fornitore di messaggistica IBM MQ .
- Le due eccezioni successive, entrambe istanze di `com.ibm.mq.jmqi.JmqiException`, vengono generate da JMQI (Java Message Queueing Interface). JMQI è il componente utilizzato da IBM MQ classes for JMS per comunicare con un gestore code.
- L'eccezione finale, un'istanza di `java.net.ConnectionException`, viene generata dalla libreria di classi Java .

Per ulteriori informazioni sull'architettura a livelli di IBM MQ classes for JMS, consultare IBM MQ classes per l'architettura JMS.

È possibile codificare l'applicazione da iterare attraverso questa catena per estrarre tutte le informazioni appropriate, come mostrato nel seguente esempio:

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
```

```
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Si noti che l'applicazione deve sempre controllare il tipo di ogni eccezione in una catena perché il tipo di eccezione può variare e le eccezioni di tipi diversi incapsulano informazioni diverse.

Come ottenere informazioni specifiche su un problema IBM MQ

Istanze di `com.ibm.mq.MQException` e `com.ibm.mq.jmqi.JmqiException` incapsulare IBM MQ informazioni specifiche su un problema.

Un `MQException` racchiude le seguenti informazioni:

- Un codice di completamento, che la tua applicazione può ottenere richiamando il metodo `getCompCode()` .
- Un codice di errore, che può essere ottenuto dall'applicazione richiamando il metodo `getReason()` .

Per esempi su come utilizzare questi metodi, consultare il codice di esempio in eccezioni collegate.

Un `JmqiException` incapsula anche un codice di completamento e un codice motivo. Inoltre, un `JmqiException` contiene le informazioni in un messaggio AMQ *nnnn* o CSQ *nnnn* , se associato all'eccezione. L'applicazione può ottenere i vari componenti di questo messaggio richiamando i seguenti metodi:

- Il metodo `getWmqMsgExplanation()` restituisce la spiegazione del messaggio AMQ *nnnn* o CSQ *nnnn* .
- Il metodo `getWmqMsgSeverity()` restituisce la severità del messaggio AMQ *nnnn* o CSQ *nnnn* .
- Il metodo `getWmqMsgSummary()` restituisce il riepilogo del messaggio AMQ *nnnn* o CSQ *nnnn* .
- Il metodo `getWmqMsgUserResponse()` restituisce la risposta utente associata al messaggio AMQ *nnnn* o CSQ *nnnn* .

Gestione delle eccezioni non controllate
I metodi sulle interfacce definite in JMS 2.0 generano eccezioni non verificate. La classe di base per queste eccezioni è `JMSRuntimeException`. Pertanto, la cattura di `JMSRuntimeExceptions` fornisce un modo generico di gestire questi tipi di eccezione.

Ogni `JMSRuntimeException` contiene le seguenti informazioni:

- Un messaggio di eccezione specifico del provider, che l'applicazione può ottenere richiamando il metodo `JMSRuntimeException.getMessage()` .
- Un codice di errore specifico del fornitore, che può essere ottenuto dall'applicazione richiamando il metodo `JMSRuntimeException.getErrorCode()` .
- Un'eccezione collegata. Un'eccezione generata da una chiamata API JMS 2.0 è spesso il risultato di un problema di livello inferiore riportato da un'altra eccezione collegata a questa eccezione. L'applicazione può ottenere un'eccezione collegata richiamando il metodo `JMSRuntimeException.getCause()` .

Quando richiami i metodi sulle interfacce fornite dall'API JMS 2.0 , la maggior parte delle eccezioni generate da IBM MQ classes for JMS sono istanze di sottoclassi di `JMSRuntimeException`.
Queste sottoclassi implementano l'interfaccia `com.ibm.msg.client.jms.JmsExceptionDetail` , che fornisce le seguenti informazioni aggiuntive:

- Una spiegazione del messaggio di eccezione. L'applicazione può ottenere questo messaggio richiamando il metodo `JmsExceptionDetail.getExplanation()`.
- Una risposta utente consigliata all'eccezione. L'applicazione può ottenere questo messaggio richiamando il metodo `JmsExceptionDetail.getUserAction()`.
- Le chiavi per il messaggio vengono inserite nel messaggio di eccezione. L'applicazione può ottenere un iteratore per tutte le chiavi richiamando il metodo `JmsExceptionDetail.getKeys()`.
- Il messaggio viene inserito nel messaggio di eccezione. Ad esempio, un inserimento di un messaggio potrebbe essere il nome della coda che ha causato l'eccezione e potrebbe essere utile per l'applicazione per accedere a tale nome. L'applicazione può ottenere l'inserimento del messaggio corrispondente a una chiave specificata richiamando il metodo `JmsExceptionDetail.getValue()`.

Tutti i metodi nell'interfaccia `JmsExceptionDetail` restituiscono un valore null se non sono disponibili dettagli.

Ad esempio, se un'applicazione tenta di creare un `JMSProducer` per una coda IBM MQ che non esiste, viene generata un'eccezione con le seguenti informazioni:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

L'eccezione generata, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, è una sottoclasse della seguente classe e implementa l'interfaccia di `com.ibm.msg.client.jms.JmsExceptionDetail`.

- V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Eccezioni concatenate

Generalmente, le eccezioni sono causate da altre eccezioni. Pertanto, per ogni `JMSRuntimeException` generato, l'applicazione deve controllare l'eccezione collegata.

La causa di `JMSRuntimeException` potrebbe essere un'altra eccezione. Queste eccezioni formano una catena che riporta al problema di base originale. La causa di un'eccezione viene implementata utilizzando il meccanismo di eccezione concatenato della classe `java.lang.Throwable` e l'applicazione può ottenere un'eccezione collegata richiamando il metodo `Throwable.getCause()`.

Ad esempio, se un'applicazione specifica un numero di porta non corretto durante la connessione a un gestore code, le eccezioni formano la seguente catena:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Di solito, ogni eccezione in una catena viene generata da un livello differente nel codice. Ad esempio, le eccezioni nella catena precedente vengono generate dai seguenti livelli:

- La prima eccezione, un'istanza di una classe secondaria di `JMSRuntimeException`, viene generata dal livello comune in IBM MQ classes for JMS.
- L'eccezione successiva, un'istanza di `com.ibm.mq.MQException`, viene generata dal fornitore di messaggistica IBM MQ .
- Le due eccezioni successive, entrambe istanze di `com.ibm.mq.jmqi.JmqiException`, vengono generate da JMQI (Java Message Queueing Interface). JMQI è il componente utilizzato da IBM MQ classes for JMS per comunicare con un gestore code.
- L'eccezione finale, un'istanza di `java.net.ConnectionException`, viene generata dalla libreria di classi Java .

Per ulteriori informazioni sull'architettura a livelli di IBM MQ classes for JMS, consultare IBM MQ classes per l'architettura JMS.

È possibile codificare l'applicazione da iterare attraverso questa catena per estrarre tutte le informazioni appropriate, come mostrato nel seguente esempio:

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
```

```
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Si noti che l'applicazione deve sempre controllare il tipo di ogni eccezione in una catena perché il tipo di eccezione può variare e le eccezioni di tipi diversi incapsulano informazioni diverse.

Come ottenere informazioni specifiche su un problema IBM MQ

Istanze di `com.ibm.mq.MQException` e `com.ibm.mq.jmqi.JmqiException` incapsulare IBM MQ informazioni specifiche su un problema.

Un `MQException` racchiude le seguenti informazioni:

- Un codice di completamento, che la tua applicazione può ottenere richiamando il metodo `getCompCode()` .
- Un codice di errore, che può essere ottenuto dall'applicazione richiamando il metodo `getReason()` .

Per esempi su come utilizzare questi metodi, consultare il codice di esempio in Eccezioni concatenate.

Un `JmqiException` incapsula anche un codice di completamento e un codice motivo. Inoltre, un `JmqiException` contiene le informazioni in un messaggio AMQ *nnnn* o CSQ *nnnn* , se associato all'eccezione. L'applicazione può ottenere i vari componenti di questo messaggio richiamando i seguenti metodi:

- Il metodo `getWmqMsgExplanation()` restituisce la spiegazione del messaggio AMQ *nnnn* o CSQ *nnnn* .
- Il metodo `getWmqMsgSeverity()` restituisce la severità del messaggio AMQ *nnnn* o CSQ *nnnn* .
- Il metodo `getWmqMsgSummary()` restituisce il riepilogo del messaggio AMQ *nnnn* o CSQ *nnnn* .
- Il metodo `getWmqMsgUserResponse()` restituisce la risposta utente associata al messaggio AMQ *nnnn* o CSQ *nnnn* .

ExceptionListeners
Gli oggetti JMS `Connection` e `JMSContext` hanno una connessione associata a un gestore code. L'applicazione può registrare un `ExceptionListener` con un JMS `Connection` o `JMSContext`. Se si verifica un problema che rende inutilizzabile la connessione associata a `Connection` o `JMSContext` , IBM MQ classes for JMS fornisce un'eccezione a `ExceptionListener` richiamandone il metodo `onException()` . L'applicazione ha quindi la possibilità di ristabilire la connessione.

IBM MQ classes for JMS può anche consegnare un'eccezione al listener delle eccezioni se si verifica un problema durante il tentativo di consegnare un messaggio in modo asincrono.

Listener di eccezione

Da IBM MQ 8.0.0 Fix Pack 2, per mantenere il funzionamento delle applicazioni JMS correnti che configurano un JMS `MessageListener` e un JMS `ExceptionListener`, e per garantire che IBM MQ classes for JMS sia congruente con la specifica JMS , il valore predefinito per la proprietà `ConnectionFactory` ASYNCEXCEPTION viene modificato in ASYNC_EXCEPTIONS_CONNECTIONBROKEN. Di conseguenza, solo le eccezioni che corrispondono ai codici di errore di connessione interrotti vengono consegnate al `ExceptionListener` di un'applicazione.

APAR IT14820, incluso da IBM MQ 9.0.0 Fix Pack 1, aggiorna IBM MQ classes for JMS in modo che:

- Un `ExceptionListener` registrato da un'applicazione viene richiamato per qualsiasi eccezione di connessione interrotta, indipendentemente dal fatto che l'applicazione stia utilizzando utenti di messaggi sincroni o asincroni.
- Le eccezioni di non interruzione della connessione (ad esempio MQRC_GET_INHIBITED) che si verificano durante la consegna dei messaggi vengono consegnati al `ExceptionListener` di un'applicazione quando l'applicazione utilizza utenti di messaggi asincroni e il JMS ConnectionFactory utilizzato dall'applicazione ha la proprietà ASYNC_EXCEPTION impostata sul valore ASYNC_EXCEPTIONS_ALL.

Nota: Un `ExceptionListener` viene richiamato solo una volta per un'eccezione di connessione interrotta, anche se due connessioni TCP/IP (una utilizzata da una connessione JMS e una utilizzata da una sessione JMS) sono interrotte.

Per qualsiasi altro tipo di problema, viene generata un'eccezione dalla chiamata API JMS corrente. Il tipo di eccezione generata dipende dalla versione dell'API JMS utilizzata dall'applicazione:

- Se l'applicazione sta utilizzando le interfacce fornite dalla specifica JMS 1.1 , l'eccezione è `JMSException`. Per ulteriori informazioni su come gestire queste eccezioni, consultare “Gestione delle eccezioni controllate” a pagina 240.
- Se l'applicazione utilizza le interfacce JMS 2.0 , l'eccezione è `JMSRuntimeException`. Per ulteriori informazioni su come gestire queste eccezioni, consultare “Gestione delle eccezioni non controllate” a pagina 243.

Se un'applicazione non registra un listener di eccezioni con un `Connection` o `JMSContext`, tutte le eccezioni che verrebbero consegnate al listener di eccezioni vengono scritte nel log IBM MQ classes for JMS .

Accesso alle funzioni IBM MQ da una applicazione IBM MQ classes for JMS

IBM MQ classes for JMS fornisce funzioni per sfruttare una serie di funzioni di IBM MQ.

Attenzione: Queste funzioni si trovano all'esterno della specifica JMS o, in alcuni casi, violano la specifica JMS . Se si utilizzano, è improbabile che l'applicazione sia compatibile con altri provider JMS . Le funzioni non conformi alla specifica JMS sono etichettate con un avviso di attenzione.

Lettura e scrittura del descrittore del messaggio da una applicazione IBM MQ classes for JMS

Si controlla la possibilità di accedere al descrittore del messaggio (MQMD) impostando le proprietà su una destinazione e un messaggio.

Alcune applicazioni IBM MQ richiedono l'impostazione di valori specifici nell'MQMD dei messaggi inviati. IBM MQ classes for JMS fornisce attributi di messaggi che consentono alle applicazioni JMS di impostare i campi MQMD e quindi abilitare le applicazioni JMS a "guidare" le applicazioni IBM MQ .

È necessario impostare la proprietà dell'oggetto di destinazione WMQ_MQMD_WRITE_ENABLED su true affinché l'impostazione delle proprietà MQMD abbia effetto. È quindi possibile utilizzare i metodi di impostazione della proprietà del messaggio (ad esempio setStringProperty) per assegnare valori ai campi MQMD. Tutti i campi MQMD sono esposti tranne StrucId e Version; BackoutCount può essere letto ma non scritto.

Questo esempio determina l'inserimento di un messaggio in una coda o in un argomento con MQMD.UserIdentifier impostato suJoeBloggs".

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

È necessario impostare WMQ_MQMD_MESSAGE_CONTEXT prima di impostare JMS_IBM_MQMD_UserIdentifier. Per ulteriori informazioni sull'utilizzo di WMQ_MQMD_MESSAGE_CONTEXT, consultare “Proprietà dell'oggetto messaggio JMS” a pagina 250.

Allo stesso modo, è possibile estrarre il contenuto dei campi MQMD impostando WMQ_MQMD_READ_ENABLED su true prima di ricevere un messaggio e quindi utilizzando i metodi get del messaggio, come la proprietà getString. Tutte le proprietà ricevute sono di sola lettura.

In questo esempio viene visualizzato il campo *valore* contenente il valore di MQMD.ApplIdentityData di un messaggio ricevuto da una coda o da un argomento.

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

Proprietà dell'oggetto destinazione JMS
Due proprietà dell'oggetto Destination controllano l'accesso a MQMD da JMSe un terzo controlla il contesto del messaggio.

Tabella 36. Nomi e descrizioni delle proprietà

Proprietà	**Forma breve**	**Descrizione**
WMQ_MQMD_WRITE_ENABLED	MDW	Se un'applicazione JMS può impostare i valori dei campi MQMD
WMQ_MQMD_READ_ENABLED	MDR	Indica se un'applicazione JMS può estrarre i valori dei campi MQMD
WMQ_MQMD_MESSAGE_ CONTESTO	MDCTX	Quale livello di contesto del messaggio deve essere impostato dall'applicazione JMS . L'applicazione deve essere in esecuzione con l'autorizzazione di contesto appropriata per rendere effettiva questa proprietà

Tabella 37. Nomi proprietà, valori e metodi set

Proprietà	Valori validi nello strumento di gestione (valori predefiniti in grassetto)	Valori validi nei programmi	Imposta metodo
WMQ_MQMD_WRITE_ENABLED	• **No** Tutte le proprietà JMS_IBM_MQMD* vengono ignorate e i relativi valori non vengono copiati nella struttura MQMD sottostante. • SÌ Le proprietà JMS_IBM_MQMD* vengono elaborate. I loro valori vengono copiati nella struttura MQMD sottostante.	• **No** • Vero	setMQMDWriteabilitato
WMQ_MQMD_READ_ABILITATO	• **No** Quando si inviano messaggi, le proprietà JMS_IBM_MQMD* di un messaggio inviato non vengono aggiornate per riflettere i valori dei campi aggiornati in MQMD. Alla ricezione dei messaggi, nessuna delle proprietà JMS_IBM_MQMD* è disponibile sul messaggio ricevuto, anche se il mittente ne ha impostate alcune o tutte. • SÌ Quando si inviano i messaggi, tutte le proprietà JMS_IBM_MQMD* su un messaggio inviato vengono aggiornate per riflettere i valori del campo aggiornati in MQMD, inclusi quelli che il mittente non ha impostato esplicitamente. Alla ricezione dei messaggi, tutte le proprietà JMS_IBM_MQMD* sono disponibili sul messaggio ricevuto, comprese quelle non impostate esplicitamente dal mittente.	• **No** • Vero	setMQMDReadAbilitata

Tabella 37. Nomi proprietà, valori e metodi set (Continua)			
Proprietà	**Valori validi nello strumento di gestione (valori predefiniti in grassetto)**	**Valori validi nei programmi**	**Imposta metodo**
WMQ_MQMD _CONTESTO_MESSAGGIO	• **Valore predefinito** La chiamata API MQOPEN e la struttura MQPMO non specificano opzioni di contesto del messaggio esplicite • SET_IDENTITY_CONTEXT La chiamata API MQOPEN specifica l'opzione di contesto del messaggio MQOO_SET_IDENTITY_CONTEXT e la struttura di MQPMO specifica MQPMO_SET_IDENTITY_CONTEXT • IMPOSTA_TUTTI_CONTESTO La chiamata API MQOPEN specifica l'opzione di contesto del messaggi MQOO_SET_ALL_CONTEXT e la struttura MQPM specifica MQPMO_SET_ALL_CONTEXT	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	Contesto setMQMDMessage

Proprietà dell'oggetto messaggio JMS
Le proprietà dell'oggetto del messaggio con prefisso JMS_IBM_MQMD consentono di impostare o leggere il campo MQMD corrispondente.

invio di messaggi

Tutti i campi MQMD tranne StrucId e Version sono rappresentati. Queste proprietà fanno riferimento solo ai campi MQMD; dove una proprietà si verifica sia nell'intestazione MQMD che MQRFH2 , la versione in MQRFH2 non è impostata o estratta.

È possibile impostare una qualsiasi di queste proprietà, ad eccezione di JMS_IBM_MQMD_BackoutCount. Qualsiasi valore impostato per JMS_IBM_MQMD_BackoutCount viene ignorato.

Se una proprietà ha una lunghezza massima e si fornisce un valore troppo lungo, il valore viene troncato.

Per alcune proprietà, è necessario impostare anche la proprietà WMQ_MQMD_MESSAGE_CONTEXT sull'oggetto Destinazione. L'applicazione deve essere in esecuzione con l'autorizzazione di contesto appropriata perché questa proprietà diventi effettiva. Se non si imposta WMQ_MQMD_MESSAGE_CONTEXT su un valore appropriato, il valore della proprietà viene ignorato. Se si imposta WMQ_MQMD_MESSAGE_CONTEXT su un valore appropriato ma non si dispone di un'autorizzazione di contesto sufficiente per il gestore code, viene emessa una JMSException. Le proprietà che richiedono valori specifici di WMQ_MQMD_MESSAGE_CONTEXT sono le seguenti.

Le seguenti proprietà richiedono l'impostazione di WMQ_MQMD_MESSAGE_CONTEXT su WMQ_MDCTX_SET_IDENTITY_CONTEXT o WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- Dati JMS_IBM_MQMD_ApplIdentity

Le proprietà riportate di seguito richiedono che WMQ_MQMD_MESSAGE_CONTEXT sia impostato su WMQ_MDCTX_SET_ALL_CONTEXT:

- Tipo JMS_IBM_MQMD_PutAppl

- Nome JMS_IBM_MQMD_PutAppl
- JMS_IBM_MQMD_PutDate
- PutTime JMS_IBM_MQMD_
- JMS_IBM_MQMD_ApplOriginDati

ricezione di messaggi

Tutte queste proprietà sono disponibili su un messaggio ricevuto se la proprietà WMQ_MQMD_READ_ENABLED è impostata su true, indipendentemente dalle proprietà effettive che l'applicazione di produzione ha impostato. Un'applicazione non può modificare le proprietà di un messaggio ricevuto a meno che tutte le proprietà non vengano prima cancellate, secondo la specifica JMS . Il messaggio ricevuto può essere inoltrato senza modificare le proprietà.

Attenzione: Se la propria applicazione riceve un messaggio da una destinazione con la proprietà WMQ_MQMD_READ_ENABLED impostata su true e lo inoltra a una destinazione con WMQ_MQMD_WRITE_ENABLED impostata su true, tutti i valori del campo MQMD del messaggio ricevuto vengono copiati nel messaggio inoltrato.

Tabella delle proprietà

Questa tabella elenca le proprietà dell'oggetto Message che rappresentano i campi MQMD. Consultare i collegamenti per una descrizione completa dei campi e dei valori consentiti.

Tabella 38. Nomi, descrizioni e tipi di proprietà

Proprietà	**Descrizione**	**Java Tipo**	**Link alla descrizione completa**
Report MQMD_IBM_JMS	Opzioni per messaggi di report	Numero intero	Prospetto
JMS_IBM_MQMD_MsgType	Tipo messaggio	Numero intero	MsgType
JMS_IBM_MQMD_Expiry	Durata messaggio	Numero intero	Scadenza
Feedback MQMD_IBM_JMS	Feedback o codice motivo	Numero intero	Feedback
Codifica JMS_IBM_MQMD_	Codifica numerica dei dati di messaggio	Numero intero	Codifica
JMS_IBM_MQMD_CodedCharSetId	CSID (Character set identifier) dei dati del messaggio	Numero intero	CodedCharSetId
Formato_MQMD_IBM_JMS	Nome formato dei dati di messaggio	Stringa	Formato
JMS_IBM_MQMD_Priority [1]	Priorità messaggio	Numero intero	Priorità
Persistenza JMS_IBM_MQMD_	Persistenza messaggio	Numero intero	Persistenza
JMS_IBM_MQMD_MsgId [2]	ID messaggio	Oggetto (byte []) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	Identificativo di correlazione	Oggetto (byte []) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	Contatore backout	Numero intero	BackoutCount

Tabella 38. Nomi, descrizioni e tipi di proprietà (Continua)			
Proprietà	**Descrizione**	**Java Tipo**	**Link alla descrizione completa**
JMS_IBM_MQMD_ReplyToQ	Nome della coda di risposta	Stringa	ReplyToQ
Gestore code JMS_IBM_MQMD_ReplyTo	Nome del gestore code di risposta	Stringa	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	Identificativo utente	Stringa	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Token account	Oggetto (byte []) [4]	AccountingToken
Dati JMS_IBM_MQMD_ApplIdentity	Dati dell'applicazione correlati all'identità	Stringa	ApplIdentityData
Tipo JMS_IBM_MQMD_PutAppl	Tipo di applicazione che inserisce il messaggio	Numero intero	PutApplType
Nome JMS_IBM_MQMD_PutAppl	Nome dell'applicazione che inserisce il messaggio	Stringa	PutApplName
JMS_IBM_MQMD_PutDate	Data in cui il messaggio è stato inserito	Stringa	PutDate
PutTime JMS_IBM_MQMD_	Ora in cui il messaggio è stato inserito	Stringa	PutTime
JMS_IBM_MQMD_ApplOriginDati	Dati dell'applicazione correlati all'origine	Stringa	ApplOriginData
JMS_IBM_MQMD_GroupId	ID gruppo	Oggetto (byte []) [4]	GroupId
Numero JMS_IBM_MQMD_MsgSeq	Numero di sequenza del messaggio logico all'interno del gruppo	Numero intero	MsgSeqNumber
Offset MQMD_IBM_JMS	Offset di dati nel messaggio fisico dall'inizio del messaggio logico	Numero intero	Offset
JMS_IBM_MQMD_MsgFlags	Indicatori di messaggio	Numero intero	MsgFlags
JMS_IBM_MQMD_OriginalLength	Lunghezza del messaggio originale	Numero intero	OriginalLength

1. **Attenzione:** Se si assegna un valore a JMS_IBM_MQMD_Priority non compreso nell'intervallo 0-9, ciò viola la specifica JMS .
2. **Attenzione:** La specifica JMS indica che l'ID messaggio deve essere impostato dal provider JMS e che deve essere univoco o null. Se si assegna un valore a JMS_IBM_MQMD_MsgId, questo valore viene copiato in JMSMessageID. Pertanto, non è impostato dal fornitore JMS e potrebbe non essere univoco: ciò viola la specifica JMS .
3. **Attenzione:** Se si assegna un valore a JMS_IBM_MQMD_CorrelId che inizia con la stringa 'ID:', ciò viola la specifica JMS .

4. **Attenzione:** L'utilizzo delle proprietà dell'array di byte su un messaggio viola la specifica JMS .

Accesso ai dati del messaggio IBM MQ da una applicazione utilizzando IBM MQ classes for JMS
È possibile accedere ai dati completi del messaggio IBM MQ all'interno di un'applicazione utilizzando IBM MQ classes for JMS. Per accedere a tutti i dati, il messaggio deve essere un JMSBytesMessage. Il corpo di JMSBytesMessage include qualsiasi intestazione MQRFH2 , qualsiasi altra intestazione IBM MQ e i seguenti dati del messaggio.

Impostare la proprietà WMQ_MESSAGE_BODY della destinazione su WMQ_MESSAGE_BODY_MQper ricevere tutti i dati del corpo del messaggio in JMSBytesMessage.

Se WMQ_MESSAGE_BODY è impostato su WMQ_MESSAGE_BODY_JMS o WMQ_MESSAGE_BODY_UNSPECIFIED, il corpo del messaggio viene restituito senza l'intestazione JMS MQRFH2 e le proprietà di JMSBytesMessage riflettono le proprietà impostate in RFH2.

Alcune applicazioni non possono utilizzare le funzioni descritte in questo argomento. Se un'applicazione è connessa a un gestore code IBM MQ V6 o se ha impostato PROVIDERVERSION su 6, le funzioni non saranno disponibili.

invio di un messaggio

Quando si inviano messaggi, la propriet ... di destinazione, WMQ_MESSAGE_BODY, ha la precedenza su WMQ_TARGET_CLIENT.

Se WMQ_MESSAGE_BODY viene impostato su WMQ_MESSAGE_BODY_JMS, IBM MQ classes for JMS genera automaticamente un'intestazione MQRFH2 in base alle impostazioni delle proprietà e dei campi di intestazione JMSMessage .

Se WMQ_MESSAGE_BODY è impostato su WMQ_MESSAGE_BODY_MQ, non viene aggiunta alcuna intestazione aggiuntiva al corpo del messaggio

Se WMQ_MESSAGE_BODY è impostato su WMQ_MESSAGE_BODY_UNSPECIFIED, IBM MQ classes for JMS invia un'intestazione MQRFH2 , a meno che WMQ_TARGET_CLIENT non sia impostato su WMQ_TARGET_DEST_MQ. In fase di ricezione, l'impostazione di WMQ_TARGET_CLIENT su WMQ_TARGET_DEST_MQ determina la rimozione di qualsiasi MQRFH2 dal corpo del messaggio.

Nota: JMSBytesMessage e JMSTextMessage non richiedono MQRFH2, mentre JMSStreamMessage, JMSMapMessagee JMSObjectMessage lo richiedono.

WMQ_MESSAGE_BODY_UNSPECIFIED è l'impostazione predefinita per WMQ_MESSAGE_BODYe WMQ_TARGET_DEST_JMS è quella predefinita per WMQ_TARGET_CLIENT.

Se si invia un JMSBytesMessage, è possibile sovrascrivere le impostazioni predefinite per il corpo del messaggio JMS quando viene creato il messaggio IBM MQ . Utilizzare le seguenti proprietà:

- JMS_IBM_Format o JMS_IBM_MQMD_Format: questa proprietà specifica il formato dell'intestazione IBM MQ o del payload dell'applicazione che avvia il corpo del messaggio JMS se non è presente alcuna intestazione WebSphere MQ precedente.
- JMS_IBM_Character_Set o JMS_IBM_MQMD_CodedCharSetId: questa proprietà specifica il CCSID dell'intestazione IBM MQ o del payload dell'applicazione che avvia il corpo del messaggio JMS se non è presente un'intestazione WebSphere MQ precedente.
- JMS_IBM_Encoding o JMS_IBM_MQMD_Encoding: questa proprietà specifica la codifica dell'intestazione IBM MQ o del payload dell'applicazione che avvia il corpo del messaggio JMS se non è presente alcuna intestazione WebSphere MQ precedente.

Se vengono specificati entrambi i tipi di proprietà, le proprietà JMS_IBM_MQMD_* sovrascrivono le proprietà JMS_IBM_* corrispondenti, purché la proprietà di destinazione WMQ_MQMD_WRITE_ENABLED sia impostata su true.

Le differenze effettive tra l'impostazione delle proprietà del messaggio utilizzando JMS_IBM_MQMD_* e JMS_IBM_* sono significative:

1. Le proprietà JMS_IBM_MQMD_* sono specifiche del fornitore IBM MQ JMS .

2. Le proprietà JMS_IBM_MQMD_* sono impostate solo in MQMD. Le proprietà JMS_IBM_* sono impostate in MQMD solo se il messaggio non ha un'intestazione MQRFH2 JMS . Altrimenti, vengono impostati nell'intestazione JMS RFH2 .
3. Le proprietà JMS_IBM_MQMD_* non hanno alcun effetto sulla codifica di testo e numeri scritti in un JMSMessage.

   È probabile che un'applicazione ricevente assuma che i valori di MQMD.Encoding e MQMD.CodedCharSetId corrispondano alla codifica e alla serie di caratteri dei numeri e del testo nel corpo del messaggio. Se vengono utilizzate le proprietà JMS_IBM_MQMD_* , è responsabilità dell'applicazione di invio renderla tale. La codifica e la serie di caratteri di numeri e testo nel corpo del messaggio sono impostate dalle proprietà JMS_IBM_* .

   Il frammento codificato in modo errato in Figura 39 a pagina 254 invia un messaggio codificato nella serie di caratteri 1208, con MQMD.CodedCharSetId impostato su 37.

   a. Invia messaggio codificato in modo errato

```
TextMessage tmo = session.createTextMessage();
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQDestination)destination).setMQMDWriteEnabled(true);
tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
tmo.setText("String one");
producer.send(tmo);
```

   b. La ricezione del messaggio, in base al valore di JMS_IBM_CHARACTER_SET impostato dal valore MQMD.CodedCharSetId:

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

   c. Output risultante:

```
Message is "éÈÊ'>...??>?"
```

Figura 39. Dati di messaggi e MQMD codificati in modo incongruente

Uno dei frammenti di codice in Figura 40 a pagina 254 risulta in un messaggio inserito in una coda o in un argomento, con il suo corpo che contiene il payload dell'applicazione senza che venga aggiunta un'intestazione MQRFH2 generata automaticamente.

1. Impostazione WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Impostazione WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                 setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

Figura 40. Inviare un messaggio con un corpo del messaggio MQ .

ricezione di un messaggio

Se WMQ_MESSAGE_BODY è impostato su WMQ_MESSAGE_BODY_JMS, il tipo e il corpo del messaggio JMS in entrata sono determinati dal contenuto del messaggio WebSphere MQ ricevuto. Il tipo e il corpo del messaggio sono determinati dai campi nell'intestazione MQRFH2 o in MQMD, se non è presente alcun MQRFH2.

Se WMQ_MESSAGE_BODY è impostato su WMQ_MESSAGE_BODY_MQ, il tipo di messaggio JMS in ingresso è JMSBytesMessage. Il corpo del messaggio JMS è costituito dai dati del messaggio restituiti dalla chiamata API MQGET sottostante. La lunghezza del corpo del messaggio è la lunghezza restituita dalla chiamata MQGET . La serie di caratteri e la codifica dei dati nel corpo del messaggio sono determinati dai campi CodedCharSetId e Codifica di MQMD. Il formato dei dati nel corpo del messaggio è determinato dal campo Formato di MQMD

Se WMQ_MESSAGE_BODY è impostato su WMQ_MESSAGE_BODY_UNSPECIFIED, il valore predefinito IBM MQ classes for JMS lo imposta su WMQ_MESSAGE_BODY_JMS.

Quando si riceve un JMSBytesMessage, è possibile decodificarlo facendo riferimento alle seguenti proprietà:

- JMS_IBM_Format o JMS_IBM_MQMD_Format: questa proprietà specifica il formato dell'intestazione IBM MQ o del payload dell'applicazione che avvia il corpo del messaggio JMS se non è presente alcuna intestazione WebSphere MQ precedente.
- JMS_IBM_Character_Set o JMS_IBM_MQMD_CodedCharSetId: questa proprietà specifica il CCSID dell'intestazione IBM MQ o del payload dell'applicazione che avvia il corpo del messaggio JMS se non è presente un'intestazione WebSphere MQ precedente.
- JMS_IBM_Encoding o JMS_IBM_MQMD_Encoding: questa proprietà specifica la codifica dell'intestazione IBM MQ o del payload dell'applicazione che avvia il corpo del messaggio JMS se non è presente alcuna intestazione WebSphere MQ precedente.

Il seguente frammento di codice risulta in un messaggio ricevuto che è un JMSBytesMessage. Indipendentemente dal contenuto del messaggio ricevuto e dal campo del formato del MQMDricevuto, il messaggio è un JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Proprietà di destinazione WMQ_MESSAGE_BODY
WMQ_MESSAGE_BODY determina se un'applicazione JMS elabora MQRFH2 di un messaggio IBM MQ come parte del payload del messaggio (ovvero, come parte del corpo del messaggio JMS).

Tabella 39. Nomi e descrizioni delle proprietà

Proprietà	Forma breve	Descrizione
CORPO_MESSAGGIO_WMQ_	MBODY	Se un'applicazione JMS elabora MQRFH2 di un messaggio IBM MQ come parte del payload del messaggio (ovvero, come parte del corpo del messaggio JMS).

Tabella 40. Nomi proprietà, valori e metodi set			
Proprietà	**Valori validi nello strumento di gestione (valori predefiniti in grassetto)**	**Valori validi nei programmi**	**Imposta metodo**
WMQ_MESSAGE_CORPO	• **NON SPECIFICATO** Durante l'invio, IBM MQ classes for JMS genera o non genera e include un'intestazione MQRFH2 , in base al valore di WMQ_TARGET_CLIENT. Quando si riceve, agisce come valore JMS. • JMS Durante l'invio, IBM MQ classes for JMS genera automaticamente un'intestazione MQRFH2 e la include nel messaggio IBM MQ . Quando si riceve, IBM MQ classes for JMS imposta le proprietà del messaggio JMS in base ai valori in MQRFH2 (se presente); non presenta MQRFH2 come parte del corpo del messaggio JMS . • MQ Durante l'invio, IBM MQ classes for JMS non genera un MQRFH2. Quando si riceve, IBM MQ classes for JMS presenta MQRFH2 come parte del contenuto del messaggio JMS .	• **WMQ_MESSAGE_ BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • MESSAGE_BODY_MQ WMQ_	setMessageBodySty le

JMS Messaggi persistenti
Le applicazioni IBM MQ classes for JMS possono utilizzare l'attributo della coda **NonPersistentMessageClass** per fornire prestazioni migliori per i messaggi persistenti JMS , a discapito di una certa affidabilità.

Una coda IBM MQ ha un attributo denominato **NonPersistentMessageClass**. Il valore di questo attributo determina se i messaggi non persistenti sulla coda vengono eliminati al riavvio del gestore code.

È possibile impostare l'attributo per una coda locale utilizzando il comando MQSC (IBM MQ Script), DEFINE QLOCAL, con uno dei seguenti parametri:

NPMCLASS (NORMALE)
I messaggi non persistenti sulla coda vengono eliminati al riavvio del gestore code. Questo è il valore predefinito.

NPMCLASS (ALTO)
I messaggi non persistenti sulla coda non vengono eliminati quando il gestore code viene riavviato in seguito a un arresto inattivo o immediato. I messaggi non persistenti potrebbero essere eliminati, tuttavia, a seguito di un arresto preventivo o di un errore.

Questo argomento descrive in che modo le applicazioni IBM MQ classes for JMS possono utilizzare questo attributo della coda per fornire prestazioni migliori per i messaggi persistenti JMS .

La proprietà PERSISTENCE di un oggetto Coda o Argomento può avere il valore HIGH. È possibile utilizzare lo strumento di amministrazione IBM MQ JMS per impostare questo valore oppure un'applicazione può richiamare il metodo Destination.setPersistence() passando il valore WMQConstants.WMQ_PER_NPHIGH come parametro.

Se un'applicazione invia un messaggio persistente JMS o un messaggio non persistente JMS ad una destinazione in cui la proprietà PERSISTENCE ha il valore HIGH e la coda IBM MQ sottostante è impostata su NPMCLASS (HIGH), il messaggio viene inserito nella coda come un IBM MQ messaggio non persistente. Se la proprietà PERSISTENCE della destinazione non ha il valore HIGH, o se la coda sottostante è impostata su NPMCLASS (NORMAL), un messaggio persistente JMS viene inserito nella coda come un messaggio persistente IBM MQ e un messaggio non persistente JMS viene inserito sulla coda come un messaggio non persistente IBM MQ .

Se un messaggio persistente JMS viene inserito su una coda come un IBM MQ messaggio non persistente e si desidera assicurarsi che il messaggio non venga eliminato dopo un arresto inattivo o immediato di un gestore code, tutte le code attraverso le quali il messaggio potrebbe essere instradato devono essere impostate su NPMCLASS (HIGH). Nel dominio di pubblicazione / sottoscrizione, queste code includono le code del sottoscrittore. Per facilitare l'applicazione di questa configurazione, IBM MQ classes for JMS genera un'eccezione InvalidDestinationse un'applicazione tenta di creare un consumatore di messaggi per una destinazione in cui la proprietà PERSISTENCE ha il valore HIGH e la coda IBM MQ sottostante è impostata su NPMCLASS (NORMAL).

L'impostazione della proprietà PERSISTENCE di una destinazione su HIGH non influisce sulla modalità di ricezione di un messaggio da tale destinazione. Un messaggio inviato come un messaggio persistente JMS viene ricevuto come un messaggio persistente JMS e un messaggio inviato come un messaggio non persistente JMS viene ricevuto come un messaggio non persistente JMS .

Quando un'applicazione invia il primo messaggio ad una destinazione in cui la proprietà PERSISTENCE ha il valore HIGH, o quando un'applicazione crea il primo consumatore di messaggi per una destinazione in cui la proprietà PERSISTENCE ha il valore HIGH, IBM MQ classes for JMS emette una chiamata MQINQ per determinare se NPMCLASS (HIGH) è impostato sulla coda IBM MQ sottostante. L'applicazione deve quindi disporre dell'autorità per richiedere informazioni sulla coda. Inoltre, IBM MQ classes for JMS conserva il risultato della chiamata MQINQ fino a quando la destinazione non viene eliminata e non emette ulteriori chiamate MQINQ. Pertanto, se si modifica l'impostazione NPMCLASS sulla coda sottostante mentre l'applicazione sta ancora utilizzando la destinazione, IBM MQ classes for JMS non nota la nuova impostazione.

Consentendo l'inserimento dei messaggi persistenti JMS nelle code IBM MQ come IBM MQ messaggi non persistenti, si ottengono prestazioni a scapito di una certa affidabilità. Se si richiede la massima affidabilità per i messaggi persistenti JMS , non inviare i messaggi ad una destinazione in cui la proprietà PERSISTENCE ha il valore HIGH.

Il livello JMS può utilizzare SYSTEM.JMS.TEMPQ.MODEL, invece di SYSTEM.DEFAULT.MODEL.QUEUE. SYSTEM.JMS.TEMPQ.MODEL crea code dinamiche permanenti che accettano messaggi persistenti, perché SYSTEM.DEFAULT.MODEL.QUEUE non può accettare messaggi persistenti. Per utilizzare le code temporanee per accettare i messaggi persistenti, è necessario utilizzare SYSTEM.JMS.TEMPQ.MODELo modificare la coda modello in una coda alternativa di propria scelta.

Utilizzo di TLS con IBM MQ classes for JMS
Le applicazioni IBM MQ classes for JMS possono utilizzare la crittografia TLS (Transport Layer Security). Per fare ciò, è necessario un fornitore JSSE.

Le connessioni IBM MQ classes for JMS che utilizzano TRANSPORT (CLIENT) supportano la codifica TLS. TLS fornisce la codifica di comunicazione, l'autenticazione e l'integrità del messaggio. Generalmente viene utilizzato per proteggere le comunicazioni tra due peer su Internet o all'interno di una intranet.

IBM MQ classes for JMS utilizza JSSE (Java Secure Socket Extension) per la gestione della codifica TLS e pertanto richiede un fornitore JSSE. Le JVM JSE v1.4 hanno un provider JSSE integrato. I dettagli su come gestire e memorizzare i certificati possono variare da fornitore a fornitore. Per informazioni, consultare la documentazione del provider JSSE.

Questa sezione presuppone che il provider JSSE sia installato e configurato correttamente e che i certificati appropriati siano stati installati e resi disponibili per il proprio provider JSSE. È ora possibile utilizzare JMSAdmin per impostare un certo numero di proprietà amministrative.

Se l'applicazione IBM MQ classes for JMS utilizza una tabella CCDT (client channel definition table) per connettersi a un gestore code, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for JMS” a pagina 286.

Proprietà oggetto SSLCIPHERSUITE
Impostare SSLCIPHERSUITE per abilitare la codifica TLS su un oggetto ConnectionFactory .

Per abilitare la codifica TLS su un oggetto ConnectionFactory , utilizzare JMSAdmin per impostare la proprietà SSLCIPHERSUITE su una CipherSuite supportata dal provider JSSE. Deve corrispondere all'impostazione CipherSpec sul canale di destinazione. Tuttavia, CipherSuites sono distinte da CipherSpecs e, pertanto, hanno nomi differenti. “TLS CipherSpecs e CipherSuites in IBM MQ classes for JMS” a pagina 261 contiene una tabella che associa i CipherSpecs supportati da IBM MQ ai CipherSuites equivalenti noti a JSSE. Per ulteriori informazioni su CipherSpecs e CipherSuites con IBM MQ, consultare Sicurezza IBM MQ.

Ad esempio, per impostare un oggetto ConnectionFactory che può essere utilizzato per creare una connessione su un canale MQI abilitato TLS con un CipherSpec di TLS_RSA_WITH_AES_128_CBC_SHA, immettere il seguente comando in JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Questo può essere impostato anche da un'applicazione, utilizzando il metodo setSSLCipherSuite () su un oggetto MQConnectionFactory .

Per comodità, se viene specificato un CipherSpec nella proprietà SSLCIPHERSUITE, JMSAdmin tenta di associare CipherSpec a una CipherSuite appropriata e invia un'avvertenza. Questo tentativo di associazione non viene effettuato se la proprietà è specificata da un'applicazione.

In alternativa, utilizzare CCDT (Client Channel Definition Table). Per ulteriori informazioni, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for JMS” a pagina 286.

proprietà oggetto SSLFIPSREQUIRED
Se si richiede una connessione per utilizzare una CipherSuite supportata dal provider FIPS JSSE IBM Java (IBMJSSEFIPS), impostare la proprietà SSLFIPSREQUIRED del factory di connessione su YES.

Nota: Su AIX, Linux, and Windows, IBM MQ fornisce la conformità FIPS 140-2 tramite il modulo crittografico IBM Crypto for C (ICC) . Il certificato per questo modulo è stato spostato nello stato cronologico. I clienti devono visualizzare il certificatoIBM Crypto for C (ICC) ed essere a conoscenza di eventuali consigli forniti da NIST. Un modulo FIPS 140-3 di sostituzione è attualmente in corso e il relativo stato può essere visualizzato ricercandolo in NIST CMVP modules in process list.

Il valore predefinito di questa proprietà è NO, che significa che una connessione può utilizzare qualsiasi CipherSuite supportato da IBM MQ.

Se un'applicazione utilizza più di una connessione, il valore di SSLFIPSREQUIRED utilizzato quando l'applicazione crea la prima connessione determina il valore utilizzato quando l'applicazione crea una connessione successiva. Ciò significa che il valore della proprietà SSLFIPSREQUIRED della factory di connessione utilizzata per creare una connessione successiva viene ignorato. È necessario riavviare l'applicazione se si desidera utilizzare un valore diverso di SSLFIPSREQUIRED.

Un'applicazione può impostare questa proprietà richiamando il metodo setSSLFipsRequired () di un oggetto ConnectionFactory . La proprietà viene ignorata se non è impostata alcuna CipherSuite .

Attività correlate

Specifica che solo i CipherSpecs certificati FIPS vengono utilizzati al runtime sul client MQI

Riferimenti correlati

FIPS (Federal Information Processing Standards) per AIX, Linux, and Windows

proprietà oggetto SSLPEERNAME

Utilizzare SSLPEERNAME per specificare un modello di DN (distinguished name), per garantire che l'applicazione JMS si connetta al gestore code corretto.

Un'applicazione JMS può garantire la connessione al gestore code corretto specificando un modello di DN (distinguished name). La connessione ha esito positivo solo se il gestore code presenta un DN che corrisponde al pattern. Per ulteriori dettagli sul formato di questo pattern, consultare gli argomenti correlati.

Il DN viene impostato utilizzando la proprietà SSLPEERNAME di un oggetto ConnectionFactory . Ad esempio, il seguente comando JMSAdmin imposta un oggetto ConnectionFactory per prevedere che il gestore code si identifichi con un Nome comune che inizia con i caratteri `QMGR.` e con almeno due nomi di unità organizzative, il primo dei quali deve essere IBM e il secondo WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Il controllo non è sensibile al maiuscolo / minuscolo e i punti e virgola possono essere utilizzati al posto delle virgole. SSLPEERNAME può essere impostato anche da un'applicazione utilizzando il metodo setSSLPeerName () su un oggetto MQConnectionFactory . Se questa proprietà non è impostata, non viene eseguito alcun controllo sul DN (Distinguished Name) fornito dal gestore code. Questa proprietà viene ignorata se non è impostata alcuna CipherSuite .

Proprietà oggetto SSLCERTSTORES

Utilizzare SSLCERTSTORES per specificare un elenco di server LDAP da utilizzare per il controllo CRL (Certificate Revocation List).

È comune utilizzare un CRL (Certificate Revocation List) per identificare i certificati non più attendibili. I CRL sono generalmente ospitati su server LDAP. JMS consente a un server LDAP di essere specificato per il controllo CRL in Java 2 v1.4 o versioni successive. Il seguente esempio JMSAdmin indica a JMS di utilizzare un CRL presente su un server LDAP denominato crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Nota: Per utilizzare correttamente un CertStore con un CRL ospitato su un server LDAP, assicurati che il tuo SDK (Software Development Kit) Java sia compatibile con il CRL. Alcuni SDK richiedono che il CRL sia conforme a RFC 2587, che definisce uno schema per LDAP v2. La maggior parte dei server LDAP v3 utilizza invece RFC 2256.

Se il server LDAP non è in esecuzione sulla porta predefinita 389, è possibile specificare la porta accodando i due punti (:) e il numero di porta al nome host. Se il certificato presentato dal gestore code è presente nel CRL ospitato su crl1.ibm.com, la connessione non viene completata. Per evitare un singolo punto di errore, JMS consente di fornire più server LDAP fornendo un elenco di server LDAP delimitati dal carattere spazio. Di seguito è riportato un esempio:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

Quando vengono specificati più server LDAP, JMS prova ciascuno a turno finché non trova un server con cui è possibile verificare correttamente il certificato del gestore code. Ciascun server deve contenere informazioni identiche.

Una stringa in questo formato può essere fornita da un'applicazione sul metodo MQConnectionFactory.setSSLCertStores (). In alternativa, l'applicazione può creare uno o più oggetti java.security.cert.CertStore , posizionarli in un oggetto Collection adatto e fornire questo oggetto Collection al metodo setSSLCertStores (). In questo modo, l'applicazione può personalizzare la verifica CRL. Consultare la documentazione JSSE per i dettagli sulla creazione e l'utilizzo degli oggetti CertStore .

Il certificato presentato dal gestore code durante l'impostazione di una connessione viene convalidato nel modo seguente:

1. Il primo oggetto CertStore nella raccolta identificata da sslCertStores viene utilizzato per identificare un server CRL.
2. È stato effettuato un tentativo di contattare il server CRL.
3. Se il tentativo ha esito positivo, il server viene ricercato per una corrispondenza per il certificato.
   a. Se viene rilevato che il certificato è stato revocato, il processo di ricerca è stato completato e la richiesta di connessione ha esito negativo con codice motivo MQRC_SSL_CERTIFICATE_REVOKED.
   b. Se il certificato non viene trovato, il processo di ricerca è stato completato e la connessione può continuare.
4. Se il tentativo di contattare il server non ha esito positivo, il successivo oggetto CertStore viene utilizzato per identificare un server CRL e il processo si ripete dal passo 2.

   Se questo era l'ultimo CertStore nella raccolta o se la raccolta non contiene oggetti CertStore , il processo di ricerca ha avuto esito negativo e la richiesta di connessione ha avuto esito negativo con codice motivo MQRC_SSL_CERT_STORE_ERROR.

L'oggetto Raccolta determina l'ordine in cui vengono utilizzati i CertStores .

Se l'applicazione utilizza setSSLCertStores () per impostare una raccolta di oggetti CertStore , non è più possibile eseguire il bind di MQConnectionFactory in uno spazio dei nomi JNDI. Il tentativo di eseguire questa operazione causa un'eccezione. Se la proprietà sslCertStores non è impostata, non viene eseguito alcun controllo di revoca sul certificato fornito dal gestore code. Questa proprietà viene ignorata se non è impostata alcuna CipherSuite .

proprietà oggetto SSLRESETCOUNT
Questa proprietà rappresenta il numero totale di byte inviati e ricevuti da una connessione prima che la chiave segreta utilizzata per la codifica venga rinegoziata.

Il numero di byte inviati è il numero prima della codifica e il numero di byte ricevuti è il numero dopo la decodifica. Il numero di byte include anche le informazioni di controllo inviate e ricevute da IBM MQ classes for JMS.

Ad esempio, per configurare un oggetto ConnectionFactory che può essere utilizzato per creare una connessione su un canale MQI abilitato a TLS con una chiave segreta che viene rinegoziata dopo il flusso di 4 MB di dati, immettere il seguente comando a JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Un'applicazione può impostare questa proprietà richiamando il metodo setSSLResetCount () di un oggetto ConnectionFactory .

Se il valore di questa proprietà è zero, che è il valore predefinito, la chiave segreta non viene mai rinegoziata. La proprietà viene ignorata se non è impostata alcuna CipherSuite .

Proprietà oggetto SSLSocketFactory
Per personalizzare altri aspetti della connessione TLS per una applicazione, creare un SSLSocketFactory e configurare JMS per utilizzarlo.

È possibile personalizzare altri aspetti della connessione TLS per un'applicazione. Ad esempio, è possibile che si desideri inizializzare l'hardware crittografico o modificare il keystore e il truststore in uso. A tale scopo, l'applicazione deve prima creare un oggetto javax.net.ssl.SSLSocketFactory personalizzato di conseguenza. Consultare la documentazione JSSE per informazioni su come eseguire questa

operazione, poiché le funzioni personalizzabili variano da provider a provider. Dopo aver ottenuto un oggetto SSLSocketFactory adatto, utilizzare il metodo MQConnectionFactory.setSSLSocketFactory () per configurare JMS per utilizzare l'oggetto SSLSocketFactory personalizzato.

Se la propria applicazione utilizza il metodo setSSLSocketFactory () per impostare un oggetto SSLSocketFactory personalizzato, l'oggetto MQConnectionFactory non può più essere collegato in uno spazio dei nomi JNDI. Il tentativo di eseguire questa operazione causa un'eccezione. Se questa proprietà non è impostata, viene utilizzato l'oggetto predefinito SSLSocketFactory . Consultare la propria documentazione JSSE per i dettagli sul comportamento dell'oggetto SSLSocketFactory predefinito. Questa proprietà viene ignorata se non è impostata alcuna CipherSuite .

Importante: Non presupporre che l'utilizzo delle proprietà SSL assicuri la sicurezza quando un oggetto ConnectionFactory viene richiamato da uno spazio nomi JNDI che non è esso stesso sicuro. In particolare, l'implementazione LDAP standard di JNDI non è sicura. Un aggressore può imitare il server LDAP, inducendo in modo fuorviante un'applicazione JMS a connettersi al server sbagliato senza accorgersene. Con adeguate disposizioni di sicurezza in atto, altre implementazioni di JNDI (come l'implementazione fscontext) sono sicure.

Esecuzione di modifiche al keystore o al truststore JSSE
Se si apportano modifiche al keystore o al truststore, è necessario eseguire determinate azioni per rendere effettive le modifiche.

Se si modifica il contenuto del keystore o del truststore JSSE o si modifica l'ubicazione del file keystore o truststore, le applicazioni IBM MQ classes for JMS in esecuzione al momento non acquisiscono automaticamente le modifiche. Per rendere effettive le modifiche, è necessario eseguire le seguenti operazioni:

- Le applicazioni devono chiudere tutte le relative connessioni ed eliminare tutte le connessioni inutilizzate nei pool di connessioni.
- Se il provider JSSE memorizza nella cache le informazioni dal keystore e dal truststore, tali informazioni devono essere aggiornate.

Una volta eseguite queste azioni, le applicazioni possono ricreare le connessioni.

A seconda della modalità di progettazione delle applicazioni e della funzione fornita dal provider JSSE, potrebbe essere possibile eseguire queste azioni senza arrestare e riavviare le applicazioni. Tuttavia, l'arresto e il riavvio delle applicazioni potrebbe essere la soluzione più semplice.

TLS CipherSpecs e CipherSuites in IBM MQ classes for JMS
La capacità delle applicazioni IBM MQ classes for JMS di stabilire connessioni a un gestore code dipende dalla CipherSpec specificata all'estremità server del canale MQI e dalla CipherSuite specificata all'estremità client.

La seguente tabella elenca i CipherSpecs supportati da IBM MQ e i relativi CipherSuitesequivalenti.

Deprecated È necessario esaminare l'argomento Deprecated CipherSpecs per verificare se uno dei CipherSpecs, elencati nella seguente tabella, è stato dichiarato obsoleto da IBM MQ e, in tal caso, è stato reso obsoleto l'aggiornamento di CipherSpec .

Importante: Le CipherSuites elencate sono quelle supportate da IBM Java Runtime Environment (JRE) fornito con IBM MQ. Le CipherSuites elencate includono quelle supportate da JRE Oracle Java . Per ulteriori informazioni sulla configurazione della tua applicazione per utilizzare un JRE Oracle Java , vedi Configurazione della tua applicazione per utilizzare le associazioni IBM Java o Oracle Java CipherSuite.

La tabella indica anche il protocollo utilizzato per le comunicazioni e se CipherSuite è conforme allo standard FIPS 140-2.

Nota: Su AIX, Linux, and Windows, IBM MQ fornisce la conformità FIPS 140-2 tramite il modulo crittografico IBM Crypto for C (ICC) . Il certificato per questo modulo è stato spostato nello stato cronologico. I clienti devono visualizzare il certificatoIBM Crypto for C (ICC) ed essere a conoscenza di eventuali consigli forniti da NIST. Un modulo FIPS 140-3 di sostituzione è attualmente in corso e il relativo stato può essere visualizzato ricercandolo in NIST CMVP modules in process list.

Le suite di cifratura indicate come conformi a FIPS 140-2 possono essere utilizzate se l'applicazione non è stata configurata per applicare la conformità FIPS 140-2, ma se la conformità FIPS 140-2 è stata configurata per l'applicazione (vedere le seguenti note sulla configurazione), è possibile configurare solo le CipherSuites contrassegnate come compatibili con FIPS 140-2; il tentativo di utilizzare altre CipherSuites genera un errore.

Nota: Ogni JRE può avere più provider di sicurezza crittografica, ognuno dei quali può contribuire con un'implementazione della stessa CipherSuite. Tuttavia, non tutti i provider di sicurezza sono certificati FIPS 140-2. Se la conformità FIPS 140-2 non viene applicata per un'applicazione, è possibile che venga utilizzata un'implementazione non certificata di CipherSuite . Le implementazioni non certificate potrebbero non essere conformi a FIPS 140-2, anche se CipherSuite in teoria soddisfa il livello di sicurezza minimo richiesto dallo standard. Per ulteriori informazioni sulla configurazione dell'applicazione FIPS 140-2 nelle applicazioni IBM MQ JMS , consultare le seguenti note.

Per ulteriori informazioni sulla conformità FIPS 140-2 e Suite - B per CipherSpecs e CipherSuites, consultare Specifica di CipherSpecs. Potrebbe anche essere necessario essere a conoscenza delle informazioni relative a Federal Information Processing Standardsdegli Stati Uniti.

Per utilizzare la serie completa di CipherSuites e per operare con la conformità FIPS 140-2 e / o Suite - B certificata, è richiesto un JRE adatto. IBM Java 7 Service Refresh 4 Fix Pack 2 o un livello superiore di IBM JRE fornisce il supporto appropriato per il TLS 1.2 CipherSuites elencato in Tabella 41 a pagina 263.

Per poter utilizzare le crittografie TLS 1.3 , il JRE che esegue l'applicazione deve supportare TLS 1.3.

Nota: Per utilizzare alcuni CipherSuites, i file della politica 'unrestricted' devono essere configurati in JRE. Per ulteriori dettagli su come vengono configurati i file delle politiche in un SDK o JRE, consultare l'argomento *IBM SDK Policy files* nel manuale *Security Reference for IBM SDK, Java Technology Edition* per la versione che si sta utilizzando.

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sì
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	no

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sì

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sì
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	no

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no
TLS_RSA_WITH_3DES_EDE_CBC_SHA "2" a pagina 281	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	no "4" a pagina 281

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	no "4" a pagina 281
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	no "4" a pagina 281

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	no "4" a pagina 281
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	no "4" a pagina 281

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	no "4" a pagina 281
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	no "4" a pagina 281

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	no
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	no
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	sì
TLS_AES_128_GCM_SHA256 "3" a pagina 281	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS v1.3	no
TLS_AES_256_GCM_SHA384 "3" a pagina 281	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS v1.3	no

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_CHACHA20_POLY1305_SHA256 "3" a pagina 281	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS v1.3	no
TLS_AES_128_CCM_SHA256 "3" a pagina 281	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS v1.3	no

Tabella 41. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 281**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_AES_128_CCM_8_SHA256 "3" a pagina 281	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS v1.3	no
Qualsiasi"3" a pagina 281	*ANY	*ANY	Multipli/e	no
ANY_TLS13 "3" a pagina 281	*TLS13	*TLS13	TLS V13	no
ANY_TLS12_OR_HIGHER "3" a pagina 281	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 e versioni successive	no
ANY_TLS13_OR_HIGHER "3" a pagina 281	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 e versioni successive	no

Note:

1. Questo è il valore configurato su un canale in IBM MQ, incluso in un CCDT (binario o JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleto. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.
3. Per poter utilizzare le crittografie TLS v1.3 , il JRE (Java runtime environment) che esegue la tua applicazione deve supportare TLS v1.3.
4. V 9.3.0.17 V 9.3.5.1 Da IBM MQ 9.3.5 CSU 1 e IBM MQ 9.3.0 CSU 17, IBM Java 8 JRE rimuove il supporto per lo scambio di chiavi RSA quando si opera in modalità FIPS.

Configurazione di Ciphersuites e FIPS - compliance in un'applicazione IBM MQ classes for JMS

- Un'applicazione che utilizza IBM MQ classes for JMS può utilizzare uno dei due metodi per impostare CipherSuite per una connessione:
  - Richiamare il metodo setSSLCipherSuite di un oggetto ConnectionFactory .
  - Utilizzare lo strumento di amministrazione IBM MQ JMS per impostare la proprietà SSLCIPHERSUITE di un oggetto ConnectionFactory .
- Un'applicazione che utilizza IBM MQ classes for JMS può utilizzare uno dei due metodi per applicare la conformità FIPS 140-2:
  - Richiamare il metodo richiesto setSSLFipsdi un oggetto ConnectionFactory .
  - Utilizzare lo strumento di gestione IBM MQ JMS per impostare la proprietà SSLFIPSREQUIRED di un oggetto ConnectionFactory .

Configurazione della tua applicazione per utilizzare le associazioni IBM Java o Oracle Java CipherSuite

Nota: V 9.3.3 Per Continuous Delivery da IBM MQ 9.3.3, la Java Proprietà di sistema `com.ibm.mq.cfg.useIBMCipherMappings`, che controlla le associazioni utilizzate, viene rimossa dal prodotto. Da IBM MQ 9.3.3, una cifratura può essere definita come il nome CipherSpec o CipherSuite ed è gestita correttamente da IBM MQ. I seguenti tre file JAR Jackson sono obbligatori se l'applicazione IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging crea connessioni TLS sicure a un gestore code:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

Importante: Le seguenti informazioni su `com.ibm.mq.cfg.useIBMCipherMappings` si applicano solo a Long Term Support e Continuous Delivery prima di IBM MQ 9.3.3 .

È possibile configurare se la propria applicazione utilizza le associazioni IBM Java CipherSuite a IBM MQ CipherSpec predefinite oppure le associazioni Oracle CipherSuite a IBM MQ CipherSpec . Pertanto, puoi utilizzare TLS CipherSuites se la tua applicazione utilizza un JRE IBM o un JRE Oracle . La Java Proprietà di sistema `com.ibm.mq.cfg.useIBMCipherMappings` controlla quali associazioni vengono utilizzate. La proprietà può essere uno dei seguenti valori:

vero, true
Utilizzare le associazioni IBM Java CipherSuite a IBM MQ CipherSpec .

Questo è il valore predefinito.

No
Utilizzare le associazioni Oracle CipherSuite a IBM MQ CipherSpec .

Per ulteriori informazioni sull'utilizzo dei codici IBM MQ Java e TLS, consultare il post del blog MQdev MQ Java, TLS Ciphers, Non -IBM JREs & APARs IT06775, IV66840, IT09423, IT10837.

Limitazioni di interoperabilità

Alcune CipherSuites potrebbero essere compatibili con più di una IBM MQ CipherSpec, in base al protocollo in uso. Tuttavia, è supportata solo la combinazione CipherSuite/CipherSpec che utilizza la versione di TLS specificata nella tabella 1. Il tentativo di utilizzare le combinazioni non supportate di CipherSuites e CipherSpecs avrà esito negativo con un'eccezione appropriata. Le installazioni che utilizzano una di queste combinazioni CipherSuite/CipherSpec devono essere spostate in una combinazione supportata.

La seguente tabella mostra CipherSuites a cui si applica questa limitazione.

Tabella 42. CipherSuites e i CipherSpecs supportati e non supportati

CipherSuite	CipherSpec TLS supportato	CipherSpec SSL non supportato
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" a pagina 282	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Nota:

1. Deprecated Questa CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleta. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.

Scrittura di uscite di canale in Java per IBM MQ classes for JMS
Le uscite canale vengono create definendo le classi Java che implementano le interfacce specificate.

Per un'introduzione alle uscite di sicurezza, iniziare con l'argomento Programmi di uscita di sicurezza del canale .

Tre interfacce sono definite nel package com.ibm.mq.exits :

- WMQSendExit, per un'uscita di invio
- WMQReceiveExit, per un'uscita di ricezione
- WMQSecurityExit, per un'uscita di sicurezza

Il seguente codice di esempio definisce una classe che implementa tutte e tre le interfaccia:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
```

```
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the security exit here
  }
}
```

Ogni uscita riceve come parametri un oggetto MQCXP e un oggetto MQCD. Questi oggetti rappresentano le strutture MQCXP e MQCD definite nell'interfaccia procedurale.

Quando si chiama un'uscita di invio, il parametro agentBuffer contiene i dati che verranno inviati al gestore code del server. Un parametro di lunghezza non è richiesto perché l'espressione agentBuffer.limit () fornisce la lunghezza dei dati. L'uscita di invio restituisce come valore i dati da inviare al gestore code del server. Tuttavia, se l'uscita di invio non è l'ultima uscita di invio in una sequenza di uscite di invio, i dati restituiti vengono passati invece alla successiva uscita di invio nella sequenza. Un'uscita di invio può restituire una versione modificata dei dati che riceve nel parametro agentBuffer oppure può restituire i dati invariati. Il corpo di uscita più semplice possibile è quindi:

```
{ return agentBuffer; }
```

Quando viene richiamata un'uscita di ricezione, il parametro agentBuffer contiene i dati ricevuti dal gestore code del server. L'uscita di ricezione restituisce come valore i dati che devono essere passati all'applicazione da IBM MQ classes for JMS. Tuttavia, se l'uscita di ricezione non è l'ultima uscita di ricezione in una sequenza di uscite di ricezione, i dati restituiti vengono trasmessi invece alla successiva uscita di ricezione nella sequenza.

Quando viene richiamata un'uscita di sicurezza, il parametro agentBuffer contiene i dati ricevuti in un flusso di sicurezza dall'uscita di sicurezza alla fine della connessione del server. L'uscita di sicurezza restituisce come valore i dati da inviare in un flusso di sicurezza all'uscita di sicurezza del server.

Le uscite del canale vengono richiamate con un buffer che ha un array di supporto. Per prestazioni ottimali, l'uscita deve restituire un buffer con un array di backup.

È possibile passare fino a 32 caratteri di dati utente a un'uscita canale quando viene richiamata. L'uscita accede ai dati utente richiamando il metodo getExitData () dell'oggetto MQCXP. Sebbene l'uscita possa modificare i dati utente richiamando il metodo setExitData (), i dati utente vengono aggiornati ogni volta che viene richiamata l'uscita. Tutte le modifiche apportate ai dati dell'utente vengono quindi perse. Tuttavia, l'uscita può trasmettere dati da una chiamata alla successiva utilizzando l'area utente di uscita dell'oggetto MQCXP. L'uscita accede all'area utente di uscita per riferimento richiamando il metodo getExitUserArea().

Ogni classe di uscita deve avere un costruttore. Il costruttore può essere il costruttore predefinito, come mostrato nell'esempio precedente, oppure un costruttore con un parametro stringa. Il costruttore viene richiamato per creare un'istanza della classe di uscita per ogni uscita definita nella classe. Pertanto, nell'esempio precedente, viene creata un'istanza della classe MyMQExits per l'uscita di invio, un'altra istanza per l'uscita di ricezione e una terza istanza per l'uscita di sicurezza. Quando viene richiamato un costruttore con un parametro stringa, il parametro contiene gli stessi dati utente passati all'uscita canale per cui viene creata l'istanza. Se una classe di uscita ha sia un costruttore predefinito che un singolo costruttore di parametri, il singolo costruttore di parametri ha la precedenza.

Non chiudere la connessione dall'interno di un'uscita canale.

Quando i dati vengono inviati all'estremità del server di una connessione, la crittografia TLS viene eseguita *dopo* che vengono richiamate tutte le uscite del canale. Allo stesso modo, quando i dati vengono ricevuti dall'estremità del server di una connessione, la decodifica TLS viene eseguita *prima* che vengano richiamate tutte le uscite del canale.

Nelle versioni di IBM MQ classes for JMS precedenti a IBM WebSphere MQ 7.0, le uscite dei canali sono state implementate utilizzando le interfacce MQSendExit, MQReceiveExite MQSecurityExit. È ancora

possibile utilizzare queste interfacce, ma le nuove interfacce sono preferite per migliorare le funzioni e le prestazioni.

Configurazione di IBM MQ classes for JMS per l'utilizzo delle uscite canale
Un'applicazione IBM MQ classes for JMS può utilizzare la sicurezza del canale, inviare e ricevere uscite sul canale MQI che viene avviato quando l'applicazione si connette a un gestore code. L'applicazione può utilizzare le uscite scritte in Java, C o C + +. L'applicazione può anche utilizzare una sequenza di uscite di invio o ricezione eseguite in successione.

Le seguenti proprietà vengono utilizzate per specificare un'uscita di invio o una sequenza di uscite di invio, utilizzata da una connessione JMS :

- La proprietà **SENDEXIT** di un oggetto MQConnectionFactory .
- La proprietà **sendexit** su una specifica di attivazione utilizzata dall'adattatore di risorse IBM MQ per la comunicazione in entrata,
- La proprietà **sendexit** in un oggetto ConnectionFactory utilizzato dall'adattatore di risorse IBM MQ per la comunicazione di output.

Il valore della proprietà è una stringa che comprende uno o più elementi separati da virgole. Ogni voce identifica un'uscita di invio in uno dei seguenti modi:

- Il nome di una classe che implementa l'interfaccia `WMQSendExit` per un'uscita di invio scritta in Java.
- Una stringa nel formato *libraryName* (*entryPointName*) per un'uscita di invio scritta in C o C + +.

In modo simile, le seguenti proprietà specificano l'uscita di ricezione o la sequenza di uscite di ricezione utilizzata da una connessione:

- La proprietà **RECEXIT** di un oggetto MQConnectionFactory .
- La proprietà **receiveexit** su una specifica di attivazione utilizzata dall'adattatore di risorse IBM MQ per la comunicazione in entrata,
- La proprietà **receiveexit** in un oggetto ConnectionFactory utilizzato dall'adattatore di risorse IBM MQ per la comunicazione di output.

Le seguenti proprietà specificano l'uscita di sicurezza utilizzata da una connessione:

- La proprietà **SECEXIT** di un oggetto MQConnectionFactory .
- La proprietà **securityexit** su una specifica di attivazione utilizzata dall'adattatore di risorse IBM MQ per la comunicazione in entrata,
- La proprietà **securityexit** in un oggetto ConnectionFactory utilizzato dall'adattatore di risorse IBM MQ per la comunicazione di output.

Per MQConnectionFactories, è possibile impostare le proprietà **SENDEXIT**, **RECEXIT** e **SECEXIT** utilizzando lo strumento di amministrazione IBM MQ JMS o IBM MQ Explorer. In alternativa, un'applicazione può impostare le proprietà richiamando i metodi `setSendExit()`, `setReceiveExit()`e `setSecurityExit()` .

Le uscite canale vengono caricate dal proprio programma di caricamento classi. Per trovare un'uscita canale, il programma di caricamento classi ricerca le seguenti ubicazioni nell'ordine specificato.

1. Il percorso classe specificato dalla proprietà **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** o dall'attributo **JavaExitsClassPath** nella sezione Canali del file di configurazione del client IBM MQ .
2. Deprecated Il percorso classe specificato dalla proprietà di sistema Java **com.ibm.mq.exitClasspath**. Notare che questa proprietà è ora obsoleta.
3. La directory delle uscite IBM MQ , come mostrato in Tabella 43 a pagina 285. Il programma di caricamento classi ricerca nella directory i file di classe che non sono compressi nei file JAR (Java archive). Se l'uscita del canale non viene trovata, il programma di caricamento classe ricerca i file JAR nella directory.

Tabella 43. La directory delle uscite IBM MQ	
Piattaforma	**Directory**
Linux AIX AIX and Linux	`/var/mqm/exits` (uscite canale a 32 bit) `/var/mqm/exits64` (uscite canale a 64 bit)
Windows Windows	*dir_dati_installazione*\exits dove *install_data_dir* è la directory scelta per i file di dati IBM MQ durante l'installazione. La directory predefinita è C:\ProgramData\IBM\MQ.

Nota: Se un'uscita del canale esiste in più di una ubicazione, IBM MQ classes for JMS carica la prima istanza che trova.

Il parent del programma di caricamento classi è il programma di caricamento classi utilizzato per caricare IBM MQ classes for JMS. È quindi possibile per il programma di caricamento classi parent caricare un'exit del canale se non è possibile trovarlo in una delle ubicazioni precedenti. Tuttavia, quando si sta utilizzando IBM MQ classes for JMS in un ambiente come un server delle applicazioni JEE , non è possibile influenzare la scelta del programma di caricamento classi parent e, quindi, il programma di caricamento classi deve essere configurato impostando la proprietà di sistema Java **`com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath`** sul server delle applicazioni.

Se l'applicazione è in esecuzione con Java security manager abilitato, il file di configurazione della politica utilizzato dall'ambiente di runtime Java in cui l'applicazione è in esecuzione deve disporre delle autorizzazioni per caricare una classe di uscita del canale. Per informazioni su come eseguire questa operazione, fare riferimento a Esecuzione delle classi IBM MQ per le applicazioni JMS in Java Security Manager.

Le interfacce `MQSendExit`, `MQReceiveExite` `MQSecurityExit` fornite con versioni precedenti a IBM WebSphere MQ 7.0 sono ancora supportate. Se si utilizzano uscite di canale che implementano queste interfacce, `com.ibm.mq.jar` deve essere presente nel percorso di classe.

Per informazioni su come scrivere uscite di canale in C, consultare “Programmi di uscita canale per canali di messaggistica” a pagina 963. È necessario memorizzare i programmi di uscita del canale scritti in C o C++ nella directory mostrata in Tabella 43 a pagina 285.

Se l'applicazione utilizza una tabella di definizione del canale client (CCDT) per connettersi a un gestore code, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for JMS” a pagina 286.

Specifica dei dati utente da passare alle uscite del canale quando si utilizza IBM MQ classes for JMS
È possibile passare fino a 32 caratteri di dati utente a un'uscita canale quando viene richiamata.

La proprietà SENDEXITINIT di un oggetto MQConnectionFactory specifica i dati utente passati a ciascuna uscita di invio quando viene richiamato. Il valore della proprietà è una stringa che comprende uno o più elementi di dati utente separati da virgole. La posizione di ogni elemento di dati utente all'interno della stringa determina a quale uscita di invio, in una sequenza di uscite di invio, vengono passati i dati utente. Ad esempio, il primo elemento dei dati utente nella stringa viene passato alla prima uscita di invio in una sequenza di uscite di invio.

È possibile impostare la proprietà SENDEXITINIT utilizzando lo strumento di gestione IBM MQ JMS o IBM MQ Explorer. In alternativa, un'applicazione può impostare la proprietà richiamando il metodo setSendExitInit().

In modo simile, la proprietà RECEXITINIT di un oggetto ConnectionFactory specifica i dati utente passati a ciascuna uscita di ricezione e la proprietà SECEXITINIT specifica i dati dell'utente passati a una uscita di sicurezza. È possibile impostare queste proprietà utilizzando lo Strumento di amministrazione di IBM MQ JMS o IBM MQ Explorer. In alternativa, un'applicazione può impostare proprietà richiamando i metodi setReceiveExitInit() e setSecurityExitInit().

Notare le seguenti regole quando si specificano i dati utente passati alle uscite del canale:

- Se il numero di elementi dei dati utente in una stringa è superiore al numero di uscite in una sequenza, gli elementi in eccesso dei dati utente vengono ignorati.
- Se il numero di elementi dei dati utente in una stringa è inferiore al numero di uscite in una sequenza, ogni elemento non specificato dei dati utente viene impostato su una stringa vuota. Due virgole in successione all'interno di una stringa, o una virgola all'inizio di una stringa, indicano anche un elemento non specificato dei dati utente.

Se un'applicazione utilizza una tabella di definizione del canale client (CCDT) per connettersi a un gestore code, i dati utente specificati in una definizione del canale di connessione client vengono trasmessi alle uscite del canale quando vengono richiamate. Per ulteriori informazioni sull'utilizzo di una tabella di definizione del canale client, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for JMS” a pagina 286.

Utilizzo di una tabella di definizione di canale client con IBM MQ classes for JMS
Un'applicazione IBM MQ classes for JMS può utilizzare le definizione di canale di connessione client memorizzate in una CCDT (client channel definition table). Configurare un oggetto ConnectionFactory per utilizzare CCDT. Ci sono alcune limitazioni sul suo utilizzo.

Come alternativa alla creazione di una definizione di canale di connessione client impostando determinate proprietà di un oggetto ConnectionFactory , un'applicazione IBM MQ classes for JMS può utilizzare le definizioni di canale di connessione client memorizzate in una tabella di definizione di canale client. Queste definizioni vengono create dai comandi IBM MQ Script (MQSC) o IBM MQ Programmable Command Format (PCF). Quando l'applicazione crea un oggetto Connection, IBM MQ classes for JMS ricerca nella tabella di definizione del canale client una definizione di canale di connessione client adatta e utilizza la definizione di canale per avviare un canale MQI. Per ulteriori informazioni sulle tabelle di definizione dei canali client e su come crearne una, consultare Tabella di definizione dei canali client.

Per utilizzare una tabella di definizione del canale client, la proprietà CCDTURL di un oggetto ConnectionFactory deve essere impostata su un oggetto URL . IBM MQ classes for JMS non legge le informazioni relative alla CCDT dal file di configurazione IBM MQ MQI client , anche se alcuni altri valori vengono utilizzati da lì (consultare “Il file di configurazione IBM MQ classes for JMS/Jakarta Messaging” a pagina 100 per quale valore si applica). L'oggetto URL incapsula un URL (Uniform Resource Locator) (URL) che identifica il nome e l'ubicazione del file contenente la tabella di definizione del canale client e specifica come è possibile accedere al file. È possibile impostare la proprietà CCDTURL utilizzando lo strumento di amministrazione di IBM MQ JMS oppure un'applicazione può impostare la proprietà creando un oggetto URL e richiamando il metodo setCCDTURL() dell'oggetto ConnectionFactory .

Ad esempio, se il file ccdt1.tab contiene una tabella di definizione del canale client ed è memorizzato nello stesso sistema su cui è in esecuzione l'applicazione, l'applicazione può impostare la proprietà CCDTURL nel modo seguente:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Come un altro esempio, si supponga che il file ccdt2.tab contenga una tabella di definizione di canale client e che sia memorizzato su un sistema diverso da quello su cui è in esecuzione l'applicazione. Se è possibile accedere al file utilizzando il protocollo FTP, l'applicazione può impostare la proprietà CCDTURL nel modo seguente:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

Oltre a impostare la proprietà CCDTURL dell'oggetto ConnectionFactory , la proprietà QMANAGER dello stesso oggetto deve essere impostata su uno dei seguenti valori:

- Il nome di un gestore code
- Un asterisco (*) seguito dal nome di un gruppo di gestori code

Si tratta degli stessi valori che possono essere utilizzati per il parametro **QMgrName** in una chiamata MQCONN emessa da un'applicazione client che utilizza MQI (Message Queue Interface). Per ulteriori

informazioni sul significato di questi valori, consultare MQCONN. È possibile impostare la proprietà QMANAGER utilizzando lo strumento di amministrazione IBM MQ JMS o IBM MQ Explorer. In alternativa, un'applicazione può impostare la proprietà richiamando il metodo setQueueManager () dell'oggetto ConnectionFactory .

Se un'applicazione crea un oggetto Connection dall'oggetto ConnectionFactory , IBM MQ classes for JMS accede alla tabella di definizione del canale client identificata dalla proprietà CCDTURL, utilizza la proprietà QMANAGER per ricercare nella tabella una definizione di canale di connessione client adatta e quindi utilizza la definizione di canale per avviare un canale MQI su un gestore code.

Notare che le proprietà CCDTURL e CHANNEL di un oggetto ConnectionFactory non possono essere entrambe impostate quando l'applicazione richiama il metodo createConnection(). Se entrambe le proprietà sono impostate, il metodo genera un'eccezione. La proprietà CCDTURL o CHANNEL viene considerata impostata se il suo valore è diverso da null, una stringa vuota o una stringa contenente tutti i caratteri vuoti.

Quando IBM MQ classes for JMS trova una definizione di canale di connessione client adatta nella tabella di definizioni del canale client, utilizza solo le informazioni estratte dalla tabella per avviare un canale MQI. Tutte le proprietà relative al canale dell'oggetto ConnectionFactory vengono ignorate.

In particolare, notare i seguenti punti se si utilizza TLS:

- Un canale MQI utilizza TLS solo se la definizione di canale estratta dalla tabella di definizione di canale del client specifica il nome di un CipherSpec supportato da IBM MQ classes for JMS.
- Una tabella di definizione del canale client contiene anche informazioni sull'ubicazione dei server LDAP (Lightweight Directory Access Protocol) che contengono i CRL (Certificate Revocation List). IBM MQ classes for JMS utilizza solo tali informazioni per accedere a server LDAP che contengono CRL.
- Una tabella di definizione del canale client può contenere anche la posizione di un responder OCSP. IBM MQ classes for JMS non è in grado di utilizzare le informazioni OCSP contenute in un file della tabella di definizione di canale client. Tuttavia, è possibile configurare OCSP come descritto nella sezione OCSP (Online Certificate Status Protocol) nelle applicazioni client Java e JMS.

Per ulteriori informazioni sull'utilizzo di TLS con una tabella di definizione del canale del client, consultare Utilizzo del client transazionale esteso con i canali TLS.

Notare anche i seguenti punti se si utilizzano le uscite del canale:

- Un canale MQI utilizza solo le uscite del canale e i dati utente associati specificati dalla definizione del canale estratta dalla tabella di definizioni del canale client.
- Una definizione di canale estratta da una tabella di definizione di canale client può specificare uscite di canali scritte in Java. Ciò significa, ad esempio, che il parametro SCYEXIT nel comando DEFINE CHANNEL per creare una definizione di canale di connessione del client può specificare il nome di una classe che implementa l'interfaccia WMQSecurityExit . Allo stesso modo, il parametro SENDEXIT può specificare il nome di una classe che implementa l'interfaccia di WMQSendExit e il parametro RCVEXIT può specificare il nome di una classe che implementa l'interfaccia di WMQReceiveExit . Per ulteriori informazioni su come scrivere un'uscita del canale in Java, consultare“Scrittura di uscite di canale in Java per IBM MQ classes for JMS” a pagina 282.

  È supportato anche l'utilizzo di uscite di canale scritte in una lingua diversa da Java . Per informazioni su come specificare i parametri SCYEXIT, SENDEXIT e RCVEXIT nel comando DEFINE CHANNEL per le uscite canale scritte in un'altra lingua, vedere DEFINE CHANNEL.

Riconnessione client JMS automatica
Configurare il client JMS per riconnettersi automaticamente in seguito a un errore di rete, del gestore code o del server.

Di solito, se un'applicazione IBM MQ classes for JMS autonoma è connessa a un gestore code utilizzando il trasporto client e il gestore code diventa non disponibile per qualche motivo (a causa di un'interruzione di rete, di un malfunzionamento del gestore code o dell'arresto del gestore code, ad esempio), IBM MQ classes for JMS emetterà una JMSException la volta successiva che l'applicazione tenta di comunicare con il gestore code. L'applicazione deve rilevare JMSException e tentare di riconnettersi al gestore code. È possibile semplificare la progettazione dell'applicazione abilitando la riconnessione automatica del

client. Quando il gestore code diventa non disponibile, IBM MQ classes for JMS tenta di riconnettersi automaticamente al gestore code per conto dell'applicazione. Ciò significa che l'applicazione non deve contenere la logica per riconnettersi.

L'utilizzo di questa implementazione di riconnessione client automatica non è supportato nei server delle applicazioni Java Platform, Enterprise Edition . Per un'implementazione alternativa, consultare “Utilizzo della riconnessione client automatica in ambienti Java EE” a pagina 293 .

Utilizzo della riconnessione automatica del client JMS
Se un'applicazione IBM MQ classes for JMS autonoma utilizza un factory di connessione con la proprietà CONNECTIONNAMELIST o CCDTURL impostata, l'applicazione è idonea a utilizzare la riconnessione client automatica.

La riconnessione automatica del client può essere utilizzata per riconnettersi ai gestori code inclusi quelli che fanno parte di una configurazione HA (High Availability). Le configurazioni HA includono gestori code a più istanze, gestori code RDQM o gestori code HA su un dispositivo IBM MQ .

Il funzionamento della funzionalità di riconnessione client automatica fornita da IBM MQ classes for JMS dipende dalle proprietà riportate di seguito:

La proprietà JMS Connection Factory TRANSPORT (Nome breve TRAN)

TRANSPORT specifica il modo in cui le applicazioni che utilizzano il factory di connessione si collegano ad un gestore code. Questa proprietà deve essere impostata sul valore CLIENT per utilizzare la riconnessione automatica del client. La riconnessione automatica del client non è disponibile per le applicazioni che si connettono a un gestore code che utilizza un factory di connessione con la proprietà TRANSPORT impostata su BIND, DIRECT o DIRECTHTTP.

La proprietà JMS Connection Factory QMANAGER (nome breve QMGR)

La proprietà QMANAGER specifica il nome del gestore code a cui si connette il factory di connessione.

La proprietà del factory di connessione JMS CONNECTIONNAMELIST (nome breve CRHOSTS)

La proprietà CONNECTIONNAMELIST è un elenco separato da virgole, in cui ciascuna voce contiene informazioni sul nome host e sulla porta da utilizzare per connettersi al gestore code specificato dalla proprietà QMANAGER quando si utilizza il trasporto CLIENT. L'elenco ha il seguente formato: nome host (porta), nome host (porta).

La proprietà JMS Connection Factory CCDTURL (nome breve CCDT)

La proprietà CCDTURL punta alla tabella di definizione del canale client che IBM MQ classes for JMS utilizza quando si connette a un gestore code utilizzando una CCDT.

La proprietà JMS Connection Factory CLIENTRECONNECTOPTIONS (nome breve CROPT)

CLIENTRECONNECTOPTIONS controlla se IBM MQ classes for JMS tenterà di connettersi automaticamente a un gestore code per conto di una applicazione se un gestore code diventa disponibile.

L'attributo DefRecon nella stanza Channels del file di configurazione client

L'attributo DefRecon fornisce un'opzione di gestione per consentire a tutte le applicazioni di riconnettersi automaticamente o per disabilitare la riconnessione automatica per le applicazioni scritte per riconnettersi automaticamente.

La riconnessione automatica del client è disponibile solo quando un'applicazione si connette correttamente a un gestore code.

Quando un'applicazione si connette a un gestore code che utilizza il trasporto CLIENT, IBM MQ classes for JMS utilizza il valore della proprietà factory di connessione CLIENTRECONNECTOPTIONS per determinare se utilizzare la riconnessione client automatica, se il gestore code a cui è connessa l'applicazione diventa non disponibile. La tabella 1 mostra i valori possibili per la proprietà CLIENTRECONNECTOPTIONS e il comportamento di IBM MQ classes for JMS per ognuno di questi valori:

Tabella 44. Valori possibili della proprietà CLIENTRECCECTOPTIONS.

CLIENTRECONNECTOPTIONS	**Comportamento di IBM MQ classes for JMS**
ANY	Se è impostato CONNECTIONNAMELIST, utilizzare il valore della proprietà CONNECTIONNAMELIST per aprire una connessione a una combinazione di nome host e porta e connettersi a qualsiasi gestore code. Per utilizzare questa opzione di riconnessione client automatica, la proprietà QMANAGER deve essere impostata sul valore predefinito o su "*". Se CCDTURL è impostato, aprire la tabella di definizione del canale client specificata dalla proprietà CCDTURL, selezionare una voce nella tabella e utilizzare tale voce per avviare un canale di connessione client a un gestore code. Per utilizzare questa opzione di riconnessione automatica del client, la proprietà QMANAGER deve essere impostata su: • Un asterisco (*) • Un asterisco (*) seguito dal nome di un gruppo di gestori code • Una stringa vuota o una stringa che contiene tutti i caratteri vuoti
ASDEF	Utilizzare il valore di DefRecon per stabilire se è disponibile la riconnessione automatica del client.
DISABILITATO	Non eseguire alcuna riconnessione automatica del client e restituire una JMSException all'applicazione.
QMGR	Specifica che il client deve riconnettersi allo stesso gestore code. Questa opzione deve essere utilizzata per le soluzioni ad alta disponibilità, dove è richiesta la riconnessione ad un'altra istanza dello stesso gestore code. Se è impostato CONNECTIONNAMELIST, utilizzare il valore della proprietà CONNECTIONNAMELIST per aprire una connessione a una combinazione di nome host e porta e connettersi al gestore code specificato dalla proprietà QMANAGER. Se CCDTURL è impostato, aprire la tabella di definizione del canale client specificata dalla proprietà CCDTURL, individuare le voci nella tabella che corrispondono al nome del gestore code specificato dalla proprietà QMANAGER e utilizzare tali voci per avviare un canale di connessione client a tale gestore code.

Se CONNECTIONNAMELIST è impostato, quando si esegue la riconnessione automatica del client, IBM MQ classes for JMS utilizza le informazioni nella proprietà CONNNECTIONNAMELIST del factory di connessione per determinare a quale sistema riconnettersi.

IBM MQ classes for JMS tenta inizialmente di riconnettersi utilizzando il nome host e la porta specificati nella prima voce nell'CONNECTIONNAMELIST. Se viene stabilita una connessione, IBM MQ classes for JMS tenta di connettersi al gestore code che ha il nome specificato nella proprietà QMANAGER. Se è possibile stabilire una connessione al gestore code, IBM MQ classes for JMS riapre tutti gli oggetti IBM MQ che l'applicazione aveva aperto prima della riconnessione automatica del client e continua l'esecuzione come prima.

Se non è possibile stabilire una connessione al gestore code richiesto utilizzando la prima voce in CONNECTIONNAMELIST, IBM MQ classes for JMS tenta la seconda voce in CONNECTIONNAMELIST e così via.

Quando IBM MQ classes for JMS ha tentato tutte le voci nel CONNECTIONNAMELIST, attende un periodo di tempo prima di provare a riconnettersi di nuovo. Per eseguire il nuovo tentativo di riconnessione, il IBM MQ classes for JMS inizia con la prima voce nel CONNECTIONNAMELIST. Quindi, provano ogni voce in CONNECTIONNAMELIST a turno fino a quando non si verifica una riconnessione o viene raggiunta la fine di CONNECTIONNAMELIST, a quel punto IBM MQ classes for JMS attende per un periodo di tempo prima di riprovare.

Se CCDTURL è impostato, quando si esegue la riconnessione automatica del client, IBM MQ classes for JMS utilizza la tabella di definizione del canale client specificata nella proprietà CCDTURL per stabilire a quale sistema riconnettersi.

Il IBM MQ classes for JMS inizialmente analizza la tabella di definizione del canale client e trova una voce adatta che corrisponde al valore della proprietà QMANAGER. Quando viene trovata una voce, IBM MQ classes for JMS tenta di riconnettersi al gestore code richiesto utilizzando tale voce. Se è possibile stabilire una connessione al gestore code, IBM MQ classes for JMS riapre tutti gli oggetti IBM MQ che l'applicazione aveva aperto prima della riconnessione automatica del client e continua l'esecuzione come prima.

Se non è possibile stabilire una connessione al gestore code richiesto, IBM MQ classes for JMS cerca un'altra voce adatta nella tabella di definizione del canale client e tenta di utilizzarla e così via.

Quando IBM MQ classes for JMS ha tentato tutte le voci adatte nella tabella di definizione del canale client, attende un periodo di tempo prima di tentare nuovamente la connessione. Per eseguire il tentativo di riconnessione, IBM MQ classes for JMS analizza nuovamente la tabella di definizione del canale client e tenta la prima voce adatta. Quindi, tenteranno ogni voce adatta nella tabella di definizione del canale client, a turno, fino a quando non si verifica una riconnessione o fino a quando non viene tentata l'ultima voce adatta nella tabella di definizione del canale client, a quel punto IBM MQ classes for JMS attende un periodo di tempo prima di riprovare.

Se si utilizza CONNECTIONNAMELIST o CCDTURL, il processo di riconnessione automatica del client continua fino a quando IBM MQ classes for JMS non si riconnette correttamente al gestore code specificato dalla proprietà QMANAGER.

Per impostazione predefinita, i tentativi di riconnessione si verificano ai seguenti intervalli:

- Il primo tentativo viene effettuato dopo un ritardo iniziale di 1 secondo, più un elemento casuale fino a 250 millisecondi.
- Il secondo tentativo viene effettuato in 2 secondi, più un intervallo casuale fino a 500 millisecondi, dopo che il primo tentativo non è riuscito.
- Il terzo tentativo viene effettuato 4 secondi, più un intervallo casuale di un massimo di 1 secondo, dopo che il secondo tentativo non è riuscito.
- Il quarto tentativo viene effettuato in 8 secondi, più un intervallo casuale di un massimo di 2 secondi, dopo che il terzo tentativo non è riuscito.
- Il quinto tentativo viene effettuato 16 secondi, più un intervallo casuale fino a un massimo di 4 secondi, dopo che il quarto tentativo non è riuscito.
- Il sesto tentativo e tutti i tentativi successivi vengono effettuati in 25 secondi, più un intervallo casuale fino a un massimo di 6 secondi e 250 millisecondi dopo che il tentativo precedente non è riuscito.

I tentativi di riconnessione vengono ritardati da intervalli in parte fissi e in parte casuali. Ciò per evitare che tutte le applicazioni IBM MQ classes for JMS connesse a un gestore code non più disponibile si riconnettano contemporaneamente.

Se è necessario aumentare i valori predefiniti, per riflettere più accuratamente la quantità di tempo richiesta per il ripristino di un gestore code o per l'attivazione di un gestore code in standby, modificare l'attributo ReconDelay nella stanza del canale del file di configurazione client; per ulteriori informazioni, consultare Stanza CHANNELS del file di configurazione client.

Se un'applicazione IBM MQ classes for JMS continua a funzionare correttamente dopo essere stata riconnessa automaticamente dipende dalla sua progettazione. Leggere gli argomenti correlati per comprendere come progettare le applicazioni può utilizzare la funzionalità di riconnessione automatica.

Codici motivo che indicano che un gestore code non è più disponibile
I codici di origine errore indicano che un gestore code non è più disponibile o non può essere raggiunto quando si tenta la riconnessione IBM MQ classes for JMS automatica.

“Riconnessione client JMS automatica” a pagina 287 fornisce una panoramica di JMSExceptions e di come le applicazioni possono essere riavviate automaticamente, e le informazioni in “Utilizzo della riconnessione automatica del client JMS” a pagina 288 illustrano i requisiti per la riconnessione automatica del client.

Le seguenti informazioni elencano i codici di errore IBM MQ che l'applicazione deve controllare:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ATTIVO

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NON_DISPONIBILE

La maggior parte delle JMSExceptions restituite alle applicazioni enterprise contengono una MQException collegata che contiene il codice motivo. Per implementare la logica dei tentativi per i codici di errore nell'elenco precedente, le proprie applicazioni enterprise devono controllare questa eccezione collegata utilizzando un codice simile al seguente esempio:

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

Concetti correlati

Classi IBM MQ per JMS

Utilizzo della riconnessione client automatica in ambienti Java SE e Java EE

È possibile utilizzare la riconnessione client automatica IBM MQ per facilitare varie soluzioni HA (high availability) e DR (disaster recovery) in un ambiente Java SE e Java EE .

Diverse soluzioni HA e DR sono disponibili su diverse piattaforme:

- Multi I gestori code a più istanze sono istanze dello stesso gestore code configurato su server differenti (consultare Gestori code a più istanze). Un'istanza del gestore code è definita come istanza attiva e un'istanza è definita come istanza in standby. Se l'istanza attiva ha esito negativo, il gestore code a più istanze viene riavviato automaticamente sul server di standby.

  Entrambi i gestori code attivi e in standby hanno lo stesso identificativo del gestore code (QMID). Le IBM MQ applicazioni client che si collegano a un gestore code a più istanze possono essere configurate per riconnettersi automaticamente a una istanza in standby di un gestore code utilizzando la riconnessione client automatica.
- Linux RDQM (replicated data queue manager) è una soluzione alta disponibilità disponibile su piattaforme Linux (consultare RDQM high availability). Una configurazione RDQM è composta da tre server configurati in un gruppo HA (High Availability), ciascuno con una istanza del gestore code. Un'istanza è il gestore code in esecuzione, che replica in modo sincrono i dati nelle altre due istanze. Se il server su cui è in esecuzione questo gestore code ha esito negativo, viene avviata un'altra istanza del gestore code con cui devono operare i dati correnti. Le tre istanze del gestore code condividono un indirizzo IP mobile, pertanto i clienti devono essere configurati solo con un singolo indirizzo IP. Le applicazioni client che si connettono a un gestore code RDQM possono essere configurate per riconnettersi automaticamente a un'istanza in standby di un gestore code utilizzando la riconnessione client automatica.
- MQ Appliance Una soluzione HA può essere fornita anche da una coppia di dispositivi IBM MQ (vedi Alta disponibilità e Disaster Recovery nella documentazione IBM MQ Appliance). Un gestore code HA viene eseguito su uno dei dispositivi, mentre i dati vengono replicati in modo sincrono sull'istanza in standby del gestore code sull'altro dispositivo. Se il dispositivo primario ha esito negativo, il gestore code viene avviato automaticamente e viene eseguito sull'altro dispositivo. Le due istanze del gestore code possono essere configurate per condividere un indirizzo IP mobile, quindi i client devono essere configurati solo con un singolo indirizzo IP. Le applicazioni client che si collegano a un gestore code HA su IBM MQ Appliance possono essere configurate per riconnettersi automaticamente all'istanza in standby di un gestore code utilizzando la riconnessione client automatica.

Nota: All'interno degli ambienti Java EE , come WebSphere Application Server, la riconnessione automatica del client con le specifiche di attivazione utilizzando la funzionalità fornita da IBM MQ classes for JMS non è supportata. L'adattatore di risorse IBM MQ fornisce il proprio meccanismo per la riconnessione delle specifiche di attivazione se il gestore code a cui si stava connettendo la specifica di attivazione diventa non disponibile. Per ulteriori informazioni, fare riferimento a “Supporto per la riconnessione automatica del client in ambienti Java EE” a pagina 294.

Concetti correlati

Gestori code a più istanze

Riconnessione automatica del client

Riferimenti correlati

Alta disponibilità rdqm

Utilizzo della riconnessione client automatica in ambienti Java SE

Le applicazioni che utilizzano gli ambienti IBM MQ classes for JMS in esecuzione in Java SE possono utilizzare la funzionalità di riconnessione client automatica tramite la proprietà factory di connessione **CLIENTRECONNECTOPTIONS**.

La proprietà della factory di connessione **CLIENTRECONNECTOPTIONS** utilizza due proprietà aggiuntive della factory di connessione, **CONNECTIONNAMELIST** e **CCDTURL**, per determinare come connettersi al server su cui è in esecuzione il gestore code.

CONNECTIONNAMELIST proprietà

La proprietà **CONNECTIONNAMELIST** è un elenco separato da virgole che contiene le informazioni sul nome host e sulla porta da utilizzare per connettersi a un gestore code in modalità client. Questa proprietà viene utilizzata con i valori **QMANAGER** e **CHANNEL** . Quando un'applicazione utilizza la proprietà **CONNECTIONNAMELIST** per creare una connessione client, IBM MQ classes for JMS tenta di connettersi a ogni host in ordine di elenco. Se il primo host del gestore code non è disponibile, IBM MQ classes for JMS tenta di connettersi all'host successivo nell'elenco. Se viene raggiunta la fine dell'elenco dei nomi di connessione senza creare una connessione, il IBM MQ classes for JMS genera il codice di errore `MQRC_QMGR_NOT_AVAILABLE` IBM MQ .

Se il gestore code a cui è connessa l'applicazione ha esito negativo, tutte le applicazioni che hanno utilizzato un **CONNECTIONNAMELIST** per connettersi a tale gestore code ricevono un'eccezione che indica che il gestore code non è disponibile. L'applicazione deve rilevare l'eccezione e cancellare tutte le risorse che stava utilizzando. Per creare una connessione, l'applicazione deve utilizzare la factory di connessione. Il factory di connessione tenta di collegarsi nuovamente a ciascun host in ordine di elenco, il gestore code che ha avuto esito negativo non è ora disponibile. Il factory di connessione tenta di connettersi a un altro host nell'elenco.

CCDTURL proprietà

La proprietà **CCDTURL** contiene un URL (Uniform Resource Locator) (URL) che punta a una CCDT (Client Channel Definition Table), questa proprietà viene utilizzata con la proprietà **QMANAGER** . CDT contiene un elenco di canali client utilizzati per connettersi a un gestore code definito su un sistema IBM MQ . Per informazioni su come i CCDT vengono utilizzati da IBM MQ classes for JMS, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for JMS” a pagina 286.

Utilizzo della propriet ... CLIENTRECONNECTOPTIONS per abilitare la riconnessione automatica del client all'interno di IBM MQ classes for JMS

La proprietà **CLIENTRECONNECTOPTIONS** viene utilizzata per consentire la riconnessione automatica del client in IBM MQ classes for JMS. I valori possibili per questa proprietà sono i seguenti:

ASDEF
Il funzionamento della riconnessione automatica del client è definito dal valore predefinito specificato nella stanza del canale del file di configurazione del client IBM MQ (`mqclient.ini`).

Disabilitato
La riconnessione automatica del client è disabilitata.

QMGR
Il IBM MQ classes for JMS tenta di connettersi a un gestore code con lo stesso identificativo del gestore code a cui era connesso, utilizzando una delle seguenti opzioni:

- La proprietà **CONNECTIONNAMELIST** e il canale definito nella proprietà **CHANNEL** .
- Il CCDT definito nella proprietà **CCDTURL** .

ANY
Il IBM MQ classes for JMS tenta di riconnettersi a un gestore code con lo stesso nome utilizzando la proprietà **CONNECTIONNAMELIST** o **CCDTURL**.

Informazioni correlate

Stanza CHANNELS del file di configurazione client

Utilizzo della riconnessione client automatica in ambienti Java EE
L'adattatore di risorse IBM MQ , che può essere distribuito in ambienti Java EE (Java Platform, Enterprise Edition) e il fornitore di messaggi WebSphere Application Server IBM MQ utilizzano IBM MQ classes for

JMS per comunicare con gestori code IBM MQ . L'adattatore di risorse IBM MQ e il fornitore di messaggi WebSphere Application Server IBM MQ forniscono una serie di meccanismi per consentire alle specifiche di attivazione, alle porte del listener WebSphere Application Server e alle applicazioni in esecuzione nei contenitori client di riconnettersi automaticamente a un gestore code. Gli EJB (Enterprise JavaBeans) e le applicazioni basate su Web devono implementare la propria logica di riconnessione.

Nota: La riconnessione automatica del client con specifiche di attivazione utilizzando la funzionalità fornita da IBM MQ classes for JMS non è supportata (consultare “Riconnessione client JMS automatica” a pagina 287). L'adattatore di risorse IBM MQ fornisce il proprio meccanismo per la riconnessione delle specifiche di attivazione se il gestore code a cui si stava connettendo la specifica di attivazione diventa non disponibile.

Il meccanismo fornito dall'adattatore di risorse è controllato da:

- La IBM MQ proprietà dell'adattatore risorse **reconnectionRetryCount**.
- La IBM MQ proprietà dell'adattatore risorse **reconnectionRetryInterval**.
- La proprietà della specifica di attivazione **connectionNameList**.

Per ulteriori informazioni su queste proprietà, consultare “Configurazione per le proprietà oggetto ResourceAdapter” a pagina 451.

L'utilizzo della riconnessione client automatica all'interno del metodo `onMessage()` di un'applicazione MDB (message - driven bean) o qualsiasi altra applicazione in esecuzione nell'ambiente Java Platform, Enterprise Edition non è supportato. L'applicazione deve implementare la propria logica di riconnessione se il gestore code a cui si stava collegando diventa non disponibile. Per ulteriori informazioni, consultare “Implementazione della logica di riconnessione in un'applicazione Java EE” a pagina 301.

Supporto per la riconnessione automatica del client in ambienti Java EE
In ambienti Java EE , come WebSphere Application Server, l'adattatore di risorse IBM MQ e il provider di messaggistica WebSphere Application Server IBM MQ forniscono una serie di meccanismi che consentono alle applicazioni di riconnettersi automaticamente a un gestore code. Tuttavia, in alcuni casi, a questo supporto si applicano restrizioni.

L'adattatore di risorse IBM MQ che può essere distribuito in ambienti Java EE e il provider di messaggistica WebSphere Application Server IBM MQ , utilizza IBM MQ classes for JMS per comunicare con i gestori code IBM MQ .

La seguente tabella riepiloga il supporto fornito dall'adattatore di risorse IBM MQ e dal fornitore di messaggi WebSphere Application Server IBM MQ per la riconnessione client automatica.

Tabella 45. Riepilogo del supporto per le opzioni di riconnessione client automatica in ambienti Java EE

Opzioni per la riconnessione automatica	**proprietà CONNECTIONNAMELIST**	**proprietà CCDTURL**	**proprietà CLIENTRECONNECTOPTIONS**	**Approccio alternativo alla riconnessione automatica del client**
Specifiche di attivazione	Supportato con limitazioni	Supportato con limitazioni	Non supportato	Le specifiche di attivazione e dell'ambiente Java EE forniscono il proprio meccanismo di riconnessione
WebSphere Application Server Porte listener	Supportato con limitazioni	Supportato con limitazioni	Non supportato	WebSphere Application Server fornisce il proprio meccanismo di riconnessione

Tabella 45. Riepilogo del supporto per le opzioni di riconnessione client automatica in ambienti Java EE (Continua)

Opzioni per la riconnessione automatica	**proprietà CONNECTIONNAMELIST**	**proprietà CCDTURL**	**proprietà CLIENTRECONNECTOPTIONS**	**Approccio alternativo alla riconnessione automatica del client**
Enterprise JavaBeans e applicazioni basate sul web	Supportato con limitazioni	Supportato con limitazioni	Non supportato	L'applicazione deve implementare la propria logica di riconnessione
Applicazioni in esecuzione all'interno dei contenitori client	Supportato	Supportato	Supportato	Non applicabile

Le applicazioni MDB (Message - Driven Bean) installate in ambiente Java EE , come IBM MQ classes for JMS, possono utilizzare le specifiche di attivazione per elaborare i messaggi su un sistema IBM MQ . Le specifiche di attivazione vengono utilizzate per rilevare i messaggi che arrivano su un sistema IBM MQ e consegnarli ai bean basati sui messaggi per l'elaborazione. I bean basati sui messaggi possono anche stabilire ulteriori connessioni ai sistemi IBM MQ dall'interno del loro metodo **onMessage()** . Per ulteriori informazioni su come queste connessioni possono utilizzare la riconnessione client automatica, consultare Enterprise JavaBeans e le applicazioni basate sul web.

Specifiche di attivazione
Per le specifiche di attivazione, le proprietà **CONNECTIONNAMELIST** e **CCDTURL** sono supportate con limitazioni e la proprietà **CLIENTRECONNECTOPTIONS** non è supportata.

Le applicazioni MDB (Message - Driven Bean) installate in un ambiente Java EE , come WebSphere Application Server, possono utilizzare specifiche di attivazione per elaborare i messaggi su un sistema IBM MQ .

Le specifiche di attivazione vengono utilizzate per rilevare i messaggi in arrivo su un sistema IBM MQ e quindi consegnarli agli MDB per l'elaborazione. Questa sezione descrive il modo in cui la specifica di attivazione controlla il sistema IBM MQ .

Gli MDB possono anche effettuare connessioni aggiuntive ai sistemi IBM MQ dall'interno del rispettivo metodo `onMessage()` .

I dettagli su come queste connessioni possono utilizzare la riconnessione automatica del client sono disponibili in “Enterprise JavaBeans e applicazioni basate sul web” a pagina 299.

CONNECTIONNAMELIST proprietà

All'avvio, la specifica di attivazione tenta di connettersi al gestore code utilizzando:

- Uno specificato nella proprietà **QMANAGER**
- Canale menzionato nella proprietà **CHANNEL**
- Informazioni su nome host e porta dalla prima voce in **CONNECTIONNAMELIST**

Se la specificazione di attivazione non è in grado di collegarsi al gestore code utilizzando la prima voce nell'elenco, la specifica di attivazione passa alla seconda voce e così via, fino a quando non viene effettuata una connessione al gestore code o fino a quando non viene raggiunta la fine dell'elenco.

Se la specifica di attivazione non è in grado di connettersi al gestore code specificato, utilizzando una delle voci in **CONNECTIONNAMELIST**, la specifica di attivazione viene arrestata e deve essere riavviata.

Una volta che la specifica di attivazione è in esecuzione, la specifica di attivazione richiama i messaggi dal sistema IBM MQ e consegna i messaggi a un MDB per l'elaborazione.

Se il gestore code non riesce durante l'elaborazione di un messaggio, l'ambiente Java EE rileva l'errore e tenta di riconnettere la specifica di attivazione.

La specifica di attivazione utilizza le informazioni nella proprietà **CONNECTIONNAMELIST** come prima, quando la specifica di attivazione esegue i tentativi di riconnessione.

Se la specifica di attivazione tenta tutte le voci in **CONNECTIONNAMELIST** e non riesce ancora a connettersi al gestore code, la specifica di attivazione attende il periodo di tempo specificato dalla IBM MQ proprietà dell'adattatore di risorse **reconnectionRetryInterval** prima di riprovare.

La IBM MQ proprietà dell'adattatore di risorse **reconnectionRetryCount** definisce il numero di tentativi di riconnessione consecutivi effettuati prima dell'arresto di una specifica di attivazione e richiede un riavvio manuale

Una volta che la specifica di attivazione si è riconnessa a un sistema IBM MQ , l'ambiente di Java EE esegue qualsiasi ripulitura transazionale richiesta e riprende la consegna dei messaggi agli MDB per l'elaborazione.

Affinché il cleanup transazionale funzioni correttamente, l'ambiente Java EE deve essere in grado di accedere ai log per il gestore code non riuscito.

Se le specifiche di attivazione vengono utilizzate con gli MDB transazionali che partecipano alle transazioni XA e si collegano a un gestore code a più istanze, **CONNECTIONNAMELIST** deve contenere una voce sia per l'istanza del gestore code attivo che per quella in standby.

Ciò significa che l'ambiente Java EE può accedere ai log del gestore code se l'ambiente deve eseguire il ripristino della transazione, indipendentemente dal gestore code a cui si riconnette l'ambiente in seguito a un errore.

Se gli MDB transazionali vengono utilizzati con i gestori code autonomi, la proprietà **CONNECTIONNAMELIST** deve contenere una singola voce, per garantire che la specifica di attivazione si riconnetta sempre allo stesso gestore code in esecuzione sullo stesso sistema in seguito a un errore.

CCDTURL proprietà

All'avvio, la specifica di attivazione tenta di collegarsi al gestore code specificato nella proprietà **QMANAGER** utilizzando la prima voce nella tabella di definizione del canale client (CCDT).

Se la specifica di attivazione non è in grado di connettersi al gestore code utilizzando la prima voce nella tabella, la specifica di attivazione passa alla seconda voce e così via, fino a quando non viene effettuata una connessione al gestore code o viene raggiunta la fine della tabella.

Se la specifica di attivazione non è in grado di connettersi al gestore code specificato, utilizzando una delle voci in CCDT, la specifica di attivazione si arresta e deve essere riavviata.

Una volta che la specifica di attivazione è in esecuzione, la specifica di attivazione richiama i messaggi dal sistema IBM MQ e consegna i messaggi a un MDB per l'elaborazione.

Se il gestore code non riesce durante l'elaborazione di un messaggio, l'ambiente Java EE rileva l'errore e tenta di riconnettere la specifica di attivazione.

La specifica di attivazione utilizza le informazioni nella proprietà CCDT come in precedenza, quando la specifica di attivazione esegue i tentativi di riconnessione.

Se la specifica di attivazione prova tutte le voci in CCDT e non è ancora in grado di connettersi al gestore code, la specifica di attivazione attende il periodo di tempo specificato dalla IBM MQ proprietà dell'adattatore di risorse **reconnectionRetryInterval** prima di riprovare.

La IBM MQ proprietà dell'adattatore di risorse **reconnectionRetryCount** definisce il numero di tentativi di riconnessione consecutivi effettuati prima dell'arresto di una specifica di attivazione e richiede un riavvio manuale

Una volta che la specifica di attivazione si è riconnessa a un sistema IBM MQ , l'ambiente di Java EE esegue qualsiasi ripulitura transazionale richiesta e riprende la consegna dei messaggi agli MDB per l'elaborazione.

Affinché il cleanup transazionale funzioni correttamente, l'ambiente Java EE deve essere in grado di accedere ai log per il gestore code non riuscito.

Se le specifiche di attivazione vengono utilizzate con gli MDB transazionali che partecipano alle transazioni XA e si collegano a un gestore code a più istanze, CCDT deve contenere una voce sia per l'istanza del gestore code attivo che per quella in standby.

Ciò significa che l'ambiente Java EE può accedere ai log del gestore code se l'ambiente deve eseguire il ripristino della transazione, indipendentemente dal gestore code a cui si riconnette l'ambiente in seguito a un errore.

Se gli MDB transazionali vengono utilizzati con gestori code autonomi, la CCDT deve contenere una singola voce, per garantire che la specifica di attivazione si riconnetta sempre allo stesso gestore code in esecuzione sullo stesso sistema in seguito a un errore.

Assicurarsi di aver impostato il valore predefinito di *PREFERRED* per la proprietà **AFFINITY** nei CCDT, utilizzati con specifiche di attivazione, in modo che le connessioni vengano effettuate allo stesso gestore code attivo.

CLIENTRECONNECTOPTIONS proprietà

Le specifiche di attivazione forniscono la propria funzionalità di riconnessione. La funzionalità fornita consente alle specifiche di riconnettersi automaticamente a un sistema IBM MQ se il gestore code a cui erano connessi ha esito negativo.

Per questo motivo, la funzionalità di riconnessione automatica del client fornita da IBM MQ classes for JMS non è supportata.

È necessario impostare la proprietà **CLIENTRECONNECTOPTIONS** su *DISABLED* per tutte le specifiche di attivazione utilizzate in Java EE.

WebSphere Application Server Porte listener
Le applicazioni MDB (Message - Driven Bean) installate in WebSphere Application Server possono anche utilizzare le porte listener per elaborare i messaggi su un sistema IBM MQ .

Le porte listener vengono utilizzate per rilevare i messaggi in arrivo su un sistema IBM MQ e quindi consegnarli agli MDB per l'elaborazione. Questo argomento spiega il modo in cui la porta listener monitora il sistema IBM MQ .

Gli MDB possono anche effettuare connessioni aggiuntive ai sistemi IBM MQ dall'interno del rispettivo metodo `onMessage()` .

Consultare “Enterprise JavaBeans e applicazioni basate sul web” a pagina 299 per ulteriori informazioni su come queste connessioni possono utilizzare la riconnessione automatica del client

Per le porte listener WebSphere Application Server :

- **CONNECTIONNAMELIST** e **CCDTURL** sono supportati con limitazioni
- **CLIENTRECONNECTOPTIONS** non è supportato

CONNECTIONNAMELIST proprietà

Le porte listener utilizzano i lotti di connessione JMS durante la connessione a IBM MQ, quindi sono soggette alle implicazioni dell'utilizzo dei lotti di connessione. Consultare “Specifiche di attivazione” a pagina 295 per ulteriori informazioni.

Se non ci sono connessioni libere e il numero massimo di connessioni non è stato ancora creato da questo factory di connessione, la proprietà **CONNECTIONNAMELIST** viene utilizzata per provare a creare una nuova connessione a IBM MQ.

Se tutti i sistemi IBM MQ in **CONNECTIONNAMELIST** non sono accessibili, la porta del listener si arresta.

La porta listener attende quindi il periodo di tempo specificato dalla proprietà personalizzata del servizio listener dei messaggi **RECOVERY.RETRY.INTERVAL** e tenta di riconnettersi nuovamente.

Questo tentativo di riconnessione verifica se sono presenti connessioni libere nel lotto di connessioni, nel caso in cui ne sia stato restituito uno tra i tentativi di connessione. Se uno non è disponibile, la porta listener utilizza **CONNECTIONNAMELIST** come prima.

Una volta che la porta del listener si è riconnessa ad un sistema IBM MQ , l'ambiente Java EE esegue qualsiasi ripulitura transazionale richiesta e quindi riprende la consegna dei messaggi agli MDB per l'elaborazione.

Affinché il cleanup transazionale funzioni correttamente, l'ambiente Java EE deve essere in grado di accedere ai log per il gestore code non riuscito.

Se le porte listener vengono utilizzate con gli MDB transazionali che partecipano alle transazioni XA e si collegano a un **gestore code a più istanze**, **CONNECTIONNAMELIST** deve contenere una voce sia per l'istanza del gestore code attivo che per quella in standby.

Ciò significa che l'ambiente Java EE può accedere ai log del gestore code se l'ambiente deve eseguire il ripristino della transazione, indipendentemente dal gestore code a cui si riconnette l'ambiente in seguito a un errore.

Se gli MDB transazionali vengono utilizzati con i gestori code autonomi, la proprietà **CONNECTIONNAMELIST** deve contenere una singola voce, per garantire che la specifica di attivazione si riconnetta sempre allo stesso gestore code in esecuzione sullo stesso sistema in seguito a un errore.

CCDTURL proprietà

Quando si avvia, la porta listener tenta di connettersi al gestore code specificato nella proprietà **QMANAGER** utilizzando la prima voce in CCDT.

Se la porta del listener non è in grado di connettersi al gestore code utilizzando la prima voce nella tabella, la porta del listener passa alla seconda voce e così via, fino a quando non viene effettuata una connessione al gestore code o non viene raggiunta la fine della tabella.

Se la porta listener non è in grado di connettersi al gestore code specificato utilizzando una delle voci in CCDT, la porta listener viene arrestata.

La porta listener attende quindi il periodo di tempo specificato dalla proprietà personalizzata del servizio listener dei messaggi **RECOVERY.RETRY.INTERVAL** e tenta di riconnettersi nuovamente.

Questo tentativo di riconnessione funziona attraverso tutte le voci in CCDT come prima.

Una volta che la porta listener è in esecuzione, riceve i messaggi dal sistema IBM MQ e li consegna a un MDB per l'elaborazione.

Se il gestore code ha esito negativo durante l'elaborazione di un messaggio, l'ambiente Java EE rileva l'errore e tenta di riconnettere la porta del listener. La porta listener utilizza le informazioni in CCDT quando esegue i tentativi di riconnessione.

Se la porta listener tenta tutte le voci in CCDT e non è ancora in grado di connettersi al gestore code, la porta attende il periodo di tempo specificato dalla proprietà **RECOVERY.RETRY.INTERVAL** prima di riprovare.

La proprietà del servizio listener dei messaggi **MAX.RECOVERY.RETRIES** definisce il numero di tentativi di riconnessione consecutivi effettuati prima che una porta listener si arresti e richieda un riavvio manuale.

Una volta che la porta del listener si è riconnessa ad un sistema IBM MQ , l'ambiente Java EE esegue qualsiasi ripulitura transazionale richiesta e quindi riprende la consegna dei messaggi agli MDB per l'elaborazione.

Affinché il cleanup transazionale funzioni correttamente, l'ambiente Java EE deve essere in grado di accedere ai log per il gestore code non riuscito.

Se le porte listener vengono utilizzate con gli MDB transazionali che partecipano alle transazioni XA e si collegano a un gestore code a più istanze, CCDT deve contenere una voce sia per l'istanza del gestore code attivo che per quella in standby.

Ciò significa che l'ambiente Java EE può accedere ai log del gestore code se l'ambiente deve eseguire il ripristino della transazione, indipendentemente dal gestore code a cui si riconnette l'ambiente in seguito a un errore.

Se gli MDB transazionali vengono utilizzati con gestori code autonomi, CCDT deve contenere una singola voce, per garantire che la porta listener si riconnetta sempre allo stesso gestore code in esecuzione sullo stesso sistema in seguito a un errore.

Assicurarsi di aver impostato il valore predefinito di *PREFERRED* per la proprietà **AFFINITY** sui CCDT, utilizzati con le porte listener, in modo che le connessioni vengano effettuate sullo stesso gestore code attivo.

CLIENTRECONNECTOPTIONS proprietà

Le porte listener forniscono la propria funzionalità di riconnessione. La funzionalità fornita consente alle porte listener di riconnettersi automaticamente ad un sistema IBM MQ se il gestore code a cui erano connesse ha esito negativo.

Per questo motivo, la funzionalità di riconnessione automatica del client fornita da IBM MQ classes for JMS non è supportata.

È necessario impostare la proprietà **CLIENTRECONNECTOPTIONS** su *DISABLED* per tutte le porte listener utilizzate in Java EE.

Enterprise JavaBeans e applicazioni basate sul web
Le applicazioni EJB (Enterprise JavaBean) e le applicazioni eseguite all'interno del contenitore Web, come ad esempio i servlet, utilizzano una factory di connessione JMS per creare una connessione a un gestore code IBM MQ .

Le seguenti limitazioni si applicano agli EJB e alle applicazioni basate sul Web:

- **CONNECTIONNAMELIST** e **CCDTURL** sono supportati con limitazioni
- **CLIENTRECONNECTOPTIONS** non è supportato

CONNECTIONNAMELIST proprietà

Se l'ambiente Java EE fornisce un pool di connessione per le connessioni JMS, consultare “Utilizzo di CONNECTIONNAMELIST o CCDT in un lotto di connessioni” a pagina 300 per informazioni sul modo in cui ciò influisce sul comportamento della proprietà **CONNECTIONNAMELIST** .

Se l'ambiente Java EE non fornisce un pool di connessioni JMS . l'applicazione utilizza la proprietà **CONNECTIONNAMELIST** nello stesso modo delle applicazioni Java SE .

Se le applicazioni vengono utilizzate con gli MDB transazionali che partecipano alle transazioni XA e si collegano a un gestore code a più istanze, **CONNECTIONNAMELIST** deve contenere una voce sia per l'istanza del gestore code attivo che per quella in standby.

Ciò significa che l'ambiente Java EE può accedere ai log del gestore code se l'ambiente deve eseguire il ripristino della transazione, indipendentemente dal gestore code a cui si riconnette l'ambiente in seguito a un errore.

Se le applicazioni vengono utilizzate con gestori code autonomi, la proprietà **CONNECTIONNAMELIST** deve contenere una singola voce, per garantire che l'applicazione si riconnetta sempre allo stesso gestore code, in esecuzione sullo stesso sistema, in seguito a un errore.

CCDTURL proprietà

Se l'ambiente Java EE fornisce un pool di connessione per le connessioni JMS , consultare “Utilizzo di CONNECTIONNAMELIST o CCDT in un lotto di connessioni” a pagina 300 per informazioni sul modo in cui ciò influisce sul comportamento della proprietà **CCDTURL** .

Se l'ambiente Java EE non fornisce un pool di connessioni JMS . l'applicazione utilizza la proprietà **CCDTURL** nello stesso modo delle applicazioni Java SE .

Se le applicazioni vengono utilizzate con gli MDB transazionali che partecipano alle transazioni XA e si collegano a un gestore code a più istanze, CCDT deve contenere una voce sia per l'istanza del gestore code attivo che per quella in standby.

Ciò significa che l'ambiente Java EE può accedere ai log del gestore code se l'ambiente deve eseguire il ripristino della transazione, indipendentemente dal gestore code a cui si riconnette l'ambiente in seguito a un errore.

Se le applicazioni vengono utilizzate con gestori code autonomi, la CCDT deve contenere una singola voce, per garantire che la specifica di attivazione si riconnetta sempre allo stesso gestore code in esecuzione sullo stesso sistema in seguito a un errore.

CLIENTRECONNECTOPTIONS proprietà

È necessario impostare la proprietà **CLIENTRECONNECTOPTIONS** su *DISABLED* per tutte le factory di connessione JMS utilizzate dagli EJB o dalle applicazioni che vengono eseguite nel contenitore Web.

Le applicazioni che richiedono la riconnessione automatica a un nuovo gestore code, se il gestore code che stanno utilizzando ha esito negativo, devono implementare la propria logica di riconnessione. Per ulteriori informazioni, consultare “Implementazione della logica di riconnessione in un'applicazione Java EE” a pagina 301.

Scenari: WebSphere Application Server con IBM MQ
Scenari: WebSphere Application Server Liberty profile con IBM MQ

Applicazioni in esecuzione all'interno dei contenitori client
Alcuni ambienti Java EE , come WebSphere Application Server, forniscono un contenitore client che è possibile utilizzare per eseguire applicazioni Java SE .

Le applicazioni in esecuzione in questi ambienti utilizzano una factory di connessione JMS per connettersi a un gestore code IBM MQ .

Per le applicazioni in esecuzione all'interno dei contenitori client:

- **CONNECTIONNAMELIST** e **CCDTURL** sono completamente supportati
- **CLIENTRECONNECTOPTIONS** è completamente supportato

CONNECTIONNAMELIST proprietà

Se l'ambiente Java EE fornisce un pool di connessione per le connessioni JMS, consultare “Utilizzo di CONNECTIONNAMELIST o CCDT in un lotto di connessioni” a pagina 300 per informazioni sul modo in cui ciò influisce sul comportamento della proprietà **CONNECTIONNAMELIST** .

Se l'ambiente Java EE non fornisce un pool di connessioni JMS . l'applicazione utilizza la proprietà **CONNECTIONNAMELIST** nello stesso modo delle applicazioni Java SE .

CCDTURL proprietà

Se l'ambiente Java EE fornisce un pool di connessione per le connessioni JMS , consultare “Utilizzo di CONNECTIONNAMELIST o CCDT in un lotto di connessioni” a pagina 300 per informazioni sul modo in cui ciò influisce sul comportamento della proprietà **CCDTURL** .

Se l'ambiente Java EE non fornisce un pool di connessioni JMS . l'applicazione utilizza la proprietà **CCDTURL** nello stesso modo delle applicazioni Java SE .

Utilizzo di CONNECTIONNAMELIST o CCDT in un lotto di connessioni
Alcuni ambienti Java EE , ad esempio WebSphere Application Server, forniscono un pool di connessione JMS . contenitore che è possibile utilizzare per eseguire le applicazioni Java SE .

Le applicazioni che creano una connessione utilizzando una produzione connessioni definita nell'ambiente Java EE ottengono una connessione libera esistente dal pool di connessioni per questa

produzione connessioni o una nuova connessione se non è presente una connessione adatta nel pool di connessioni.

Ciò può avere implicazioni se la factory di connessione è stata configurata con la proprietà **CONNECTIONNAMELIST** o **CCDTURL** definita.

La prima volta che il factory di connessione viene utilizzato per creare una connessione, l'ambiente Java EE utilizza **CONNECTIONNAMELIST**. o **CCDTURL** per creare una nuova connessione al sistema IBM MQ . Quando questa connessione non è più richiesta, viene restituita al pool di connessioni in cui la connessione diventa disponibile per il riutilizzo.

Se un altro elemento crea una connessione dalla factory di connessione, l'ambiente Java EE restituisce la connessione dal pool di connessioni, invece di utilizzare le proprietà **CONNECTIONNAMELIST** o **CCDTURL** per creare una nuova connessione.

Se una connessione viene utilizzata quando un'istanza del gestore code ha esito negativo, la connessione viene eliminata. Tuttavia, il contenuto del pool di connessione potrebbe non essere, il che significa che il pool può potenzialmente contenere ancora connessioni a un gestore code che non è più in esecuzione.

In questa situazione, la volta successiva che viene effettuata una richiesta di creazione di una connessione dalla factory di connessione, viene restituita una connessione al gestore code non riuscito. Qualsiasi tentativo di utilizzare questa connessione ha esito negativo, poiché il gestore code non è più in esecuzione, causando l'eliminazione della connessione.

Solo quando il lotto connessioni è vuoto, l'ambiente Java EE utilizzerà le proprietà **CONNECTIONNAMELIST** o **CCDTURL** per creare una nuova connessione a IBM MQ.

A causa del modo in cui **CONNECTIONNAMELIST** e CCDT vengono utilizzati per creare connessioni JMS , è anche possibile avere un pool di connessioni che contiene connessioni a sistemi IBM MQ differenti.

Ad esempio, si supponga che una factory di connessione sia stata configurata con la proprietà **CONNECTIONNAMELIST** impostata sul seguente valore:

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Si supponga che la prima volta che un'applicazione tenta di creare una connessione a un gestore code autonomo da questa factory di connessione, il gestore code in esecuzione sul sistema `hostname1(port1)` non sia accessibile. Ciò significa che l'applicazione termina con una connessione al gestore code in esecuzione su `hostname2(port2)`.

Un'altra applicazione viene fornita e crea una connessione JMS dalla stessa factory di connessione. Il gestore code su `hostname1(port1)` ora è disponibile, quindi viene creata una nuova connessione JMS a questo IBM MQ sistema e viene restituita all'applicazione.

Al termine di entrambe le applicazioni, chiudono le relative connessioni JMS , il che determina la restituzione delle connessioni al pool di connessione.

Il risultato è che il pool di connessione per il factory di connessione contiene ora due connessioni JMS :

- Una connessione al gestore code in esecuzione su `hostname1(port1)`
- Una connessione al gestore code in esecuzione su `hostname2(port2)`

Ciò può causare problemi relativi al recupero della transazione. Se il sistema Java EE ha bisogno di eseguire il rollback di una transazione, deve essere in grado di connettersi a un gestore code che ha accesso ai log delle transazioni.

Implementazione della logica di riconnessione in un'applicazione Java EE
Le applicazioni enterprise JavaBeans e basate sul web che desiderano riconnettersi automaticamente se un gestore code non riesce devono implementare la propria logica di riconnessione.

Le seguenti opzioni forniscono ulteriori informazioni su come ottenere questo risultato.

Consenti l'esito negativo dell'applicazione

Questo approccio non richiede alcuna modifica dell'applicazione, ma richiede una riconfigurazione amministrativa della definizione del factory di connessioni per includere la proprietà **CONNECTIONNAMELIST** . Tuttavia, questo approccio richiede che il chiamante sia in grado di gestire un malfunzionamento in modo appropriato. Tenere presente che ciò è necessario anche per gli errori come MQRC_Q_FULL non correlati all'errore di connessione.

Codice di esempio per questo processo:

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
          try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
   }
 }
}
```

Il codice precedente presuppone che il factory di connessione, utilizzato da questo servlet, abbia la proprietà **CONNECTIONNAMELIST** definita.

Quando il servlet viene elaborato per la prima volta, viene creata una nuova connessione utilizzando la proprietà **CONNECTIONNAMELIST** , presumendo che non siano disponibili connessioni in pool da altre applicazioni che si collegano allo stesso gestore code.

Quando la connessione viene rilasciata in seguito a una chiamata `close()` , questa connessione viene restituita al pool e riutilizzata la prossima volta che il servlet viene eseguito - senza fare riferimento a **CONNECTIONNAMELIST** - finché non si verifica un errore di connessione, a quel punto viene generato un evento CONNECTION_ERROR_OCCURS. Questo evento richiede al pool di eliminare la connessione non riuscita.

Quando l'applicazione viene eseguita successivamente, non è disponibile alcuna connessione in pool e **CONNECTIONNAMELIST** viene utilizzato per connettersi al primo gestore code disponibile. Se si è verificato un failover del gestore code (ad esempio, l'errore non è stato un errore di rete transitorio), il servlet si connette all'istanza di backup quando è disponibile.

Se altre risorse, come i database, sono coinvolte nell'applicazione, potrebbe essere appropriato indicare che il server delle applicazioni deve eseguire il rollback della transazione.

Gestire la riconnessione all'interno dell'applicazione

Se il programma di richiamo non è in grado di elaborare un errore dal servlet, la riconnessione deve essere gestita all'interno dell'applicazione. Come mostrato nel seguente esempio, per gestire una riconnessione all'interno dell'applicazione è necessario che l'applicazione richieda una nuova connessione in modo che possa memorizzare nella cache la factory di connessione ricercata da JNDI

e gestire un JMSException come JMSCMQ0001: WebSphere MQ non riuscita con compcode '2' ('MQCC_FAILED ') motivo' 2009 '('MQRC_CONNECTION_BROKEN').

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
        setupResources();
    }
  }
```

Nel seguente esempio, setupResources() crea gli oggetti JMS e include un loop di sospensione e nuovo tentativo per gestire la riconnessione non istantanea. In pratica, questo metodo impedisce molti tentativi di riconnessione. Notare che le condizioni di uscita sono state omesse dall'esempio per chiarezza.

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

Se l'applicazione gestisce la riconnessione, è importante che l'applicazione rilasci tutte le connessioni che sono tenute ad altre risorse, se queste risorse sono altri gestori code IBM MQ o altri servizi di back end come i database. È necessario ristabilire queste connessioni quando la riconnessione a una nuova istanza del gestore code IBM MQ è completa. Se non si ristabiliscono le connessioni, le risorse del server delle applicazioni vengono conservate inutilmente durante il tentativo di riconnessione e potrebbero essere scadute quando vengono riutilizzate.

Utilizzo di WorkManager

Per le applicazioni di lunga durata (ad esempio, l'elaborazione batch) in cui il tempo di elaborazione è superiore a poche decine di secondi, è possibile utilizzare WebSphere Application Server WorkManager . Di seguito è riportato un esempio di frammenti di codice per WebSphere Application Server :

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
```

```
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

dove `web.xml` contiene:

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

e il batch viene ora implementato tramite l'interfaccia di lavoro:

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
               ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

Se l'elaborazione batch impiega molto tempo per essere eseguita, ad esempio, messaggi di grandi dimensioni, rete lenta o accesso esteso al database (specialmente quando accoppiato con un failover lento), il server inizia ad emettere avvisi di thread in sospeso, simili al seguente esempio:

WSVR0605W: il thread "`WorkManager.DefaultWorkManager` : 0" (00000035) è stato attivo per 694061 millisecondi e potrebbe essere bloccato. Nel server sono presenti 1 thread in totale che potrebbero essere bloccati.

Queste avvertenze possono essere ridotte riducendo la dimensione del batch o aumentando il timeout del thread in sospeso. Tuttavia, è generalmente preferibile implementare questa elaborazione in un EJB (per l'invio batch) o in un bean basato sui messaggi (per l'elaborazione di consume o consume e reply).

Notare che la riconnessione gestita dall'applicazione non fornisce una soluzione generale per la gestione degli errori di runtime e l'applicazione deve ancora gestire gli errori non correlati all'errore di connessione.

Ad esempio, il tentativo di inserire un messaggio in una coda piena (2053 MQRC_Q_FULL) o il tentativo di connettersi a un gestore code utilizzando credenziali di protezione non valide (2035 MQRC_NOT_AUTHORIZED).

L'applicazione deve anche gestire gli errori 2059 MQRC_Q_MGR_NOT_AVAILABLE quando non sono immediatamente disponibili istanze quando il failover è in corso. Ciò può essere ottenuto dall'applicazione che riporta le eccezioni JMS quando si verificano, invece di tentare in modalità non presidiata la riconnessione.

Lotto oggetti IBM MQ classes for JMS
L'uso di una forma di pool di connessioni al di fuori di Java EE consente di ridurre il carico complessivo risultante, ad esempio, da alcune applicazioni autonome che utilizzano i framework o che vengono distribuite in ambienti cloud e anche da un numero maggiore di connessioni client in `QueueManagers` , determinando un aumento del consolidamento dei server delle applicazioni e dei gestori code

All'interno del modello di programmazione Java EE , esiste un ciclo di vita ben definito dei vari oggetti in uso. Gli MDB (Message - Driven Bean) sono più limitati, mentre i servlet forniscono maggiore libertà. Pertanto, le opzioni di pool disponibili nei server Java EE si adattano ai vari modelli di programmazione utilizzati.

Con Java SE (o con un altro framework come Spring) i modelli di programmazione sono estremamente flessibili. Pertanto un'unica strategia di raggruppamento non si adatta a tutti. Si dovrebbe considerare se esiste un quadro che potrebbe fare qualsiasi forma di messa in comune, ad esempio la primavera.

La strategia di pool da utilizzare dipende dall'ambiente in cui è in esecuzione l'applicazione.

Pool di oggetti in un ambiente Java EE
I server delle applicazioni Java EE forniscono la funzione di pool di connessioni che può essere utilizzata dalle applicazioni bean basate sui messaggi, Enterprise Java Beans e Servlets.

WebSphere Application Server gestisce un lotto di connessioni ad un fornitore JMS , al fine di migliorare le prestazioni. Quando un'applicazione crea una connessione JMS , il server delle applicazioni determina se esiste già una connessione nel pool di connessioni libero. In tal caso, la connessione viene restituita all'applicazione; in caso contrario, viene creata una nuova connessione.

La Figura 41 a pagina 306 mostra in che modo le specifiche di attivazione e le porte listener stabiliscono una connessione JMS e utilizzano tale connessione per monitorare una destinazione per i messaggi in modalità normale.

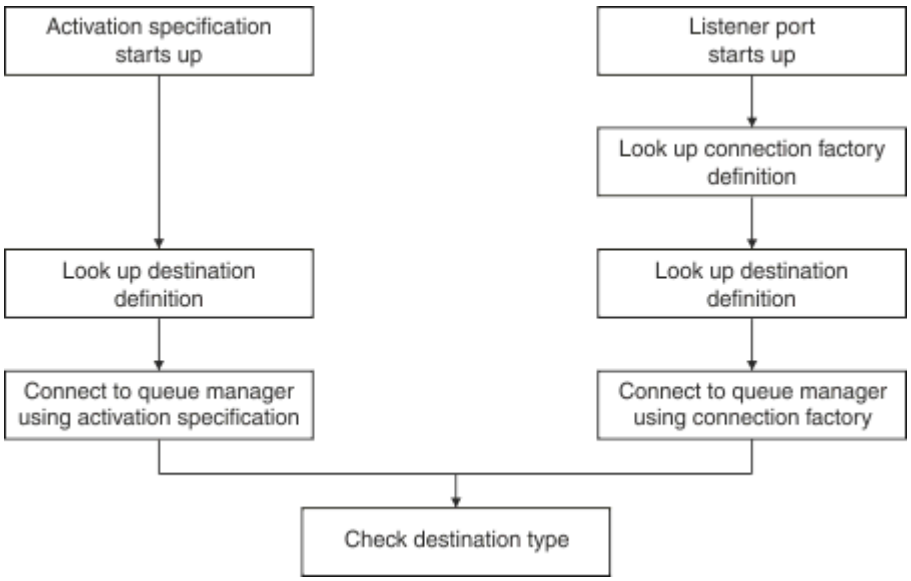


Figura 41. Modalità normale

Quando si utilizza il provider di messaggistica IBM MQ , le applicazioni che eseguono la messaggistica in uscita (come i bean e i servlet Java enterprise) e il componente porta listener bean basato sui messaggi, possono utilizzare questi pool di connessione.

Le specifiche di attivazione del provider di messaggistica IBM MQ utilizzano la funzionalità del pool di connessioni fornita dall'adattatore di risorse IBM MQ . Per ulteriori informazioni, consultare Configurazione delle proprietà per l'adattatore di risorse WebSphere MQ .

"Esempi di utilizzo del pool di connessione" a pagina 310 spiega come le applicazioni che eseguono la messaggistica in uscita e le porte listener utilizzano il pool libero quando creano le connessioni JMS .

"Thread di manutenzione del pool di connessioni liberi" a pagina 312 spiega cosa accade a queste connessioni quando un'applicazione, o una porta listener, ha terminato le connessioni.

"Esempi di thread di gestione pool" a pagina 314 spiega come viene ripulito il pool di connessioni libero per evitare che JMS le connessioni diventino obsolete.

WebSphere Application Server ha un limite sul numero di connessioni che possono essere create da un factory, specificato dalla proprietà *connessioni massime* del factory di connessione. Il valore predefinito per questa proprietà è 10, il che significa che possono essere create fino a 10 connessioni da un factory in qualsiasi momento.

Ogni factory ha un pool di connessioni libere associato. All'avvio del server delle applicazioni, i pool di connessioni liberi sono vuoti. Il numero massimo di connessioni che possono esistere nel lotto libero per un factory è specificato anche dalla proprietà Numero massimo di connessioni.

Suggerimento: Con JMS 2.0, una factory di connessione può essere utilizzata per creare sia connessioni che contesti. Come risultato, è possibile avere un pool di connessioni associato a un factory di connessione che contiene una combinazione di connessioni e contesti. Si consiglia di utilizzare una factory di connessione solo per creare connessioni o contesti. Ciò garantisce che il pool di connessioni per tale factory di connessione contenga solo oggetti di un singolo tipo, rendendo il pool più efficace.

Per informazioni sul funzionamento del pool di connessioni in WebSphere Application Server, consultare Configurazione del pool di connessioni per connessioni JMS. Per altri server delle applicazioni, fare riferimento alla documentazione appropriata del server delle applicazioni.

Modalità di utilizzo del lotto connessioni

Ogni factory di connessione JMS ha un pool di connessione associato e il pool di connessione contiene zero o più connessioni JMS . Ogni connessione JMS ha un pool di sessioni JMS associato e ogni pool di sessioni JMS contiene zero o più sessioni JMS .

Figura 42 a pagina 307 mostra la relazione tra questi oggetti.

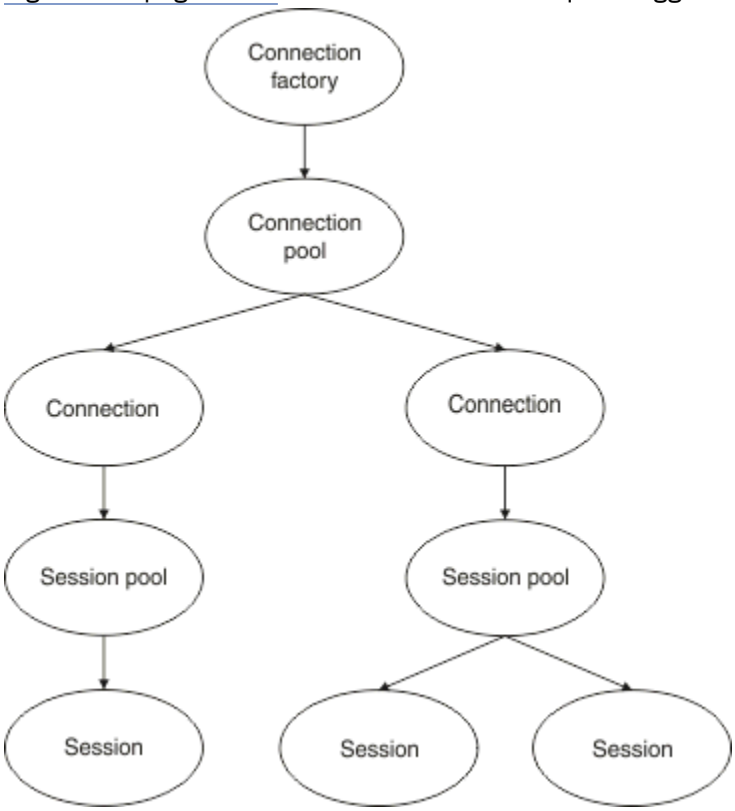


Figura 42. Pool di connessioni e pool di sessione

Quando una porta listener viene avviata o un'applicazione che desidera eseguire la messaggistica in uscita utilizza il factory per creare una connessione, la porta o l'applicazione richiama uno dei seguenti metodi:

- **connectionFactory.createConnection()**
- **ConnectionFactory.createConnection(String, String)**
- **QueueConnectionFactory.createQueueConnection()**
- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

Il gestore connessioni WebSphere Application Server tenta di ottenere una connessione dal pool libero per questo factory e la restituisce all'applicazione.

Se non ci sono connessioni libere nel pool e il numero di connessioni create da questo factory non ha raggiunto il limite specificato nella proprietà *Numero massimo di connessioni* di tale factory, Connection Manager crea una nuova connessione per l'applicazione da utilizzare.

Tuttavia, se un'applicazione tenta di creare una connessione, ma il numero di connessioni create da questo factory è già uguale alla proprietà *connessioni massime* del factory, l'applicazione attende che una connessione diventi disponibile (per essere reinserita nel pool libero).

Il tempo di attesa dell'applicazione è specificato nella proprietà *timeout connessione* del lotto connessioni, che ha un valore predefinito di 180 secondi. Se una connessione viene reinserita nel lotto libero entro questo periodo di 180 secondi, Connection Manager la rimette immediatamente fuori dal lotto e la trasmette all'applicazione. Tuttavia, se il periodo di timeout scade, viene generata una *ConnectionWaitTimeoutException* .

Quando un'applicazione ha terminato la connessione e la chiude richiamando:

- **Connection.close()**

- **QueueConnection.close()**
- **TopicConnection.close()**

la connessione viene effettivamente tenuta aperta e viene restituita al pool libero in modo che possa essere riutilizzata da un'altra applicazione. Pertanto, è possibile avere connessioni aperte tra WebSphere Application Server e il provider di JMS , anche se nessuna applicazione JMS è in esecuzione sul server delle applicazioni.

Proprietà pool di connessione avanzate
Esistono numerose proprietà avanzate che possono essere utilizzate per controllare il funzionamento dei pool di connessioni JMS .

Protezione da sovratensione

"Modalità di utilizzo del pool di connessioni da parte delle applicazioni che eseguono la messaggistica in uscita" a pagina 311 descrive l'utilizzo del metodo `sendMessage()`, che incorpora `connectionFactory.createConnection()`.

Considerare la situazione in cui si dispone di 50 EJB tutti che creano connessioni JMS dallo stesso factory di connessione come parte del metodo `ejbCreate()` .

Se tutti questi bean vengono creati contemporaneamente e non ci sono connessioni nel pool di connessioni libero del factory, il server delle applicazioni tenta di creare 50 connessioni JMS allo stesso fornitore JMS contemporaneamente. Il risultato è un carico significativo sia su WebSphere Application Server che sul fornitore JMS .

Le proprietà di protezione da sovraccarico possono impedire questa situazione limitando il numero di connessioni JMS che possono essere create da un factory di connessioni in qualsiasi momento e sfalsando la creazione di connessioni aggiuntive.

La limitazione del numero di connessioni JMS in qualsiasi momento viene ottenuta utilizzando due proprietà:

- Soglia di sovraccarico
- Intervallo di creazione del sovraccarico.

Quando le applicazioni EJB provano a creare una connessione JMS da una factory di connessione, il gestore connessioni verifica il numero di connessioni create. Se tale numero è minore o uguale al valore della proprietà `surge threshold` , il gestore connessioni continua ad aprire nuove connessioni.

Tuttavia, se il numero di connessioni che vengono create supera la proprietà `surge threshold` , il gestore connessioni attende il periodo di tempo specificato dalla proprietà `surge creation interval` prima di creare e aprire una nuova connessione.

Connessioni bloccate

Una connessione JMS viene considerata `stuck`, se un'applicazione JMS utilizza tale connessione per inviare una richiesta al provider JMS e il provider non risponde entro un determinato periodo di tempo.

WebSphere Application Server fornisce un modo per rilevare connessioni `stuck` JMS Per utilizzare questa funzione, è necessario impostare tre proprietà:

- Timer tempo di blocco
- Tempo di blocco
- Soglia di blocco

"Esempi di thread di gestione pool" a pagina 314 spiega come il thread di manutenzione del pool viene eseguito periodicamente e controlla il contenuto del pool libero di un factory di connessione, ricercando le connessioni che sono state inutilizzate per un periodo di tempo o che sono state esistenti per troppo tempo.

Per rilevare connessioni bloccate, il server delle applicazioni gestisce anche un thread di connessione bloccato che controlla lo stato di tutte le connessioni attive create da una factory di connessione per verificare se una di esse è in attesa di una risposta dal provider JMS .

Quando il thread di connessione bloccato viene eseguito viene determinato dalla proprietà `Stuck time timer` . Il valore predefinito per questa proprietà è zero, il che significa che il rilevamento della connessione bloccata non viene mai eseguito.

Se il thread trova un thread in attesa di una risposta, determina per quanto tempo è rimasto in attesa e confronta questo tempo con il valore della proprietà `Stuck time` .

Se il tempo impiegato dal provider JMS per rispondere supera il tempo specificato dalla proprietà `Stuck time` , il server delle applicazioni contrassegna la connessione JMS come bloccata.

Ad esempio, si supponga che il factory di connessione `jms/CF1` abbia la proprietà `Stuck time timer` impostata su 10 e la proprietà `Stuck time` impostata su 15.

Il thread di connessione bloccato diventa attivo ogni 10 secondi e verifica se una connessione creata da `jms/CF1` è in attesa da più di 15 secondi per una risposta da IBM MQ.

Si supponga che un EJB crei una connessione JMS a IBM MQ utilizzando `jms/CF1`e quindi tenti di creare una sessione JMS utilizzando tale connessione richiamando `Connection.createSession()`.

Tuttavia, qualcosa impedisce al provider JMS di rispondere alla richiesta. È possibile che la macchina sia stata bloccata o che un processo in esecuzione sul provider JMS sia bloccato, impedendo l'elaborazione di qualsiasi nuovo lavoro:

Dieci secondi dopo il richiamo dell'EJB `Connection.createSession()`, il timer della connessione bloccata diventa attivo e esamina le connessioni attive create da `jms/CF1`.

Si supponga che esista solo una connessione attiva, ad esempio denominata `c1`. Il primo EJB ha atteso 10 secondi una risposta a una richiesta inviata a `c1`, che è inferiore al valore di `Stuck time`, quindi il timer della connessione bloccata ignora questa connessione e diventa inattivo.

10 secondi dopo, il thread di connessione bloccato diventa nuovamente attivo e controlla le connessioni attive per `jms/CF1`. Come in precedenza, si supponga che vi sia solo una connessione, `c1`.

Sono trascorsi 20 secondi dal primo EJB denominato `createSession()`e l'EJB è ancora in attesa di una risposta. 20 secondi è più lungo del tempo specificato nella proprietà `Stuck time` , quindi il thread di connessione bloccato contrassegna `c1` come bloccato.

Se, cinque secondi dopo, IBM MQ finalmente risponde e consente al primo EJB di creare una sessione JMS , la connessione è di nuovo in uso.

Il server delle applicazioni conta il numero di connessioni JMS create da un factory di connessione bloccate. Quando un'applicazione utilizza tale factory di connessioni per creare una nuova connessione JMS e non vi sono connessioni libere nel pool libero di tale factory, il gestore connessioni confronta il numero di connessioni bloccate con il valore della proprietà `Stuck threshold` .

Se il numero di connessioni bloccate è inferiore al valore impostato per la proprietà `Stuck threshold`, il gestore connessioni crea una nuova connessione e la fornisce all'applicazione.

Tuttavia, se il numero di connessioni bloccate è uguale al valore della proprietà `Stuck threshold` , l'applicazione riceve un'eccezione di risorsa.

Partizioni pool

WebSphere Application Server fornisce due proprietà che consentono di partizionare il pool di connessione libero per una produzione connessioni:

- `Number of free pool partitions` indica al server delle applicazioni il numero di partizioni in cui si desidera dividere il pool di connessione libero.
- `Free pool distribution table size` determina la modalità di indicizzazione delle partizioni.

Lasciare queste proprietà sui valori predefiniti di zero, a meno che non venga richiesto di modificarle dal centro di supporto IBM .

Tenere presente che WebSphere Application Server dispone di un'ulteriore proprietà del pool di connessioni avanzata denominata `Number of shared partitions`. Questa proprietà specifica il numero di partizioni utilizzate per memorizzare le connessioni condivise. Tuttavia, poiché la condivisione delle connessioni JMS è sempre annullata, questa proprietà non viene applicata.

Esempi di utilizzo del pool di connessione
Il componente della porta del listener del bean basato sui messaggi e le applicazioni che eseguono la messaggistica in uscita utilizzano un lotto di connessione JMS .

Figura 43 a pagina 310 mostra come funziona il pool di connessione per WebSphere Application Server V7.5 e V8.0.

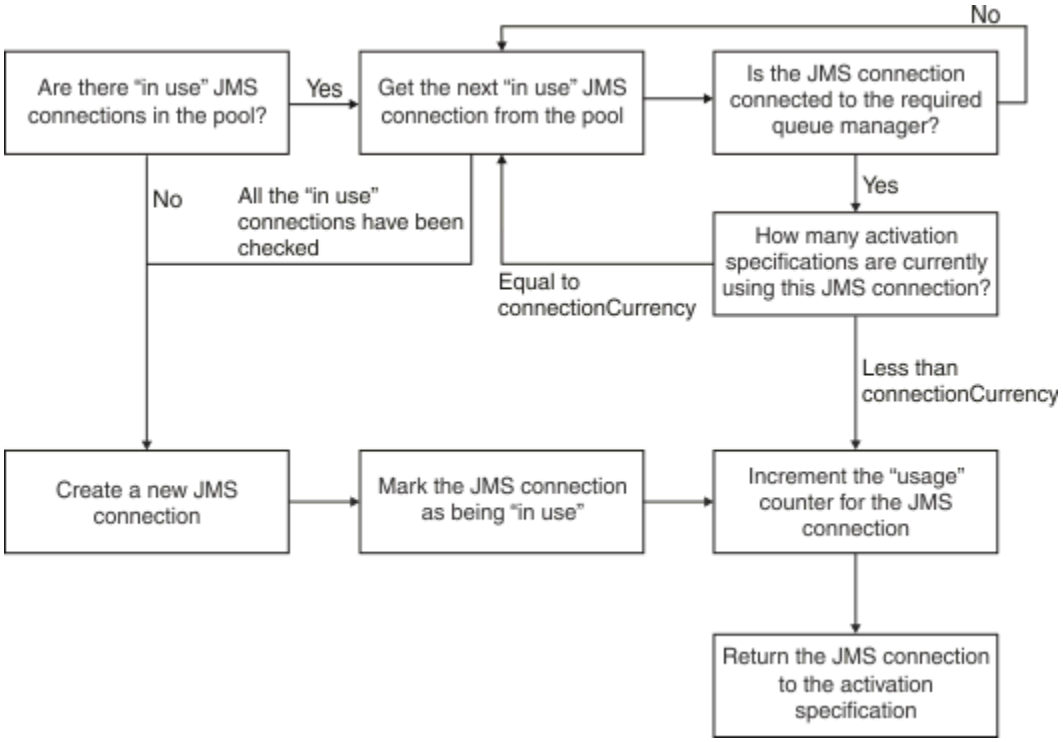


Figura 43. WebSphere Application Server V7.5 e V8.0 - come funziona il pool di connessione

Figura 44 a pagina 310 mostra come funziona il pool di connessioni per WebSphere Application Server V8.5.

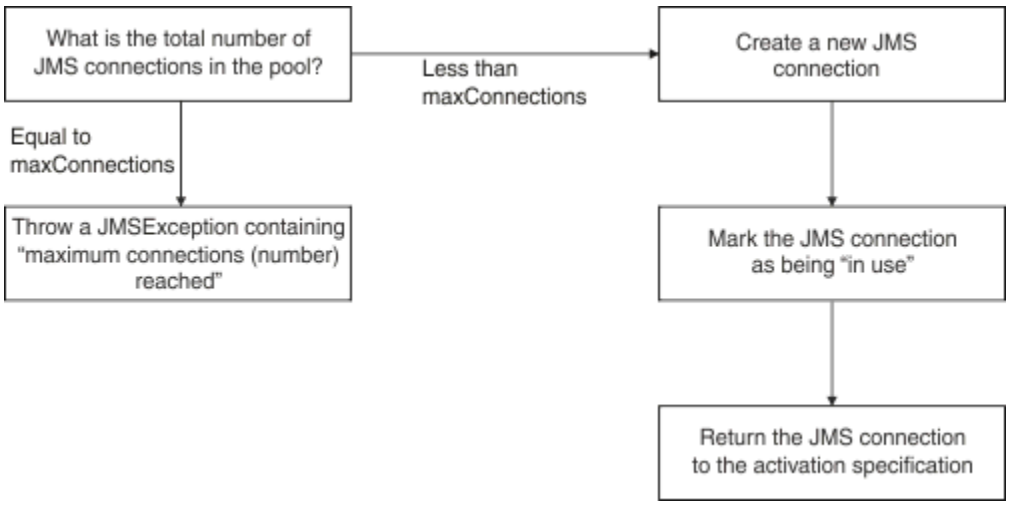


Figura 44. WebSphere Application Server V8.5 - funzionamento del pool di connessioni

Modalità di utilizzo del pool di connessioni da parte delle porte listener MDB

Si supponga di disporre di un MDB distribuito su un sistema WebSphere Application Server Network Deployment, che utilizza IBM MQ come provider JMS . L'MDB viene distribuito su una porta listener che utilizza un factory di connessione denominato, ad esempio, `jms/CF1`, che ha la proprietà *numero massimo di connessioni* impostata su 2, il che significa che è possibile creare solo due connessioni da questo factory alla volta.

Quando viene avviata la porta listener, la porta tenta di creare una connessione a IBM MQ, utilizzando la factory di connessione`jms/CF1` .

A tale scopo, la porta richiede una connessione dal gestore connessioni. Poiché questa è la prima volta che viene utilizzata la factory di connessione `jms/CF1` , non ci sono connessioni nel lotto di connessioni libero `jms/CF1` , quindi il gestore connessioni ne crea una nuova denominata, ad esempio, `c1`. Notare che questa connessione esiste per l'intera durata della porta listener.

Ora, considerare la situazione in cui si arresta la porta listener utilizzando la console di gestione WebSphere Application Server . In questo caso, il gestore connessioni prende la connessione e la reinserisce nel lotto libero. Tuttavia, la connessione a IBM MQ rimane aperta.

Se si riavvia la porta listener, la porta richiederà nuovamente al gestore connessioni una connessione al gestore code. Poiché ora si dispone di una connessione (`c1`) nel pool libero, il gestore connessioni prende questa connessione dal pool e la rende disponibile per la porta listener.

Ora, si supponga di avere un secondo MDB distribuito nel server delle applicazioni e che stia utilizzando una porta listener differente.

Si supponga, quindi, di provare ad avviare una terza porta listener, configurata anche per utilizzare la factory di connessione `jms/CF1` . La terza porta listener richiede una connessione dal gestore connessioni, che ricerca `jms/CF1` nel lotto libero e rileva che è vuota. Verifica quindi il numero di connessioni già create dalla factory `jms/CF1` .

Poiché la proprietà del numero massimo di connessioni per `jms/CF1` è impostato su 2 e sono già state create due connessioni da questo factory, il gestore connessioni attende 180 secondi (il valore predefinito della proprietà di timeout della connessione) per rendere disponibile una connessione.

Tuttavia, se si arresta la prima porta listener, la relativa connessione `c1` viene inserita nel lotto libero per `jms/CF1`. Il gestore connessioni richiama questa connessione e la fornisce al terzo listener.

Se si tenta ora di riavviare il primo listener, questo listener deve attendere l'arresto di una delle altre porte del listener prima che il primo listener possa essere riavviato. Se nessuna delle porte del listener in esecuzione viene arrestata entro 180 secondi, il primo listener riceve un errore `ConnectionWaitTimeoutException` e si arresta.

Modalità di utilizzo del pool di connessioni da parte delle applicazioni che eseguono la messaggistica in uscita

Per questa opzione, si supponga che sia presente un singolo EJB richiamato, ad esempio `EJB1`installato nel server delle applicazioni. Il bean implementa un metodo denominato `sendMessage()` mediante:

- Creazione di una connessione JMS a IBM MQ da un factory `jms/CF1`, utilizzando `connectionFactory.createConnection()`.
- Creazione di una sessione JMS dalla connessione.
- Creazione di un produttore di messaggi dalla sessione.
- Invio di un messaggio.
- Chiusura del produttore.
- Chiusura della sessione.
- Chiusura della connessione, richiamando `connection.close()`.

Si supponga che il pool libero per il factory jms/CF1 sia vuoto. Quando l'EJB viene richiamato per la prima volta, il bean tenta di creare una connessione a IBM MQ dal factory jms/CF1. Poiché il lotto libero per la produzione è vuoto, il gestore connessioni crea una nuova connessione e la fornisce a EJB1.

Prima che il metodo venga chiuso, il metodo richiama connection.close(). Invece di chiudere c1, il gestore connessioni prende la connessione e la inserisce nel pool libero per jms/CF1.

Al successivo richiamo di sendMessage() , il metodo connectionFactory.createConnection() restituisce c1 all'applicazione.

Si supponga di avere una seconda istanza dell'EJB in esecuzione contemporaneamente alla prima istanza. Quando entrambe le istanze richiamano sendMessage(), vengono create due connessioni dalla factory di connessione jms/CF1 .

Si supponga ora che venga creata una terza istanza del bean. Quando il terzo bean richiama sendMessage(), il metodo richiama connectionFactory.createConnection() per creare una connessione da jms/CF1.

Tuttavia, esistono attualmente due connessioni create da jms/CF1, che corrispondono al valore del numero massimo di connessioni per questo factory. Pertanto, il metodo createConnection() attende per 180 secondi (il valore predefinito della proprietà di timeout della connessione) che una connessione diventi disponibile.

Tuttavia, se il metodo sendMessage() per il primo EJB richiama connection.close() ed esce, la connessione utilizzata, c1, viene reinserita nel pool di connessione libero. Il gestore connessioni riporta la connessione dal pool libero e la fornisce al terzo EJB. La chiamata da quel bean a connectionFactory.createConnection() viene restituita, consentendo il completamento del metodo sendMessage().

Porte listener MDB e EJB che utilizzano lo stesso pool di connessioni

I due esempi precedenti mostrano in che modo le porte listener e gli EJB possono utilizzare il pool di connessioni in isolamento. Tuttavia, Š possibile avere sia una porta listener che un EJB in esecuzione all'interno dello stesso server delle applicazioni e creare connessioni JMS utilizzando la stessa produzione connessioni.

È necessario considerare le implicazioni di questa situazione

La cosa chiave da ricordare è che la factory di connessione è condivisa tra la porta del listener e l'EJB.

Ad esempio, si supponga di avere un listener e un EJB in esecuzione contemporaneamente. Entrambi utilizzano la factory di connessione jms/CF1 , il che significa che è stato raggiunto il limite di connessione specificato dalla proprietà del numero massimo di connessioni per tale factory.

Se si tenta di avviare un'altra porta listener o un'altra istanza di un EJB, è necessario attendere che una connessione venga restituita al pool di connessione libero per jms/CF1.

Thread di manutenzione del pool di connessioni liberi
Associato a ciascun pool di connessioni libere è un thread di gestione pool, che controlla il pool libero per garantire che le connessioni in esso siano ancora valide.

Se il thread di gestione del pool decide che una connessione nel pool libero deve essere eliminata, il thread chiude fisicamente la connessione JMS a IBM MQ.

Come funziona il thread di manutenzione del pool

Il funzionamento del thread di gestione del lotto è determinato dal valore di quattro proprietà del lotto di connessioni:

Timeout scaduto
La quantità di tempo per cui una connessione resta aperta.

Numero minimo di connessioni
Il numero minimo di connessioni che il gestore connessioni conserva nel lotto libero di una produzione connessioni.

Frequenza controllo
La frequenza di esecuzione del thread di gestione pool.

Timeout non utilizzato
Per quanto tempo una connessione rimane nel pool libero prima di essere chiusa.

Per impostazione predefinita, il thread gestito dal pool viene eseguito ogni 180 secondi, anche se questo valore può essere modificato impostando la proprietà **Reap time** del pool di connessioni.

Il thread di manutenzione esamina ogni connessione nel lotto, controlla per quanto tempo è stato nel lotto e quanto tempo è trascorso da quando è stato creato e utilizzato l'ultima volta.

Se la connessione non è stata utilizzata per un periodo più lungo del valore della proprietà **Unused timeout** per il pool di connessioni, il thread di manutenzione controlla il numero di connessioni attualmente nel pool libero. Se tale numero è:

- Maggiore del valore di **Minimum connections**, il gestore connessioni chiude la connessione.
- Equivale al valore di **Minimum connections**, la connessione non viene chiusa e rimane nel lotto libero.

Il valore predefinito della propriet ... **Minimum connections** Š *1*, il che significa che, per motivi di prestazioni, il gestore connessioni tenta sempre di mantenere almeno una connessione nel pool libero.

La proprietà **Unused timeout** ha un valore predefinito di 1800 secondi. Per impostazione predefinita, se una connessione viene reinserita nel pool libero e non viene riutilizzata per almeno 1800 secondi, tale connessione viene chiusa, purché venga chiusa, lasciando almeno una connessione nel pool libero.

Questa procedura evita che le connessioni inutilizzate diventino obsolete. Per disattivare questa funzione, impostare la proprietà **Unused timeout** su zero.

Se una connessione si trova nel pool libero e il tempo trascorso dalla sua creazione è maggiore del valore della proprietà **Aged timeout** per il pool di connessione, viene chiusa indipendentemente dal tempo trascorso dall'ultimo utilizzo.

Per default, la proprietà **Aged timeout** è impostata su zero, il che significa che il thread di manutenzione non esegue mai questo controllo. Le connessioni che sono state in giro per più tempo rispetto alla proprietà **Aged timeout** vengono eliminate indipendentemente dal numero di connessioni che rimarranno nel pool libero. Notare che la proprietà **Minimum connections** non ha alcun effetto in questa situazione.

Disabilitazione del thread di gestione pool

Dalla descrizione precedente, è possibile notare che il thread di manutenzione del pool esegue una grande quantità di lavoro quando è attivo, in particolare se vi è un numero elevato di connessioni nel pool libero del factory di connessione.

Ad esempio, si supponga che vi siano tre factory di connessione JMS , con la proprietà **Maximum connections** impostata su 10 per ogni factory. Ogni 180 secondi, tre thread di manutenzione del lotto diventano attivi ed eseguono la scansione dei pool liberi per ogni factory di connessione, rispettivamente. Se i pool liberi hanno molte connessioni, i thread di manutenzione hanno molto lavoro da eseguire, il che può influire in modo significativo sulle prestazioni.

È possibile disabilitare il thread di manutenzione del lotto per un singolo lotto di connessioni libere impostandone la proprietà **Reap time** su zero.

La disabilitazione del thread di manutenzione significa che le connessioni non vengono mai chiuse, anche se **Unused timeout** è trascorso. Tuttavia, le connessioni possono ancora essere chiuse se **Aged timeout** è stato superato.

Quando un'applicazione ha terminato con una connessione, il gestore connessioni verifica per quanto tempo la connessione è esistita e se tale periodo è più lungo del valore della proprietà **Aged timeout** , il gestore connessioni chiude la connessione invece di restituirla al pool libero.

Implicazioni transazionali del timeout scaduto

Come descritto nella sezione precedente, la proprietà **Aged timeout** specifica per quanto tempo una connessione al fornitore JMS rimane aperta prima che il gestore connessioni la chiuda.

Il valore predefinito per la proprietà **Aged timeout** è zero, il che significa che la connessione non verrà mai chiusa perché è troppo vecchia. È necessario lasciare la proprietà **Aged timeout** su questo valore, poiché l'abilitazione di **Aged timeout** potrebbe avere implicazioni transazionali quando si utilizza JMS all'interno di EJB.

In JMS, l'unità di transazione è una JMS sessione , creata da una JMS connessione . È la sessione JMS elencata nelle transazioni e non la connessione JMS .

A causa della progettazione del server delle applicazioni, le connessioni JMS possono essere chiuse perché **Aged timeout** è trascorso, anche se le sessioni JMS create da tale connessione sono coinvolte in una transazione.

La chiusura di una connessione JMS causa il rollback di qualsiasi lavoro transazionale in sospeso sulle sessioni JMS, come descritto nella specifica JMS . Tuttavia, il server delle applicazioni ignora che le sessioni JMS create dalla connessione non sono più valide. Quando il server tenta di utilizzare la sessione per eseguire il commit o il rollback di una transazione, si verifica un `IllegalStateException` .

Importante: Se si desidera utilizzare **Aged timeout** con le connessioni JMS dall'interno degli EJB, assicurarsi che il lavoro JMS sia esplicitamente sottoposto a commit sulla sessione JMS , prima che il metodo EJB che esegue le operazioni JMS venga chiuso.

Esempi di thread di gestione pool
Utilizzo dell'esempio EJB (Enterprise JavaBean) per comprendere il funzionamento del thread di gestione pool. Notare che è anche possibile utilizzare MDB (Message Driven Beans) e porte listener, poiché tutto ciò di cui si ha bisogno è un modo per ottenere connessioni nel pool libero.

Consultare "Modalità di utilizzo del pool di connessioni da parte delle applicazioni che eseguono la messaggistica in uscita" a pagina 311 per ulteriori dettagli sul metodo `sendMessage()` .

È stato configurato il factory di connessione con i valori seguenti:

- **Reap time** al valore predefinito di 180 secondi
- **Aged timeout** al valore predefinito di zero secondi
- **Unused timeout** impostato su 300 secondi

Dopo l'avvio del server delle applicazioni, viene richiamato il metodo `sendMessage()` .

Il metodo crea una connessione chiamata, ad esempio, `c1`, utilizzando il factory `jms/CF1`, utilizza tale factory per inviare un messaggio e quindi richiama `connection.close()`, che fa sì che `c1` venga inserito nel pool libero.

Dopo 180 secondi, il thread di manutenzione del pool viene avviato e viene visualizzato il pool di connessioni libere `jms/CF1` . È presente una connessione libera `c1` nel pool, quindi il thread di manutenzione esamina l'ora in cui è stata reinserita la connessione e la confronta con l'ora corrente.

Sono passati 180 secondi da quando la connessione è stata inserita nel pool libero, che è inferiore al valore della proprietà **Unused timeout** per `jms/CF1`. Pertanto, il thread di manutenzione lascia la connessione da sola.

180 secondi dopo, il thread di manutenzione del pool viene eseguito nuovamente. Il thread di manutenzione trova la connessione `c1`e determina che la connessione è stata nel pool per 360 secondi, che è più lunga del valore **Unused timeout** impostato, in modo che il gestore connessioni chiuda la connessione.

Se si esegue nuovamente il metodo `sendMessage()` , quando l'applicazione richiama `connectionFactory.createConnection()` , il gestore connessioni crea una nuova connessione a IBM MQ poiché il pool di connessioni libero per la factory di connessioni è vuoto.

L'esempio precedente mostra come il thread di manutenzione utilizza le proprietà **Reap time** e **Unused timeout** per evitare connessioni obsolete, quando la proprietà **Aged timeout** è impostata su zero.

Come funziona la proprietà **Aged timeout** ?

Nel seguente esempio, si supponga di aver impostato:

- Proprietà **Aged timeout** a 300 secondi
- **Unused timeout** su zero.

Richiamare il metodo `sendMessage()` e questo metodo tenta di creare una connessione dalla factory di connessione `jms/CF1` .

Poiché il pool libero per questo factory è vuoto, il gestore connessioni crea una nuova connessione, `c1`, e la restituisce all'applicazione. Quando `sendMessage()` richiama `connection.close()`, `c1` viene reinserito nel pool di connessione libero.

180 secondi dopo, viene eseguito il thread di manutenzione del pool. Il thread trova `c1` nel pool di connessione libero e controlla quanto tempo fa è stato creato. La connessione è esistita per 180 secondi, che è inferiore a **Aged timeout**, quindi il thread di manutenzione del pool lo lascia da solo e ritorna inattivo.

60 secondi dopo, `sendMessage()` viene richiamato di nuovo. Questa volta, quando il metodo richiama `connectionFactory.createConnection()`, il gestore connessioni rileva che è disponibile una connessione, `c1`, nel pool libero per `jms/CF1`. Il gestore connessioni toglie `c1` dal pool libero e fornisce tale connessione all'applicazione.

La connessione viene restituita al pool libero quando `sendMessage()` viene chiuso. 120 secondi dopo, il thread di manutenzione del pool si riattiva, esegue la scansione del contenuto del pool libero per `jms/CF1` e rileva `c1`.

Sebbene la connessione sia stata utilizzata solo 120 secondi fa, il thread di manutenzione del pool chiude la connessione, poiché la connessione è stata attiva per un totale di 360 secondi, che è superiore al valore di 300 secondi impostato per la proprietà **Aged timeout** .

In che modo la proprietà Numero minimo di connessioni influenza il thread di gestione del pool

Utilizzando nuovamente l'esempio “Modalità di utilizzo del pool di connessioni da parte delle porte listener MDB” a pagina 311 , si supponga di disporre di due MDB distribuiti nel proprio server delle applicazioni, ciascuno utilizzando una porta listener diversa.

Ogni porta listener è configurata per utilizzare la factory di connessione di `jms/CF1` , configurata con:

- Proprietà **Unused timeout** impostata su 120 secondi
- Proprietà **Reap time** impostata su 180 secondi
- Proprietà **Minimum connections** impostata su 1

Si supponga che il primo listener venga arrestato e che la relativa connessione `c1` venga inserita nel pool libero. 180 secondi dopo, il thread di manutenzione del pool si attiva, esegue la scansione del contenuto del pool libero per `jms/CF1`e rileva che `c1` è stato nel pool libero per un periodo più lungo del valore della proprietà **Unused timeout** per il factory di connessione.

Tuttavia, prima di chiudere `c1`, il thread di manutenzione del pool cerca di vedere quante connessioni rimarranno nel pool se questa connessione viene gettata via. Poiché `c1` è la sola connessione nel pool di connessioni libere, il gestore connessioni non la chiude, poiché ciò renderebbe il numero di connessioni che rimangono nel pool libero inferiore al valore impostato per **Minimum connections**.

Ora, si supponga che il secondo listener sia arrestato. Il pool di connessioni libere contiene ora due connessioni libere - `c1` e `c2`.

180 secondi dopo, il thread di manutenzione del pool viene eseguito nuovamente. A questo punto, `c1` è stato nel pool di connessioni libero per 360 secondi e `c2` per 180 secondi.

Il thread di gestione pool controlla `c1` e rileva che è stato nel pool per un periodo più lungo del valore della proprietà **Unused timeout** .

Il thread, quindi, verifica il numero di connessioni nel pool libero e lo confronta con il valore della proprietà **Minimum connections** . Poiché il lotto contiene due connessioni e **Minimum connections** è impostato su 1, il gestore connessioni chiude c1.

Il thread di manutenzione ora esamina c2. Questo è stato anche nel pool di connessioni libero per un periodo più lungo del valore della proprietà **Unused timeout** . Tuttavia, poiché la chiusura di c2 lascerebbe il pool di connessioni libere con un numero di connessioni inferiore al numero minimo impostato, il gestore connessioni lascia c2 da solo.

JMS connections e IBM MQ
Informazioni sull'utilizzo di IBM MQ come provider JMS .

Utilizzo del trasporto dei bind

Se una factory di connessione è stata configurata per utilizzare il trasporto bind, ogni connessione JMS stabilisce una conversazione (nota anche come **hconn**) con IBM MQ. La conversazione utilizza la comunicazione tra processi (o la memoria condivisa) per comunicare con il gestore code.

Utilizzo del trasporto client

Quando una factory di connessione del provider di messaggistica IBM MQ è stata configurata per utilizzare il trasporto client, ogni connessione creata da tale factory stabilirà una nuova conversazione (nota anche come**hconn**) con IBM MQ.

Per le factory di connessione che si collegano a un gestore code utilizzando la modalità normale del fornitore di messaggistica IBM MQ , è possibile che più connessioni JMS create dalla factory di connessione condividano una connessione TCP/IP con IBM MQ. Per ulteriori informazioni, fare riferimento a “Condivisione di una connessione TCP/IP in IBM MQ classes for JMS” a pagina 318.

Per determinare il numero massimo di canali client utilizzati dalle connessioni JMS in qualsiasi momento, aggiungere il valore della proprietà *Numero massimo di connessioni* per tutte le factory di connessione che fanno riferimento allo stesso gestore code.

Ad esempio, si supponga di avere due factory di connessione, jms/CF1 e jms/CF2 , che sono stati configurati per connettersi allo stesso gestore code IBM MQ utilizzando lo stesso canale IBM MQ .

Queste produzioni utilizzano le proprietà del pool di connessione predefinito, il che significa che *Numero massimo di connessioni* è impostato su 10. Se tutte le connessioni vengono utilizzate da jms/CF1 e jms/CF2 contemporaneamente, ci saranno 20 conversazioni tra il server delle applicazioni e IBM MQ.

Se la factory di connessione si connette al gestore code utilizzando la modalità normale del fornitore di messaggistica IBM MQ , il numero massimo di connessioni TCP/IP che può esistere tra il server delle applicazioni e il gestore code per queste factory di connessione è:

```
20/the value of SHARECNV for the IBM MQ channel
```

Se il factory di connessione è configurato per connettersi utilizzando la modalità di migrazione del fornitore di messaggistica IBM MQ , il numero massimo di connessioni TCP/IP tra il server delle applicazioni e IBM MQ per tali factory di connessione sarà 20 (uno per ogni connessione JMS nei pool di connessione per i due factory).

Concetti correlati
“Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

Pool di oggetti in un ambiente Java SE
Con Java SE (o con un altro framework come Spring) i modelli di programmazione sono estremamente flessibili. Pertanto un'unica strategia di raggruppamento non si adatta a tutti. Si dovrebbe considerare se esiste un quadro che potrebbe fare qualsiasi forma di messa in comune, ad esempio la primavera.

Altrimenti, la logica dell'applicazione potrebbe eseguire questa operazione. Ti chiedi quanto sia complessa l'applicazione stessa? È meglio comprendere l'applicazione e cosa richiede dalla connettività al sistema di messaggistica. Le applicazioni sono spesso scritte anche all'interno del proprio codice wrapper intorno all'API JMS di base.

Anche se questo può essere un approccio molto ragionevole e può nascondere la complessità, vale la pena tenere a mente che può introdurre problemi. Ad esempio, un metodo generico `getMessage()` , che viene richiamato di frequente, non dovrebbe solo aprire e chiudere i consumatori.

Punti da considerare:

- Per quanto tempo l'applicazione dovrà accedere a IBM MQ? Tutto il tempo, o solo occasionalmente.
- Con quale frequenza verranno inviati i messaggi? Meno frequentemente, più una singola connessione a IBM MQ potrebbe essere condivisa.
- Un'eccezione di connessione interrotta è di solito un segnale della necessità di ricreare una connessione raggruppata. Informazioni su:
  - Eccezioni di sicurezza o host non disponibile
  - Eccezioni di coda piena
- Se si verifica un'eccezione di connessione interrotta, cosa dovrebbe accadere alle altre connessioni libere nel pool? Dovrebbero essere chiusi e ricreati?
- Se viene utilizzato TLS, ad esempio, per quanto tempo si desidera che una singola connessione rimanga aperta?
- In che modo una connessione in pool si identificherà in modo tale che un amministratore del gestore code può individuare la connessione e tenerne traccia.

È necessario considerare tutti gli oggetti JMS per il pooling e il pool di quell' oggetto ogni volta che è possibile farlo. Gli oggetti includono:

- JMS connessioni
- Sessione
- Contesti
- Produttori e consumatori di tutti i tipi

Quando si utilizza il trasporto client, le connessioni, le sessioni e i contesti JMS utilizzeranno i socket durante la comunicazione con il gestore code IBM MQ . Raggruppando questi oggetti, i risparmi sono sul numero di connessioni IBM MQ in entrata (`hConns`) al gestore code e una riduzione del numero di istanze del canale.

L'uso del trasporto dei collegamenti al gestore code rimuove completamente il livello di rete. Tuttavia, molte applicazioni utilizzano il trasporto client per fornire una configurazione più altamente disponibile e bilanciata del carico di lavoro.

I produttori e i consumatori JMS aprono le destinazioni sul gestore code. Se viene aperto un numero minore di code o argomenti e più parti dell'applicazione utilizzano questi oggetti, la creazione di lotti può essere utile.

Da una prospettiva IBM MQ , questo processo salva una sequenza di operazioni MQOPEN e MQCLOSE.

Connessioni, sessioni e contesti

Tutti questi oggetti incapsulano gli handle di connessione IBM MQ nel gestore code e vengono generati da `ConnectionFactory`. È possibile aggiungere la logica a un'applicazione per limitare il numero di connessioni e altri oggetti, creati da un singolo factory di connessioni a uno specifico numero.

È possibile utilizzare una struttura dati semplice nell'applicazione per contenere le connessioni create. Il codice dell'applicazione che deve utilizzare una di queste strutture dati può *estrarre* un oggetto da utilizzare.

Prendere in considerazione i fattori seguenti:

- Quando rimuovere le connessioni dal pool? Generalmente, creare un listener di eccezioni sulla connessione. Quando il listener viene richiamato per elaborare un'eccezione, è necessario ricreare la connessione e tutte le sessioni create da tale connessione.
- Se una CCDT è in uso per il bilanciamento del carico di lavoro, le connessioni potrebbero andare a gestori code differenti. Ciò potrebbe essere applicabile per i requisiti di raggruppamento.

Tenere presente che la specifica JMS indica che si tratta di un errore di programmazione per più thread che accedono a una sessione o a un contesto contemporaneamente. Il codice IBM MQ JMS tenta di essere rigoroso nella gestione dei thread. Tuttavia, è necessario aggiungere la logica all'applicazione, per garantire che una sessione o un oggetto di contesto venga utilizzato solo da un thread alla volta.

Produttori e consumatori

Ogni producer e consumer creato apre una destinazione sul gestore code. Se la stessa destinazione verrà utilizzata per una varietà di attività, è opportuno mantenere aperti gli oggetti consumer o producer. Chiudere l'oggetto solo al termine di tutto il lavoro.

Anche se l'apertura e la chiusura di una destinazione sono operazioni brevi, se vengono eseguite frequentemente il tempo impiegato può sommare.

L'ambito di questi oggetti si trova all'interno della sessione o del contesto da cui vengono creati, quindi, è necessario che siano contenuti in tale ambito. In generale, le applicazioni sono scritte in modo che questo è abbastanza semplice da fare.

Monitoraggio

In che modo le applicazioni monitoreranno i relativi lotti oggetti? La risposta a questo problema è in gran parte determinata dalla complessità della soluzione della messa in comune attuata.

Se si considera un'implementazione del pool JavaEE, sono disponibili numerose opzioni, tra cui:

- Dimensione corrente dei lotti
- Tempo che gli oggetti hanno trascorso in essi
- Pulizia delle piscine
- Aggiornamento delle connessioni

È inoltre necessario considerare il modo in cui una singola sessione riutilizzata viene visualizzata sul gestore code. Esistono proprietà di factory di connessione per identificare l'applicazione (come `appName`) che potrebbero essere utili.

“Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

Condivisione di una connessione TCP/IP in IBM MQ classes for JMS
È possibile creare più istanze di un canale MQI per condividere una connessione TCP/IP singola.

Le applicazioni in esecuzione nello stesso ambiente di runtime Java e che utilizzano l'adattatore di risorse IBM MQ classes for JMS o IBM MQ per connettersi a un gestore code utilizzando il trasporto CLIENT, possono essere eseguite per condividere un'istanza del canale.

Se un canale è stato definito con il parametro **SHARECNV** impostato su un valore superiore a 1, tale numero di conversazioni può condividere un'istanza del canale. Per abilitare una factory di connessione o

una specifica di attivazione all'utilizzo di questa funzione, impostare la proprietà **SHARECONVALLOWED** su YES.

Ogni connessione JMS e sessione JMS creata da un'applicazione JMS crea la propria conversazione con il gestore code.

Quando una specifica di attivazione viene avviata, l'adattatore di risorse IBM MQ avvia una conversazione con il gestore code per la specifica di attivazione da utilizzare. Ogni sessione del server nel pool di sessioni del server associato alla specifica di attivazione avvia anche una conversazione con il gestore code.

L'attributo **SHARECNV** è un approccio ottimale alla condivisione della connessione. Pertanto, quando un valore **SHARECNV** maggiore di 0 viene utilizzato con IBM MQ classes for JMS, non è garantito che una nuova richiesta di connessione condivida sempre una connessione già stabilita.

Modalità di condivisione delle connessioni TCP/IP

Esistono due strategie disponibili per condividere le connessioni TCP/IP:

La strategia GLOBALE
: Questa strategia è la strategia predefinita per condividere le connessioni TCP/IP. Qualsiasi connessione o sessione JMS può utilizzare una conversazione su qualsiasi connessione TCP/IP adatta. L'idoneità è determinata da fattori quali indirizzo host, numero di porta, ID utente e password e parametri TLS/SSL.

  Questo approccio per la condivisione delle connessioni TCP/IP riduce il numero di istanze di canale in uso, ma a costo di contesa per l'accesso a un pool globale di connessioni TCP/IP.

La strategia CONNECTION
: Con questa strategia, le istanze del canale vengono condivise solo tra oggetti JMS correlati. In particolare, quando viene creata una connessione JMS , viene creata un'istanza del canale e le conversazioni aggiuntive su tale istanza del canale sono disponibili solo per le sessioni JMS create da tale connessione JMS .

  Se vengono create più conversazioni di quelle specificate dall'attributo SHARECNV, viene creata una nuova istanza del canale che può essere utilizzata solo dalle sessioni JMS create dalla connessione JMS originale.

  Questo approccio alla condivisione delle istanze del canale riduce il conflitto per le conversazioni, a discapito della potenziale richiesta di un numero significativamente maggiore di istanze del canale.

Specifica esplicita della strategia di condivisione di un'istanza del canale

V 9.3.2

Per impostazione predefinita, la strategia GLOBAL viene utilizzata se le applicazioni non sono ricollegabili. Le applicazioni ricollegabili utilizzano sempre la strategia CONNECTION.

Per le applicazioni che utilizzano IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, la strategia CONNECTION può essere abilitata per le applicazioni non ricollegabili a livello di applicazione. È possibile abilitare la strategia CONNECTION impostando la proprietà di sistema `com.ibm.mq.jms.channel.sharing` sul valore `CONNECTION`. Questo valore non è sensibile al maiuscolo / minuscolo e qualsiasi valore diverso da `CONNECTION` viene ignorato.

È possibile impostare la proprietà di sistema `com.ibm.mq.jms.channel.sharing` in uno dei seguenti modi:

- Impostare la proprietà come parte dell'inizializzazione JVM utilizzando l'opzione della riga comandi "-D":

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- Impostare la proprietà prima di qualsiasi utilizzo di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging utilizzando `System.setProperty()`

Calcolo del numero di istanze del canale per la strategia di condivisione GLOBAL

Utilizzare le formule riportate di seguito per determinare il numero massimo di istanze di canale create da un'applicazione:

Specifiche di attivazione

Numero di istanze del canale = (*maxPoolDepth_value* + 1) / *SHARECNV_value*

Dove *maxPoolDepth_value* è il valore della proprietà **maxPoolDepth** e *SHARECNV_value* è il valore della proprietà **SHARECNV** sul canale utilizzato dalla specifica di attivazione.

Altre applicazioni JMS

Numero di istanze canale = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

Dove *jms_connections* è il numero di connessioni che un'applicazione ha creato, *jms_sessions* è il numero di sessioni JMS create dall'applicazione e *SHARECNV_value* è il valore della proprietà **SHARECNV** sul canale utilizzato dalla specifica di attivazione.

Calcolo del numero di istanze del canale per la strategia di condivisione CONNECTION

Il numero di istanze del canale dipende dalla distribuzione delle sessioni JMS tra le connessioni JMS nell'applicazione.

Consentire una conversazione per la connessione JMS e una per ciascuna sessione JMS in quella connessione JMS , quindi dividere per il valore **SHARECNV** , arrotondando per eccesso. Questo calcolo fornisce le istanze del canale necessarie per tale connessione JMS .

Lo stesso principio può essere applicato alle specifiche di attivazione. Considerare la specifica di attivazione come una connessione JMS e la proprietà **maxPoolDepth** come il numero di sessioni JMS .

Esempi

I seguenti esempi mostrano come utilizzare le formule per calcolare il numero di istanze del canale create su un gestore code dalle applicazioni utilizzando l'adattatore di risorse IBM MQ classes for JMS o IBM MQ .

Esempio di applicazione JMS
Una connessione dell'applicazione JMS si connette a un gestore code utilizzando il trasporto CLIENT e crea una connessione JMS e tre sessioni JMS . Il canale che l'applicazione sta utilizzando per connettersi al gestore code ha la proprietà **SHARECNV** impostata sul valore 10. Quando l'applicazione è in esecuzione, ci sono quattro conversazioni tra l'applicazione e il gestore code e un'istanza di canale. Le quattro conversazioni condividono tutte l'istanza del canale.

Esempio di specifica di attivazione
Una specifica di attivazione si connette a un gestore code utilizzando il trasporto CLIENT. La specifica di attivazione è configurata con la proprietà **maxPoolDepth** impostata su 10. Il canale che la specifica di attivazione è configurata per utilizzare ha la proprietà **SHARECNV** impostata su 10. Quando la specifica di attivazione è in esecuzione ed elabora 10 messaggi contemporaneamente, il numero di conversazioni tra la specifica di attivazione e il gestore code è 11 (10 conversazioni per la sessione server e una per la specifica di attivazione). Il numero di istanze del canale utilizzate dalla specifica di attivazione è 2.

Esempio di specifica di attivazione
Una specifica di attivazione si connette a un gestore code utilizzando il trasporto CLIENT. La specifica di attivazione è configurata con la proprietà **maxPoolDepth** impostata su 5. Il canale che la specifica di attivazione è configurata per utilizzare ha la proprietà **SHARECNV** impostata su 0. Quando la specifica di attivazione è in esecuzione ed elabora 5 messaggi contemporaneamente, il numero di conversazioni tra la specifica di attivazione e il gestore code è 6 (cinque conversazioni per le sessioni del server e una per la specifica di attivazione). Il numero di istanze del canale utilizzate dalla specifica di attivazione è 6 perché la proprietà **SHARECNV** nel canale è impostata a 0, ogni conversazione utilizza la propria istanza del canale.

Attività correlate

“Determinazione del numero di connessioni TCP/IP create da WebSphere Application Server a IBM MQ” a pagina 503
Utilizzando la funzione di condivisione delle conversazioni, più conversazioni possono condividere istanze di canale MQI, questa è anche nota come connessione TCP/IP.

Specifica di un intervallo di porte per connessioni client in IBM MQ classes for JMS
Utilizzare la proprietà LOCALADDRESS per specificare un intervallo di porte a cui l'applicazione può collegarsi.

Quando un'applicazione IBM MQ classes for JMS tenta di connettersi a un gestore code IBM MQ in modalità client, un firewall potrebbe consentire solo le connessioni che hanno origine da porte specificate o da un intervallo di porte. In questa situazione, è possibile utilizzare la proprietà LOCALADDRESS di un oggetto factory ConnectionFactory, QueueConnectiono TopicConnectionper specificare una porta o un intervallo di porte a cui l'applicazione può collegarsi.

È possibile impostare la proprietà LOCALADDRESS utilizzando lo strumento di amministrazione IBM MQ JMS o richiamando il metodo setLocalAddress () in una applicazione JMS . Di seguito viene riportato un esempio di impostazione della proprietà dall'interno di un'applicazione:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

Quando l'applicazione si connette successivamente a un gestore code, l'applicazione si collega a un indirizzo IP locale e a un numero di porta compresi nell'intervallo tra 192.0.2.0(2000) e 192.0.2.0(3000).

In un sistema con più di un'interfaccia di rete, è anche possibile utilizzare la proprietà LOCALADDRESS per specificare quale interfaccia di rete deve essere utilizzata per una connessione.

Per una connessione in tempo reale a un broker, la proprietà LOCALADDRESS è rilevante solo quando viene utilizzato il multicast. In questo caso, è possibile utilizzare la proprietà per specificare quale interfaccia di rete locale deve essere utilizzata per una connessione, ma il valore della proprietà non deve contenere un numero di porta o un intervallo di numeri di porta.

Gli errori di connessione potrebbero verificarsi se si limita l'intervallo di porte. Se si verifica un errore, viene generata una JMSException con una MQException integrata che contiene il codice di errore IBM MQ MQRC_Q_MGR_NOT_AVAILABLE e il seguente messaggio:

```
Tentativo di connessione socket rifiutato a causa delle limitazioni LOCAL_ADDRESS_PROPERTY
```

Si potrebbe verificare un errore se tutte le porte nell'intervallo specificato sono in uso o se l'indirizzo IP, il nome host o il numero di porta specificati non sono validi (ad esempio, un numero di porta negativo).

Poiché IBM MQ classes for JMS potrebbe creare connessioni diverse da quelle richieste da un'applicazione, considerare sempre la possibilità di specificare un intervallo di porte. In generale, ogni sessione creata da un'applicazione richiede una porta e IBM MQ classes for JMS potrebbe richiedere tre o quattro porte aggiuntive. Se si verifica un errore di connessione, aumentare l'intervallo di porte.

Il pooling di connessioni, utilizzato per impostazione predefinita in IBM MQ classes for JMS, potrebbe avere un effetto sulla velocità con cui le porte possono essere riutilizzate. Di conseguenza, potrebbe verificarsi un errore di connessione mentre le porte vengono liberate.

Compressione canale in IBM MQ classes for JMS
Un'applicazione IBM MQ classes for JMS può utilizzare le funzionalità IBM MQ per comprimere un'intestazione o i dati di un messaggio.

La compressione dei dati che fluiscono su un canale IBM MQ può migliorare le prestazioni del canale e ridurre il traffico di rete. Utilizzando la funzione fornita con IBM MQ, è possibile comprimere i dati che fluiscono sui canali di messaggi e sui canali MQI. In entrambi i tipi di canale, è possibile comprimere i dati di intestazione e di messaggio indipendentemente l'uno dall'altro. Per impostazione predefinita, non viene compresso alcun dato su un canale.

Un'applicazione IBM MQ classes for JMS specifica le tecniche che è possibile utilizzare per comprimere i dati dell'intestazione o del messaggio su una connessione creando un oggetto java.util.Collection . Ogni

tecnica di compressione è un oggetto intero nella raccolta e l'ordine in cui l'applicazione aggiunge le tecniche di compressione alla raccolta è l'ordine in cui le tecniche di compressione vengono negoziate con il gestore code quando l'applicazione crea la connessione. L'applicazione può quindi passare la raccolta a un oggetto ConnectionFactory richiamando il metodo setHdrCompList(), per i dati di intestazione o il metodo setMsgCompList(), per i dati del messaggio. Quando l'applicazione è pronta, può creare la connessione.

I seguenti frammenti di codice illustreranno l'approccio descritto. Il primo frammento di codice mostra come implementare la compressione dei dati di intestazione:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

Il secondo frammento di codice mostra come implementare la compressione dei dati del messaggio:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

Nel secondo esempio, le tecniche di compressione vengono negoziate nell'ordine RLE, quindi ZLIBHIGH, quando viene creato il collegamento. La tecnica di compressione selezionata non può essere modificata durante la durata dell'oggetto Connection. Per utilizzare la compressione su una connessione, è necessario richiamare i metodi setHdrCompList() e setMsgCompList() prima di creare l'oggetto Connection.

Inserimento asincrono dei messaggi in IBM MQ classes for JMS
Normalmente, quando un'applicazione invia messaggi a una destinazione, l'applicazione deve attendere che il gestore code confermi di aver elaborato la richiesta. È possibile migliorare le prestazioni della messaggistica in alcune circostanze scegliendo invece di inserire i messaggi in modo asincrono. Quando un'applicazione inserisce un messaggio in maniera asincrona, il gestore code non restituisce l'esito positivo o negativo di ciascuna chiamata, ma è possibile controllare periodicamente la presenza di errori.

Se una destinazione restituisce il controllo all'applicazione, senza determinare se il gestore code ha ricevuto il messaggio in modo sicuro, dipende dalle seguenti proprietà:

La JMS proprietà destinazione PUTASYNCALLOWED (nome breve - PAALD).
PUTASYNCALLOWED controlla se le applicazioni JMS possono inserire i messaggi in modo asincrono, se questa opzione è consentita dalla coda sottostante o dall'argomento rappresentato dalla destinazione JMS .

La proprietà della coda o dell'argomento IBM MQ DEFPRESP (Tipo di risposta di inserimento predefinito).
DEFPRESP specifica se le applicazioni che inserono i messaggi nella coda o pubblicano i messaggi nell'argomento possono utilizzare la funzionalità di inserimento asincrono.

La tabella riportata di seguito mostra i valori possibili per le proprietà PUTASYNCALLOWED e DEFPRESP e le combinazioni di valori utilizzate per abilitare la funzionalità di inserimento asincrono:

Tabella 46. Come le proprietà PUTASYNCALLOWED e DEFPRESP si combinano per determinare se i messaggi vengono inseriti in una destinazione in modo asincrono.

proprietà Coda IBM MQ	PUTASYNCALLOWED = NO	PUTASYNCALLOWED = SI	Funzionalità put asincrono abilitata
DEFPRESP=SYNC	Funzionalità di inserimento asincrono non abilitata	Funzionalità put asincrono abilitata	PUTASYNCALLOWED = AS_DEST o AS_Q_DEF o AS_T_DEF
DEFPRESP=ASYNC	Funzionalità di inserimento asincrono non abilitata	Funzionalità put asincrono abilitata	PUTASYNCALLOWED = AS_DEST o AS_Q_DEF o AS_T_DEF

È possibile modificare il funzionamento specificando la proprietà di destinazione IBM MQ-JMS per dire "NO" o "YES" come mostrato nella tabella, ma può anche essere sovrascritto per l'intero Java Virtual Machine utilizzando la JVM **SystemProperty** e il valore:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

Per i messaggi inviati in una sessione sottoposta a transazione, l'applicazione determina se il gestore code ha ricevuto i messaggi in modo sicuro quando richiama `commit()`.

Se un'applicazione invia messaggi persistenti all'interno di una sessione sottoposta a transazione e uno o più messaggi non vengono ricevuti in modo sicuro, la transazione non riesce a eseguire il commit e genera un'eccezione. Tuttavia, se un'applicazione invia messaggi non persistenti all'interno di una sessione sottoposta a transazione e uno o più messaggi non vengono ricevuti in modo sicuro, la transazione viene sottoposta a commit correttamente. L'applicazione non riceve alcun feedback sul fatto che i messaggi non persistenti non sono arrivati in modo sicuro.

Per i messaggi non persistenti inviati in una sessione non sottoposta a transazione, la proprietà SENDCHECKCOUNT dell'oggetto *ConnectionFactory* specifica il numero di messaggi da inviare, prima che IBM MQ classes for JMS verifichi che il gestore code abbia ricevuto i messaggi in modo sicuro.

Se un controllo rileva che uno o più messaggi non sono stati ricevuti in modo sicuro e l'applicazione ha registrato un listener di eccezioni con la connessione, IBM MQ classes for JMS richiama il metodo `onException()` del listener di eccezioni per passare un'eccezione JMS all'applicazione.

L'eccezione JMS ha un codice di errore JMSWMQ0028 e questo codice visualizza il messaggio seguente:

```
At least one asynchronous put message failed or gave a warning.
```

L'eccezione JMS dispone anche di un'eccezione collegata che fornisce ulteriori dettagli. Il valore predefinito della proprietà SENDCHECKCOUNT è zero, il che significa che non vengono eseguiti tali controlli.

Questa ottimizzazione è molto utile per un'applicazione che si connette a un gestore code in modalità client e deve inviare una sequenza di messaggi in rapida successione, ma non richiede un feedback immediato dal gestore code per ogni messaggio inviato. Tuttavia, un'applicazione può ancora utilizzare questa ottimizzazione anche se si connette a un gestore code in modalità di bind, ma il vantaggio delle prestazioni previsto non è lo stesso.

Nota: Se si sta utilizzando un **MessageProducer** non identificato per inviare un messaggio in una transazione, per impostazione predefinita i messaggi vengono inseriti nella coda utilizzando il meccanismo di inserimento asincrono.

Ciò può verificarsi perché l'API JMS consente di creare il **MessageProducer** senza specificare una destinazione, utilizzando la sintassi:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

In questo scenario, la destinazione JMS viene fornita quando il messaggio viene inviato piuttosto che in anticipo quando viene creato il **`MessageProducer`** . In termini di API IBM MQ , ciò comporta l'emissione di MQPUT1 per inserire il messaggio nella coda.

Se si esegue questa operazione in un punto di sincronizzazione IBM MQ , il che significa (nella terminologia JMS) che inserisce il messaggio in una transazione, utilizzando una sessione JMS transatta o utilizzando un XA`Session` , l'API IBM MQ classes for JMS passa all'utilizzo dell'immissione asincrona.

Utilizzo della lettura anticipata con IBM MQ classes for JMS
La funzionalità di lettura anticipata fornita da IBM MQ consente ai messaggi non persistenti ricevuti al di fuori di una transazione di essere inviati a IBM MQ classes for JMS prima che un'applicazione li richieda. Il IBM MQ classes for JMS memorizza i messaggi in un buffer interno e li trasmette all'applicazione quando l'applicazione li richiede.

Le applicazioni IBM MQ classes for JMS che utilizzano `MessageConsumers` o `MessageListeners` per ricevere i messaggi da una destinazione esterna a una transazione possono utilizzare la funzionalità di lettura anticipata. L'utilizzo della lettura anticipata consente alle applicazioni che utilizzano questi oggetti di beneficiare di prestazioni migliori quando ricevono messaggi.

Se un'applicazione che utilizza `MessageConsumers` o `MessageListeners` può utilizzare la lettura anticipata dipende dalle seguenti proprietà:

La proprietà di destinazione JMS READAHEADALLOWED (nome breve - RAALD).
READAHEADALLOWED controlla se le applicazioni JMS possono utilizzare la lettura anticipata quando richiamano o sfogliano messaggi non persistenti all'esterno di una transazione, se la coda sottostante o l'argomento rappresentato dalla destinazione JMS , consente questa opzione.

La coda IBM MQ o la proprietà dell'argomento DEFREADA (lettura anticipata predefinita).
DEFREADA specifica se le applicazioni che stanno ricevendo o sfogliando messaggi non persistenti al di fuori di una transazione possono utilizzare la lettura anticipata.

La seguente tabella mostra i possibili valori per le proprietà READAHEADALLOWED e DEFREADA e le combinazioni di valori utilizzate per abilitare la funzionalità di lettura anticipata:

Tabella 47. Modalità di combinazione delle proprietà READAHEADALLOWED e DEFREADA per determinare se la lettura anticipata viene utilizzata durante la ricezione o l'esplorazione di messaggi non persistenti all'esterno di una transazione.

proprietà Coda IBM MQ	**READAHEADALLOWED = SÌ**	**READAHEADALLOWED = NO**	**AS_DEST o AS_Q_DEF o AS_T_DEF**
DEFREADA = NO	Funzionalità di lettura anticipata abilitata	Funzionalità di lettura anticipata non abilitata	Funzionalità di lettura anticipata non abilitata
DEFREADA = SI	Funzionalità di lettura anticipata abilitata	Funzionalità di lettura anticipata non abilitata	Funzionalità di lettura anticipata abilitata
DEFREADA = DISABILITATO	Funzionalità di lettura anticipata non abilitata	Funzionalità di lettura anticipata non abilitata	Funzionalità di lettura anticipata non abilitata

Se la funzionalità di lettura anticipata è abilitata, quando `MessageConsumer` o `MessageListener` viene creato da un'applicazione, IBM MQ classes for JMS crea un buffer interno per la destinazione monitorata da `MessageConsumer` o `MessageListener` . Esiste un buffer interno per ogni `MessageConsumer` o `MessageListener`. Il gestore code avvia l'invio di messaggi non persistenti a IBM MQ classes for JMS quando l'applicazione chiama uno dei seguenti metodi:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

Il IBM MQ classes for JMS restituisce automaticamente il primo messaggio all'applicazione, mediante la chiamata al metodo effettuata dall'applicazione. Gli altri messaggi non persistenti vengono memorizzati

da IBM MQ classes for JMS nel buffer interno creato per la destinazione. Quando l'applicazione richiede l'elaborazione del messaggio successivo, IBM MQ classes for JMS restituirà il successivo messaggio nel buffer interno.

Il IBM MQ classes for JMS richiede più messaggi non persistenti dal gestore code quando il buffer interno è vuoto.

Il buffer interno utilizzato da IBM MQ classes for JMS viene eliminato quando un'applicazione chiude un `MessageConsumer`o la sessione JMS a cui è associato un `MessageListener` .

Per `MessageConsumers`, i messaggi non elaborati nel buffer interno vengono persi.

Quando si utilizza `MessageListeners`, ciò che accade ai messaggi nel buffer interno dipende dalla proprietà di destinazione JMS READAHEADCLOSEPOLICY (nome breve - RACP). Il valore predefinito della proprietà è DELIVER_ALL, che indica che la sessione JMS utilizzata per creare `MessageListener` non viene chiusa fino a quando tutti i messaggi nel buffer interno non vengono consegnati all'applicazione. Se la proprietà è impostata su DELIVER_CURRENT, la sessione JMS verrà chiusa dopo che il messaggio corrente è stato elaborato dall'applicazione e tutti i restanti messaggi nel buffer interno verranno eliminati.

Pubblicazioni conservate in IBM MQ classes for JMS
Un client IBM MQ classes for JMS può essere configurato per utilizzare le pubblicazioni conservate.

Un autore (publisher) può specificare che una copia di una pubblicazione deve essere conservata in modo che possa essere inviata ai futuri sottoscrittori (subscriber) che registrano un interesse per l'argomento. Ciò viene eseguito in IBM MQ classes for JMS impostando la proprietà integer JMS_IBM_RETAIN sul valore 1. Le costanti sono state definite per questi valori nell'interfaccia com.ibm.msg.client.jms.JmsConstants . Ad esempio, se è stato creato un messaggio *msg*, per impostarlo come pubblicazione conservata utilizzare il seguente codice:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

È ora possibile inviare il messaggio come normale. JMS_IBM_RETAIN può anche essere interrogato in un messaggio ricevuto. È quindi possibile verificare se un messaggio ricevuto è una pubblicazione conservata.

Supporto XA in IBM MQ classes for JMS

JMS supporta le transazioni conformi a XA nei bind e nelle modalità client con un gestore transazioni supportato all'interno di un contenitore JEE .

Se si richiede la funzione XA in un ambiente del server delle applicazioni, è necessario configurare l'applicazione in modo appropriato. Fare riferimento alla documentazione del proprio server delle applicazioni per informazioni su come configurare le applicazioni per utilizzare le transazioni distribuite.

Un gestore code IBM MQ non può agire come gestore transazioni per JMS.

Ritardo di consegna per i messaggi JMS

Per JMS 2.0 o versioni successive, è possibile specificare un ritardo di consegna quando si invia un messaggio. Il gestore code non consegna il messaggio fino a quando non è trascorso il ritardo di consegna specificato.

Un'applicazione può specificare un ritardo di recapito in millisecondi, quando invia un messaggio, utilizzando `MessageProducer.setDeliveryDelay(long deliveryDelay)` o `JMSProducer.setDeliveryDelay(long deliveryDelay)`. Questo valore viene aggiunto all'ora in cui il messaggio viene inviato e fornisce la prima ora in cui qualsiasi altra applicazione può ricevere tale messaggio.

Il ritardo di consegna viene implementato utilizzando una singola coda di staging interna. I messaggi con un ritardo di recapito diverso da zero vengono inseriti in questa coda con un'intestazione che indica il ritardo di recapito e le informazioni sulla coda di destinazione. Un componente del gestore code chiamato processore di ritardo consegna monitora i messaggi sulla coda di staging. Quando il ritardo di recapito di

un messaggio viene completato, il messaggio viene eliminato dalla coda di staging e inserito nella coda di destinazione.

Client di messaggistica

L'implementazione IBM MQ del ritardo di consegna è disponibile solo per l'utilizzo quando si utilizza il client JMS . Le seguenti limitazioni si applicano se si utilizza il ritardo di consegna con IBM MQ. Queste restrizioni si applicano anche a `MessageProducers` e `JMSProducers`, ma `JMSRuntimeExceptions` viene generato nel caso di `JMSProducers`.

- Qualsiasi tentativo di richiamare `MessageProducer.setDeliveryDelay` con un valore diverso da zero quando si è connessi a un gestore code precedente a IBM MQ 8.0, risulta in un `JMSException` con un messaggio MQRC_FUNCTION_NOT_SUPPORTED.
- Il ritardo di recapito non è supportato per le destinazioni cluster che hanno un valore **DEFBIND** diverso da MQBND_BIND_NOT_FIXED. Se per un `MessageProducer` è impostato un ritardo di consegna diverso da zero e viene effettuato un tentativo di invio a una destinazione che non corrisponde a questo requisito, la chiamata risulta in un `JMSException` con un messaggio MQRC_OPTIONS_ERROR.
- Qualsiasi tentativo di impostare un valore TTL (time to live) inferiore a un ritardo di consegna diverso da zero precedentemente specificato, o viceversa, risulta in un `JMSException` con un messaggio MQRC_EXPIRY_ERROR. Questa verifica viene effettuata quando si richiamano i metodi `setTimeToLive` o `setDeliveryDelay` o `send` , in base alla serie esatta di operazioni scelte.
- L'utilizzo delle pubblicazioni conservate e il ritardo di consegna non è supportato. Il tentativo di pubblicare un messaggio con un ritardo di consegna se tale messaggio è stato contrassegnato come conservato utilizzando `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` risulta in un messaggio `JMSException` con un messaggio MQRC_OPTIONS_ERROR.
- Il ritardo di consegna e il raggruppamento dei messaggi non sono supportati e qualsiasi tentativo di utilizzare questa combinazione risulta in un `JMSException` con un messaggio MQRC_OPTIONS_ERROR.

Qualsiasi errore nell'invio di un messaggio con ritardo di consegna determina l'emissione da parte del client di un `JMSException` con un messaggio di errore appropriato, ad esempio una coda piena. In alcune situazioni, il messaggio di errore potrebbe essere applicato alla destinazione, alla coda di staging o a entrambi.

Nota: IBM MQ consente alle applicazioni che inserendo un messaggio in un'unità di lavoro di ottenere nuovamente lo stesso messaggio anche se l'unità di lavoro non ha eseguito il commit. Questa tecnica non funziona con il ritardo di consegna poiché il messaggio non viene inserito nella coda di staging fino a quando non viene eseguito il commit dell'unità di lavoro e, di conseguenza, non verrà inviato alla destinazione di destinazione.

Autorizzazione

IBM MQ esegue controlli di autorizzazione sulla destinazione originale quando l'applicazione invia un messaggio con un ritardo di consegna diverso da zero. Se l'applicazione non è autorizzata, l'invio ha esito negativo. Quando il gestore code rileva che il ritardo di consegna di un messaggio è completo, apre la coda di destinazione. A questo punto non viene eseguita alcuna verifica di autorizzazione.

SYSTEM.DDELAY.LOCAL.QUEUE

Una coda di sistema, SYSTEM.DDELAY.LOCAL.QUEUEviene utilizzato per implementare il ritardo di consegna.

- Multi Su Multiplatforms, SYSTEM.DDELAY.LOCAL.QUEUE esiste per impostazione predefinita. La coda di sistema deve essere modificata in modo che gli attributi MAXMSGL e MAXDEPTH siano sufficienti per il carico previsto.

- z/OS Su IBM MQ for z/OS, SYSTEM.DDELAY.LOCAL.QUEUE viene utilizzata come coda di staging per i messaggi inviati con ritardo di recapito alle code locali e condivise. Su z/OS, la coda deve essere creata e definita in modo che gli attributi MAXMSGL e MAXDEPTH siano sufficienti per il carico previsto.

Quando questa coda viene creata, deve essere protetta in modo che il minor numero possibile di utenti possa accedervi. L'accesso alla coda deve essere solo per scopi di manutenzione e monitoraggio.

Quando un messaggio viene inviato da un'applicazione JMS con ritardo di consegna diverso da zero, viene inserito in questa coda con un nuovo ID messaggio. L'ID messaggio originale viene inserito nell'ID di correlazione del messaggio. Questo ID di correlazione consente a un'applicazione di richiamare un messaggio dalla coda di staging quando richiesto, ad esempio se un ritardo di consegna elevato è stato utilizzato per errore.

Considerazioni per z/OS

z/OS

Se il sistema è in esecuzione su z/OS, ci sono ulteriori considerazioni da prendere in considerazione se si desidera utilizzare il ritardo di consegna.

Se deve essere utilizzato il ritardo di consegna, la coda di sistema SYSTEM.DDELAY.LOCAL.QUEUE deve essere definito. Deve essere definito con una classe di memoria sufficiente per il carico previsto e con INDXTYPE (NONE) e MSGDLVSQ (FIFO) specificati. Viene fornita una definizione di esempio della coda di sistema, commentata, nel JCL CSQ4INSG .

Code condivise

Il ritardo di consegna è supportato per l'invio di messaggi alle code condivise. Tuttavia, esiste solo una singola coda di staging privata utilizzata indipendentemente dal fatto che la coda di destinazione sia condivisa o meno. Il gestore code che possiede la coda privata deve essere in esecuzione per poter inviare il messaggio ritardato alla coda condivisa di destinazione al termine del ritardo.

Nota: Se un messaggio non persistente viene inserito con un ritardo di consegna in una coda condivisa e il gestore code proprietario della coda di staging si arresta, il messaggio originale viene perso. Di conseguenza, è più probabile che i messaggi non persistenti inviati con ritardo di recapito a una coda condivisa vadano persi rispetto ai messaggi non persistenti inviati senza ritardo di recapito a una coda condivisa.

Risoluzione destinazione di destinazione

Se il messaggio viene inviato a una coda, la risoluzione viene eseguita due volte; una volta dall'applicazione JMS e una volta dal gestore code quando il messaggio viene eliminato dalla coda di staging e viene inviato alla coda di destinazione.

Le sottoscrizioni di destinazione per le pubblicazioni corrispondono quando l'applicazione JMS richiama il metodo di invio.

Se un messaggio viene inviato con persistenza o priorità in base alla definizione della coda, il valore viene impostato sulla prima risoluzione e non sulla seconda.

Intervallo di scadenza

Il ritardo di consegna conserva il comportamento della proprietà di scadenza, **`MQMD.Expiry`**. Ad esempio, se un messaggio è stato inserito da un'applicazione JMS con un intervallo di scadenza di 20.000 ms e un ritardo di consegna di 5.000 ms e ottenuto dopo un tempo trascorso di 10.000 ms, il valore del campo MQMD.expiry potrebbe essere di circa 50 decimi di secondo. Questo valore indica che sono trascorsi 15 secondi dal momento in cui è stato immesso il messaggio al momento in cui è stato ricevuto.

Se un messaggio scade mentre è nella coda di staging ed è impostata una delle opzioni MQRO_EXPIRATION_ *, il report generato è per il messaggio originale come inviato dall'applicazione, l'intestazione utilizzata per contenere le informazioni sul ritardo di consegna viene rimossa.

Arresto e avvio del processore di ritardo consegna

z/OS Su z/OS, il processore di ritardo recapito è integrato nello spazio di indirizzi MSTR del gestore code. Quando il gestore code viene avviato, viene avviato anche il processore di ritardo consegna. Se la coda di staging è disponibile, apre la coda e attende che i messaggi arrivino su di essa per essere elaborati. Se la coda di staging non è stata definita, o è disabilitata per le ricezioni, o si verifica un altro errore, il processore di ritardo consegna si arresta. Se la coda di staging viene successivamente definita o modificata per essere abilitata, il processore di ritardo consegna viene riavviato. Se il processore dei ritardi di consegna si arresta per qualsiasi altro motivo, può essere riavviato modificando l'attributo **PUT** della coda di staging da ENABLED a DISABLED e di nuovo a ENABLED . Se è necessario arrestare il processore di ritardo consegna per qualsiasi motivo, impostare l'attributo PUT della coda di staging su DISABLED.

Multi Su Multiplatforms, il processore di ritardo viene avviato con il gestore code e viene riavviato automaticamente in caso di errore recuperabile.

Impossibile inserire nella coda di destinazione

Se un messaggio ritardato non può essere inserito nella coda di destinazione una volta completato il ritardo, il messaggio viene gestito come indicato nelle opzioni di report: viene eliminato o inviato alla coda di messaggi non recapitati. Se questa azione ha esito negativo, viene effettuato un tentativo di inserire il messaggio in un secondo momento. Se l'azione ha esito positivo, viene generato un report di eccezione e inviato alla coda specificata, se il report è richiesto. Se non è stato possibile inviare il messaggio di report, questo viene inviato alla coda dei messaggi non recapitabili. Se l'invio del report alla DLQ (dead letter queue) ha esito negativo e il messaggio è persistente, tutte le modifiche vengono eliminate e il messaggio originale viene sottoposto a rollback e riconsegnato in un secondo momento. Se il messaggio non è persistente, il messaggio di report viene eliminato, ma viene eseguito il commit di altre modifiche. Se una pubblicazione ritardata non può essere consegnata perché un sottoscrittore ha annullato la sottoscrizione o, nel caso di un sottoscrittore non durevole, perché si è disconnesso, il messaggio viene eliminato in modalità non presidiata. I messaggi di report vengono ancora generati come descritto in precedenza.

Se una pubblicazione ritardata non può essere consegnata a un sottoscrittore e viene invece inserita nella coda di messaggi non recapitabili e l'inserimento nella coda di messaggi non recapitabili ha esito negativo, il messaggio viene eliminato.

Per ridurre la probabilità che l'inserimento nella coda di destinazione non riesca dopo il completamento del ritardo di recapito, il gestore code esegue alcuni controlli di base quando il client JMS invia un messaggio con ritardo di recapito diverso da zero. Questi controlli includono se la coda è disabilitata, se il messaggio è più grande della lunghezza massima consentita e se la coda è piena.

Pubblicazione/sottoscrizione

La corrispondenza di una pubblicazione alle sottoscrizioni disponibili si verifica quando l'applicazione JMS invia un messaggio con un ritardo di consegna diverso da zero. Un messaggio per ogni utente corrispondente viene inserito nel SISTEMA SYSTEM.DDELAY.LOCAL.QUEUE , dove viene conservata fino al completamento del ritardo di recapito. Se uno di tali sottoscrittori è una sottoscrizione proxy per un altro gestore code, il fan - out su tale gestore code si verifica dopo il completamento del ritardo di recapito. Ciò potrebbe comportare che i sottoscrittori (subscriber) sull'altro gestore code ricevano le pubblicazioni originariamente pubblicate prima della sottoscrizione. Si tratta di una deviazione dalla specifica JMS 2.0 o successiva.

Il ritardo di consegna con la pubblicazione / sottoscrizione è supportato solo se l'argomento di destinazione è configurato con (N) PMSGDLV = ALLAVAIL. Un tentativo di utilizzare altri valori causa un errore MQRC_PUBLICATION_FAILURE. Se il processore dei ritardi di consegna ha esito negativo mentre inserisce il messaggio nella coda di destinazione, il risultato è quello descritto nella sezione "Errore di inserimento nella coda di destinazione".

Messaggi di report

Tutte le opzioni di report sono supportate e azionate dal processore di consegna, diverse dalle seguenti opzioni ignorate, ma trasmesse al messaggio quando viene inviato alla coda di destinazione:

- COA_MQRO*
- COD_MQRO*
- PAN/MQRO_NAN MQRO
- ATTIVITÀ MQRO

Sottoscrizioni clonate e condivise

In IBM MQ 8.0 o versioni successive, esistono due metodi per fornire a più consumatori l'accesso alla stessa sottoscrizione. Questi due metodi vengono utilizzati utilizzando sottoscrizioni clonate o sottoscrizioni condivise.

Sottoscrizioni clonate

La sottoscrizione clonata è un'estensione IBM MQ . Le sottoscrizioni clonate abilitano più consumer in differenti JVM (Java virtual machine), l'accesso simultaneo alla sottoscrizione. Questo comportamento può essere utilizzato impostando la proprietà **CLONESUPP** su `Abilitato` su un oggetto ConnectionFactory . Per impostazione predefinita, **CLONESUPP** è `Disabilitato`. Le sottoscrizioni clonate possono essere abilitate solo su sottoscrizioni durevoli. Se **CLONESUPP** è abilitato, ogni connessione successiva effettuata utilizzando questo ConnectionFactory viene clonata.

Una sottoscrizione durevole può essere considerata clonata se uno o più consumer vengono creati per ricevere messaggi da tale sottoscrizione, ovvero, sono stati creati specificando lo stesso nome di sottoscrizione. Questa operazione può essere eseguita solo se la connessione in cui sono stati creati i consumer è **CLONESUPP** impostata su `Abilitato` su MQConnectionFactory. Quando un messaggio viene pubblicato sull'argomento della sottoscrizione, una copia di tale messaggio viene inviata alla sottoscrizione. Il messaggio è disponibile per tutti i consumer, ma solo uno lo riceve.

Nota: L'abilitazione delle sottoscrizioni clonate estende le specifiche JMS .

Sottoscrizioni condivise

Le sottoscrizioni condivise, che sono state introdotte dalla specificazione JMS 2.0 , consentono la condivisione dei messaggi da una sottoscrizione argomento tra più consumer. Ogni messaggio della sottoscrizione viene consegnato solo a uno dei consumer di tale sottoscrizione. Le sottoscrizioni condivise sono abilitate dalla chiamata pertinente all'API JMS 2.0 o successiva.

Le API possono essere richiamate in uno dei seguenti modi:

- Da un'applicazione Java SE (o Java EE Client Container).
- Da un servlet o dall'implementazione di un MDB.

La specifica JMS 2.0 o successiva non definisce alcun modo standard di guidare un MDB da una sottoscrizione condivisa, quindi IBM MQ 8.0 o successiva fornisce la proprietà della specifica di attivazione **sharedSubscription** per questo scopo. Per ulteriori informazioni su questa proprietà, consultare “Configurazione dell'adattatore di risorse per la comunicazione in entrata” a pagina 454 e “Esempi di come definire la proprietà sharedSubscription” a pagina 470.

Se una sottoscrizione condivisa è abilitata, non è possibile annullarla.

Le sottoscrizioni condivise possono essere create come sottoscrizioni durevoli o non durevoli. Non è richiesto di creare separatamente alcun oggetto sul lato del gestore code oltre alla normale configurazione di JMS . Tutti gli oggetti richiesti vengono creati dinamicamente.

Scelta tra sottoscrizioni condivise o clonate

Quando si decide se utilizzare sottoscrizioni condivise o clonate, considerare i vantaggi di entrambe. Laddove possibile, utilizzare le sottoscrizioni condivise poiché si tratta di un comportamento definito dalla specifica, piuttosto che un'estensione specifica di IBM MQ .

La seguente tabella contiene alcuni dei punti da considerare quando si decide tra sottoscrizioni condivise e clonate:

Tabella 48. Confronto delle considerazioni per le sottoscrizioni condivise e le sottoscrizioni clonate

Sottoscrizioni condivise	**Sottoscrizioni clonate**
Le sottoscrizioni condivise sono una parte standard della specifica JMS 2.0 o successiva.	Le sottoscrizioni clonate sono un'estensione IBM MQ specifica.
Le sottoscrizioni condivise vengono create utilizzando chiamate esplicite al metodo API.	Le sottoscrizioni clonate sono controllate amministrativamente al livello ConnectionFactory .
Le sottoscrizioni condivise possono essere durevoli o non durevoli.	Le sottoscrizioni clonate possono essere durevoli.
Le sottoscrizioni condivise vengono create esplicitamente in base alla singola sottoscrizione.	Le sottoscrizioni clonate vengono utilizzate per qualsiasi sottoscrizione durevole in una connessione per cui la funzione è abilitata.
Se una sottoscrizione viene creata come condivisa, non può essere successivamente modificata in non condivisa o viceversa.	Una sottoscrizione può essere modificata da clonata a non clonata ogni volta che viene riaperta se la proprietà **CLONESUPP** della connessione proprietaria è stata modificata.

Concetti correlati

Sottoscrittori e sottoscrizioni

Durata sottoscrizione

Attività correlate

Utilizzo delle sottoscrizioni condivise JMS 2.0

Riferimenti correlati

“Esempi di come definire la proprietà sharedSubscription” a pagina 470

È inoltre possibile definire la proprietà sharedSubscription di una specifica di attivazione all'interno di un file WebSphere Liberty `server.xml` . In alternativa, è possibile definire la proprietà all'interno di un MDB (message driven bean) utilizzando le annotazioni.

CLONESUPP

proprietà SupportMQExtensions

La specifica JMS 2.0 ha introdotto modifiche al modo in cui funzionano determinati comportamenti. IBM MQ 8.0 e versioni successive include la proprietà **`com.ibm.mq.jms.SupportMQExtensions`**, che può essere impostata su TRUE per ripristinare questi comportamenti modificati alle implementazioni precedenti.

V 9.3.0 JM 3.0 V 9.3.0 La proprietà **`com.ibm.mq.jakarta.jms.SupportMQExtensions`** (Jakarta Messaging 3.0) è supportata da IBM MQ classes for Jakarta Messaging, disponibili in `com.ibm.mq.jakarta.client.jar`.

JMS 2.0 La proprietà **`com.ibm.mq.jms.SupportMQExtensions`** (JMS 2.0) è supportata da IBM MQ classes for JMS, disponibili in `com.ibm.mq.allclient.jar` o `com.ibm.mqjms.jar`.

Tre aree di funzionalità vengono ripristinate impostando **`SupportMQExtensions`** su `True`:

Priorità messaggio

Ai messaggi può essere assegnata una priorità compresa tra 0 e 9. Prima di JMS 2.0, i messaggi potevano anche utilizzare il valore -1, che indica che viene utilizzata la priorità predefinita di una

coda. JMS 2.0 e versioni successive non consentono l'impostazione di una priorità di messaggio di -1. L'attivazione di **SupportMQExtensions** consente di utilizzare il valore -1.

ID client
La specifica JMS 2.0 o successiva richiede che gli ID client non null vengano controllati per l'univocità quando effettuano una connessione. L'attivazione di **SupportMQExtensions** significa che questo requisito viene ignorato e che un ID client può essere riutilizzato.

NoLocal
La specifica JMS 2.0 o successiva richiede che quando questa costante è attivata, un utente non può ricevere messaggi pubblicati dallo stesso ID client. Prima di JMS 2.0, questo attributo era impostato su un sottoscrittore per evitare che ricevesse messaggi pubblicati dalla propria connessione. L'attivazione di **SupportMQExtensions** ripristina questo comportamento alla sua implementazione precedente.

Questa proprietà può essere impostata come segue:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

Questa proprietà può essere impostata come una proprietà di sistema JVM standard nel comando **java** o contenuta nel file di configurazione IBM MQ classes for JMS .

Concetti correlati
“Il file di configurazione IBM MQ classes for JMS/Jakarta Messaging” a pagina 100
I file di configurazione di IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging specificano le proprietà utilizzate per configurare IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Riferimenti correlati
“Proprietà utilizzate per configurare il comportamento del client JMS” a pagina 107
Utilizzare queste proprietà per configurare il comportamento del client JMS .

Utilizzo delle sottoscrizioni condivise nelle applicazioni JMS

Con le sottoscrizioni condivise, una singola sottoscrizione viene condivisa tra più consumer, con solo uno dei consumer che riceve una pubblicazione in qualsiasi momento.

Le sottoscrizioni condivise sono disponibili a partire da JMS 2.0 . Quando si sviluppa un'applicazione JMS per IBM MQ 8.0 o successiva, potrebbe essere necessario considerare l'impatto di questa funzionalità sul gestore code.

L'idea alla base delle sottoscrizioni condivise è quella di condividere il carico tra più consumatori. Una sottoscrizione durevole può anche essere condivisa tra più consumer.

Ad esempio, si supponga che:

- Sottoscrizione SUB, sottoscrizione a un argomento FIFA2014/UPDATES per ricevere gli aggiornamenti delle partite di calcio, condiviso da tre utenti C1, C2e C3
- Pubblicazione del produttore P1 sull'argomento FIFA2014/UPDATES

Quando viene effettuata una pubblicazione su FIFA2014/UPDATES, la pubblicazione verrà ricevuta solo da uno dei tre consumer (C1, o C2, o C3) ma non da tutti.

Il seguente esempio illustra l'utilizzo delle sottoscrizioni condivise e l'uso dell'API aggiuntiva in JMS 2.0, `Message.receiveBody()`, per richiamare solo il corpo del messaggio.

L'esempio crea tre thread del sottoscrittore, che creano una sottoscrizione condivisa per l'argomento FIFA2014/UPDATES e un thread del publisher.

V 9.3.0 JM 3.0 V 9.3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;
```

```
import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
```

```
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
```

```
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

Concetti correlati

Interfacce di lingua IBM MQ Java

V 9.3.2 Configurazione della tua applicazione modulare per utilizzare IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging

V 9.3.2 È possibile utilizzare IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging in modo modulare, richiedendo il modulo appropriato all'interno dell'applicazione e includendo la directory appropriata nel percorso del modulo.

Il packaging modulare

I file JAR unificati per IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging forniscono nomi modulo automatici, che sostituiscono i nomi predefiniti derivati dai nomi file JAR.

- IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) sono forniti con un nome modulo `com.ibm.mq.javax`.
- IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) sono forniti con un nome modulo `com.ibm.mq.jakarta`.

La directory `MQ_HOME/java/lib` predefinita non è adatta per l'uso modulare poiché i moduli non possono contenere lo stesso package e la directory predefinita contiene gli stessi package in più JAR. Pertanto, sono disponibili nuove directory che contengono solo i file JAR necessari, senza duplicazione dei package tra i JAR. Queste directory sono adatte per l'inclusione in un `module-path`.

Nota: Se si dispone di applicazioni che utilizzano i file JAR disponibili in un contesto modulare utilizzando i nomi modulo predefiniti, è necessario aggiornare le proprie applicazioni per richiedere i nuovi nomi modulo. I nomi modulo predefiniti derivano dai nomi file JAR.

Configurazione della tua applicazione modulare per utilizzare IBM MQ classes for JMS

È possibile configurare la propria applicazione modulare per utilizzare IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) completando la seguente procedura:

- Configurare l'applicazione per richiedere il modulo `com.ibm.mq.javax` .
- Configurare l'applicazione per includere la directory `MQ_HOME/java/lib/modules/javax` nel percorso del modulo.

Configurazione della tua applicazione modulare per utilizzare IBM MQ classes for Jakarta Messaging

È possibile configurare la propria applicazione modulare per utilizzare IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) completando la seguente procedura:

- Configurare l'applicazione per richiedere il modulo `com.ibm.mq.jakarta` .
- Configurare l'applicazione per includere la directory `MQ_HOME/java/lib/modules/jakarta` nel percorso del modulo.

Configurazione della tua applicazione modulare per utilizzare IBM MQ classes for Java

Per utilizzare IBM MQ classes for Java da un'applicazione modulare, è possibile utilizzare la configurazione per IBM MQ classes for JMS o la configurazione per IBM MQ classes for Jakarta Messaging, poiché entrambi i file JAR del client supportano IBM MQ classes for Java. Tuttavia, l'applicazione deve utilizzare solo una di queste configurazioni, non entrambe.

Funzioni di IBM MQ classes for JMS Application Server

Questo argomento descrive come IBM MQ classes for JMS implementa la classe ConnectionConsumer e la funzionalità avanzata nella classe Session. Riepiloga anche la funzione di un pool di sessioni server.

Importante: Queste informazioni sono fornite solo come riferimento. Un'applicazione non deve essere scritta per utilizzare questa interfaccia: viene utilizzata nell'adattatore di risorse IBM MQ per connettersi ai server Java EE. Per informazioni pratiche sulla connessione, consultare “Utilizzo dell'adattatore di risorse IBM MQ” a pagina 437.

IBM MQ classes for JMS supporta le ASF (Application Server Facilities) specificate nella *Specifica Java Message Service* (consultare Oracle Technology Network for Java Developers). Questa specifica identifica tre ruoli all'interno di questo modello di programmazione:

- **Il fornitore JMS** fornisce ConnectionConsumer e la funzionalità di sessione avanzata.
- **Il server delle applicazioni** fornisce la funzionalità ServerSessionPool e ServerSession .
- **L'applicazione client** utilizza la funzione fornita dal provider JMS e dal server delle applicazioni.

Le informazioni contenute in questo argomento non si applicano se un'applicazione utilizza una connessione in tempo reale ad un broker.

JMS ConnectionConsumer

L'interfaccia ConnectionConsumer fornisce un metodo ad alte prestazioni per consegnare i messaggi contemporaneamente a un pool di thread.

La specifica JMS consente a un application server di integrarsi strettamente con un'implementazione JMS utilizzando l'interfaccia `ConnectionConsumer` . Questa funzione fornisce l'elaborazione simultanea dei messaggi. Di solito, un server delle applicazioni crea un lotto di thread e l'implementazione JMS rende i messaggi disponibili per tali thread. Un server delle applicazioni che riconosce JMS(come WebSphere Application Server) può utilizzare questa funzione per fornire funzionalità di messaggistica di alto livello, come i bean basati sui messaggi.

Le applicazioni normali non utilizzano ConnectionConsumer, ma i client JMS esperti potrebbero utilizzarlo. Per tali client, ConnectionConsumer fornisce un metodo ad alte prestazioni per consegnare i messaggi contemporaneamente a un lotto di thread. Quando un messaggio arriva su una coda o su un argomento, JMS seleziona un thread dal lotto e gli consegna un batch di messaggi. A tale scopo, JMS esegue un metodo MessageListenerassociato `onMessage()` .

È possibile ottenere lo stesso risultato creando più oggetti Session e MessageConsumer , ciascuno con un MessageListenerregistrato. Tuttavia, ConnectionConsumer fornisce prestazioni migliori, un minore utilizzo delle risorse e una maggiore flessibilità. In particolare, è richiesto un minor numero di oggetti Session.

Pianificazione di un'applicazione con ASF

In questa sezione viene descritto come pianificare un'applicazione che include:

- “Principi generali per la messaggistica point - to - point utilizzando ASF” a pagina 337
- “Principi generali per la messaggistica di pubblicazione / sottoscrizione utilizzando ASF” a pagina 338
- “Rimozione dei messaggi dalla coda in ASF” a pagina 338
- Gestione dei messaggi dannosi in ASF. Consultare “Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236.

Principi generali per la messaggistica point - to - point utilizzando ASF
Utilizzare questo argomento per informazioni generali sulla messaggistica point - to - point utilizzando ASF.

Quando un'applicazione crea un ConnectionConsumer da un oggetto QueueConnection , specifica un oggetto coda JMS e una stringa selettore. ConnectionConsumer inizia quindi a fornire messaggi alle sessioni nel pool ServerSessionassociato. I messaggi arrivano sulla coda e, se corrispondono al selettore, vengono consegnati alle sessioni nel pool ServerSessionassociato.

In termini IBM MQ , l'oggetto coda fa riferimento a un QLOCAL o a un QALIAS sul gestore code locale. Se si tratta di un QALIAS, tale QALIAS deve fare riferimento a un QLOCAL. Il IBM MQ QLOCAL completamente risolto è noto come *QLOCAL sottostante*. Un ConnectionConsumer viene definito *attivo* se non è chiuso e il relativo QueueConnection principale è avviato.

È possibile eseguire più ConnectionConsumers, ognuno con selettori differenti, rispetto allo stesso QLOCAL sottostante. Per mantenere le prestazioni, i messaggi indesiderati non devono essere accumulati sulla coda. I messaggi indesiderati sono quelli per cui nessun ConnectionConsumer attivo dispone di un selettore corrispondente. È possibile impostare il factory QueueConnectionin modo che questi messaggi indesiderati vengano rimossi dalla coda (per i dettagli, vedere “Rimozione dei messaggi dalla coda in ASF” a pagina 338). È possibile impostare questo comportamento in due modi:

- Utilizzare lo strumento di gestione JMS per impostare il factory QueueConnectionsu MRET (NO).
- Nel programma, utilizzare:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Se non si modifica questa impostazione, il valore predefinito è di conservare tali messaggi indesiderati nella coda.

Quando si imposta il gestore code IBM MQ , considerare i seguenti punti:

- Il QLOCAL sottostante deve essere abilitato per l'input condiviso. Per eseguire questa operazione, utilizzare il seguente comando MQSC:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- Il gestore code deve avere una coda di messaggi non recapitabili abilitata. Se un ConnectionConsumer riscontra un problema quando inserisce un messaggio nella coda di messaggi non recapitabili, la consegna del messaggio dal QLOCAL sottostante si arresta. Per definire una coda di messaggi non recapitabili, utilizzare:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- L'utente che esegue ConnectionConsumer deve disporre dell'autorità per eseguire MQOPEN con MQOO_SAVE_ALL_CONTEXT e MQOO_PASS_ALL_CONTEXT. Per i dettagli, consultare la documentazione IBM MQ per la piattaforma specifica.
- Se i messaggi indesiderati vengono lasciati nella coda, le prestazioni del sistema vengono ridotte. Pertanto, pianificare i selettori dei messaggi in modo che tra di essi, ConnectionConsumers rimuoverà tutti i messaggi dalla coda.

Per dettagli sui comandi MQSC, consultare Comandi MQSC.

Principi generali per la messaggistica di pubblicazione / sottoscrizione utilizzando ASF
ConnectionConsumers riceve messaggi per un argomento specificato. Un ConnectionConsumer può essere durevole o non durevole. È necessario specificare la coda o le code utilizzate da ConnectionConsumer .

Quando un'applicazione crea un ConnectionConsumer da un oggetto TopicConnection , specifica un oggetto Topic e una stringa selettore. Il ConnectionConsumer inizia quindi a ricevere i messaggi che corrispondono al selettore su tale argomento, incluse le pubblicazioni conservate per l'argomento sottoscritto.

In alternativa, un'applicazione può creare un ConnectionConsumer durevole associato a un nome specifico. Questo ConnectionConsumer riceve i messaggi che sono stati pubblicati sull'argomento dall'ultima volta che ConnectionConsumer è stato attivo. Riceve tutti questi messaggi che corrispondono al selettore sull'argomento. Tuttavia, se ConnectionConsumer sta utilizzando la lettura anticipata, può perdere i messaggi non persistenti che si trovano nel buffer del client quando viene chiuso.

Se IBM MQ classes for JMS si trova in modalità di migrazione del provider di messaggistica IBM MQ , viene utilizzata una coda separata per le sottoscrizioni ConnectionConsumer non durevoli.L'opzione configurabile CCSUB sul factory TopicConnectionspecifica la coda da utilizzare. Normalmente, CCSUB specifica una singola coda per l'utilizzo da parte di tutti i ConnectionConsumers che utilizzano lo stesso factory TopicConnection. Tuttavia, è possibile fare in modo che ogni ConnectionConsumer generi una coda temporanea specificando un prefisso del nome della coda seguito da un asterisco (*).

Se IBM MQ classes for JMS si trova in modalità di migrazione del provider di messaggistica IBM MQ , la proprietà CCDSUB dell'argomento specifica la coda da utilizzare per le sottoscrizioni durevoli. Di nuovo, questa può essere una coda già esistente o un prefisso del nome della coda seguito da un asterisco (*). Se si specifica una coda già esistente, tutti i ConnectionConsumers durevoli che sottoscrivono l'argomento utilizzano questa coda. Se si specifica un prefisso del nome della coda seguito da un asterisco (*), viene generata una coda la prima volta che un ConnectionConsumer durevole viene creato con un nome particolare. Questa coda viene riutilizzata successivamente quando viene creato un ConnectionConsumer durevole con lo stesso nome.

Quando si imposta il gestore code IBM MQ , considerare i seguenti punti:

- Il gestore code deve avere una coda di messaggi non recapitabili abilitata. Se un ConnectionConsumer riscontra un problema quando inserisce un messaggio nella coda di messaggi non recapitabili, la consegna del messaggio dal QLOCAL sottostante si arresta. Per definire una coda di messaggi non recapitabili, utilizzare:

  ```
  ALTER QMGR DEADQ( your.dead.letter.queue.name )
  ```

- L'utente che esegue ConnectionConsumer deve disporre dell'autorità per eseguire MQOPEN con MQOO_SAVE_ALL_CONTEXT e MQOO_PASS_ALL_CONTEXT. Per i dettagli, consultare la documentazione IBM MQ per la propria piattaforma.
- È possibile ottimizzare le prestazioni per un singolo ConnectionConsumer creando una coda dedicata separata. Questo è al costo di un ulteriore utilizzo delle risorse.

Rimozione dei messaggi dalla coda in ASF
Quando un'applicazione utilizza ConnectionConsumers, JMS potrebbe dover rimuovere i messaggi dalla coda in diverse situazioni.

Queste situazioni sono le seguenti:

Messaggio formattato in modo non corretto
Potrebbe arrivare un messaggio che JMS non può analizzare.

Messaggio dannoso
Un messaggio potrebbe raggiungere la soglia di backout, ma ConnectionConsumer non riesce a riaccodare il messaggio sulla coda di backout.

Nessun ConnectionConsumer interessato
Per la messaggistica point - to - point, quando QueueConnectionFactory è impostato in modo da non conservare i messaggi indesiderati, arriva un messaggio indesiderato da parte di ConnectionConsumers.

In queste situazioni, ConnectionConsumer tenta di eliminare il messaggio dalla coda. Le opzioni di disposizione nel campo report di MQMD del messaggio impostano il comportamento esatto. Queste opzioni sono:

MQRO_DEAD_LETTER_Q
Il messaggio viene riaccodato alla coda di messaggi non instradabili del gestore code. Questa è l'opzione predefinita.

MQRO_DISCARD_MSG
Il messaggio viene eliminato.

Anche ConnectionConsumer genera un messaggio di report, che dipende anche dal campo del report MQMD del messaggio. Questo messaggio viene inviato alla coda ReplyTodel messaggio sul gestore code ReplyTo. Se si verifica un errore durante l'invio del messaggio di report, il messaggio viene inviato alla coda di messaggi non recapitabili. Le opzioni del report di eccezione nel campo del report dei dettagli della serie MQMD del messaggio del report. Queste opzioni sono:

MQRO_ECCEZIONE
Viene generato un messaggio di report che contiene l'MQMD del messaggio originale. Non contiene dati del corpo del messaggio.

MQRO_EXCEPTION_WITH_DATA
Viene generato un messaggio di report che contiene MQMD, tutte le intestazioni MQ e 100 byte di dati del corpo.

MQRO_EXCEPTION_WITH_FULL_DATA
Viene generato un messaggio di report che contiene tutti i dati del messaggio originale.

valore predefinito
Non viene generato alcun messaggio di prospetto.

Quando vengono generati messaggi di report, vengono rispettate le opzioni riportate di seguito:

- ID_MSG_NEW_MQRO
- MQRO_PASS_MSG_ID
- ID_COPY_MQRO_MSG_TO_CORREL_ID
- ID CORREL_PASS_MQRO_

Se un messaggio non elaborabile non può essere riaccodato, forse perché la coda di messaggi non recapitabili è piena o l'autorizzazione è specificata in modo non corretto, ciò che accade dipende dalla persistenza del messaggio. Se il messaggio non è persistente, viene eliminato e non viene generato alcun messaggio di report. Se il messaggio è persistente, la consegna dei messaggi a tutti i consumer di connessione in ascolto su tale destinazione si arresta. Tali utenti della connessione devono essere chiusi e il problema risolto prima che possano essere ricreati e la consegna del messaggio riavviata.

È importante definire una coda di messaggi non recapitabili e controllarla regolarmente per assicurarsi che non si verifichino problemi. In particolare, verificare che la coda di messaggi non recapitabili non raggiunga la profondità massima e che la dimensione massima del messaggio sia sufficiente per tutti i messaggi.

Quando un messaggio viene riaccodato alla coda di messaggi non recapitabili, viene preceduto da un'intestazione di messaggi non recapitabili (MQDLH) IBM MQ . Consultare MQDLH - Dead - letter header per i dettagli sul formato di MQDLH. È possibile identificare i messaggi che un ConnectionConsumer ha inserito nella coda di messaggi non recapitabili o segnalare i messaggi generati da un ConnectionConsumer dai seguenti campi:

- PutApplIl tipo è `MQAT_JAVA (0x1C)`
- PutApplIl nome è "`MQ JMS ConnectionConsumer`"

Questi campi si trovano nell'MQDLH dei messaggi nella coda di messaggi non instradabili e nell'MQMD dei messaggi di report. Il campo feedback di MQMD e il campo Motivo di MQDLH contengono un codice che descrive l'errore. Per dettagli su questi codici, consultare “Codici di errore e feedback in ASF” a pagina 341. Altri campi sono quelli descritti in MQDLH - Dead - letter header.

Gestione dei messaggi non elaborabili in ASF
All'interno delle funzioni del server delle applicazioni, la gestione dei messaggi non elaborabili viene gestita in modo leggermente diverso rispetto a quanto avviene altrove in IBM MQ classes for JMS.

Per informazioni sulla gestione dei messaggi non elaborabili in IBM MQ classes for JMS, consultare “Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236.

Quando si utilizzano le ASF (Application Server Facilities), ConnectionConsumer, invece di MessageConsumer, elabora messaggi non elaborabili. Il ConnectionConsumer riaccoda i messaggi in base alle proprietà QName BackoutThreshold e BackoutRequeuedella coda.

Quando un'applicazione utilizza ConnectionConsumers, le circostanze in cui viene eseguito il backout di un messaggio dipendono dalla sessione fornita dal server delle applicazioni:

- Quando la sessione non è sottoposta a transazione, con AUTO_RICONOSCIUTO_AUTOMATICO o DUPS_OK_RICONOSCIMENTO, viene eseguito il backout di un messaggio solo dopo un errore di sistema o se l'applicazione termina in modo imprevisto.
- Quando la sessione non è sottoposta a transazioni con CLIENT_ACKNOWLEDGED, i messaggi non riconosciuti possono essere sottoposti a backout dal server delle applicazioni che richiama Session.recover().

  Di solito, l'implementazione client di MessageListener o il server delle applicazioni richiama Message.acknowledge(). Message.acknowledge() riconosce tutti i messaggi consegnati nella sessione fino a quel momento.
- Quando la sessione viene eseguita, i messaggi non riconosciuti possono essere ripristinati dal server delle applicazioni che richiama Session.rollback().
- Se il server delle applicazioni fornisce una XASession, i messaggi vengono sottoposti a commit o a backout in base a una transazione distribuita. Il server delle applicazioni si assume la responsabilità del completamento della transazione.

Concetti correlati
“Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236
Un messaggio non elaborabile è quello che non può essere elaborato da un'applicazione ricevente. Se un messaggio non elaborabile viene consegnato a un'applicazione e ne viene eseguito il rollback un numero specificato di volte, IBM MQ classes for JMS può spostarlo in una coda di backout.

Gestione degli errori

Questa sezione descrive vari aspetti della gestione degli errori, inclusi “Ripristino da condizioni di errore in ASF” a pagina 340 e “Codici di errore e feedback in ASF” a pagina 341.

Ripristino da condizioni di errore in ASF
Se un ConnectionConsumer rileva un errore grave, la consegna del messaggio a tutti i ConnectionConsumers con un interesse nello stesso QLOCAL si arresta. Quando ciò si verifica, viene notificato qualsiasi ExceptionListener registrato con la connessione interessata. Esistono due modi in cui un'applicazione può eseguire il ripristino da queste condizioni di errore.

Di solito, un errore grave di questa natura si verifica se ConnectionConsumer non può riaccodare un messaggio alla coda di messaggi non recapitabili o se si verifica un errore durante la lettura dei messaggi da QLOCAL.

Poiché ogni ExceptionListener registrato con la connessione interessata viene notificato, è possibile utilizzarli per identificare la causa del problema. In alcuni casi, l'amministratore di sistema deve intervenire per risolvere il problema.

Utilizzare una delle tecniche riportate di seguito per eseguire il ripristino da queste condizioni di errore:

- Chiamare `close()` su tutti gli ConnectionConsumersinteressati. L'applicazione può creare nuovi ConnectionConsumers solo dopo che tutti i ConnectionConsumers interessati sono stati chiusi e gli eventuali problemi di sistema sono stati risolti.
- Richiamare `stop()` su tutte le connessioni interessate. Dopo che tutte le connessioni sono state arrestate e gli eventuali problemi di sistema sono stati risolti, l'applicazione può `start()` eseguire correttamente le connessioni.

Codici di errore e feedback in ASF
Utilizzare i codici di errore e di feedback per determinare la causa di un errore. Di seguito vengono forniti i codici di errore comuni generati da ConnectionConsumer .

Per determinare la causa di un errore, utilizzare le seguenti informazioni:

- Il codice di feedback nei messaggi di report
- Il codice di errore in MQDLH di tutti i messaggi nella coda di messaggi non instradabili

ConnectionConsumers genera i seguenti codici di errore.

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Causa
Il messaggio ha raggiunto la soglia di backout definita su QLOCAL, ma non viene definita alcuna coda di backout.

Sulle piattaforme in cui non è possibile definire la coda di backout, il messaggio ha raggiunto la soglia di backout definita da JMSdi 20.

Azione
Se non si desidera, definire la coda di backout per il QLOCAL pertinente. Cercare anche la causa dei più backout.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Causa
Nella messaggistica point - to - point, è presente un messaggio che non corrisponde ad alcuno dei selettori per ConnectionConsumers che controllano la coda. Per mantenere le prestazioni, il messaggio viene riaccodato alla coda di messaggi non recapitabili.

Azione
Per evitare questa situazione, assicurarsi che ConnectionConsumers che utilizza la coda fornisca una serie di selettori che trattano tutti i messaggi oppure impostare il factory QueueConnectionper conservare i messaggi.

In alternativa, esaminare l'origine del messaggio.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Causa
JMS non può interpretare il messaggio sulla coda.

Azione
Esaminare l'origine del messaggio. JMS normalmente consegna i messaggi di un formato non previsto come BytesMessage o TextMessage. Di tanto in tanto, questa operazione non riesce se il messaggio è formattato in modo non corretto.

Altri codici visualizzati in questi campi sono causati da un tentativo non riuscito di riaccodare il messaggio a una coda di backout. In questa situazione, il codice descrive il motivo per cui la riaccodamento non è riuscito. Per diagnosticare la causa di questi errori, fare riferimento a codici di errore e completamento API API>.

Se il messaggio di report non può essere inserito nella coda ReplyTo, viene inserito nella coda di messaggi non recapitabili. In questa situazione, il campo di feedback di MQMD viene completato come descritto in questo argomento. Il campo motivo in MQDLH spiega perché non è stato possibile inserire il messaggio di report nella coda ReplyTo.

La funzione di un pool di sessioni server in AFS

Questo argomento riepiloga la funzione di un pool di sessioni server.

Figura 45 a pagina 342 riepiloga i principi della funzionalità ServerSessionPool e ServerSession .

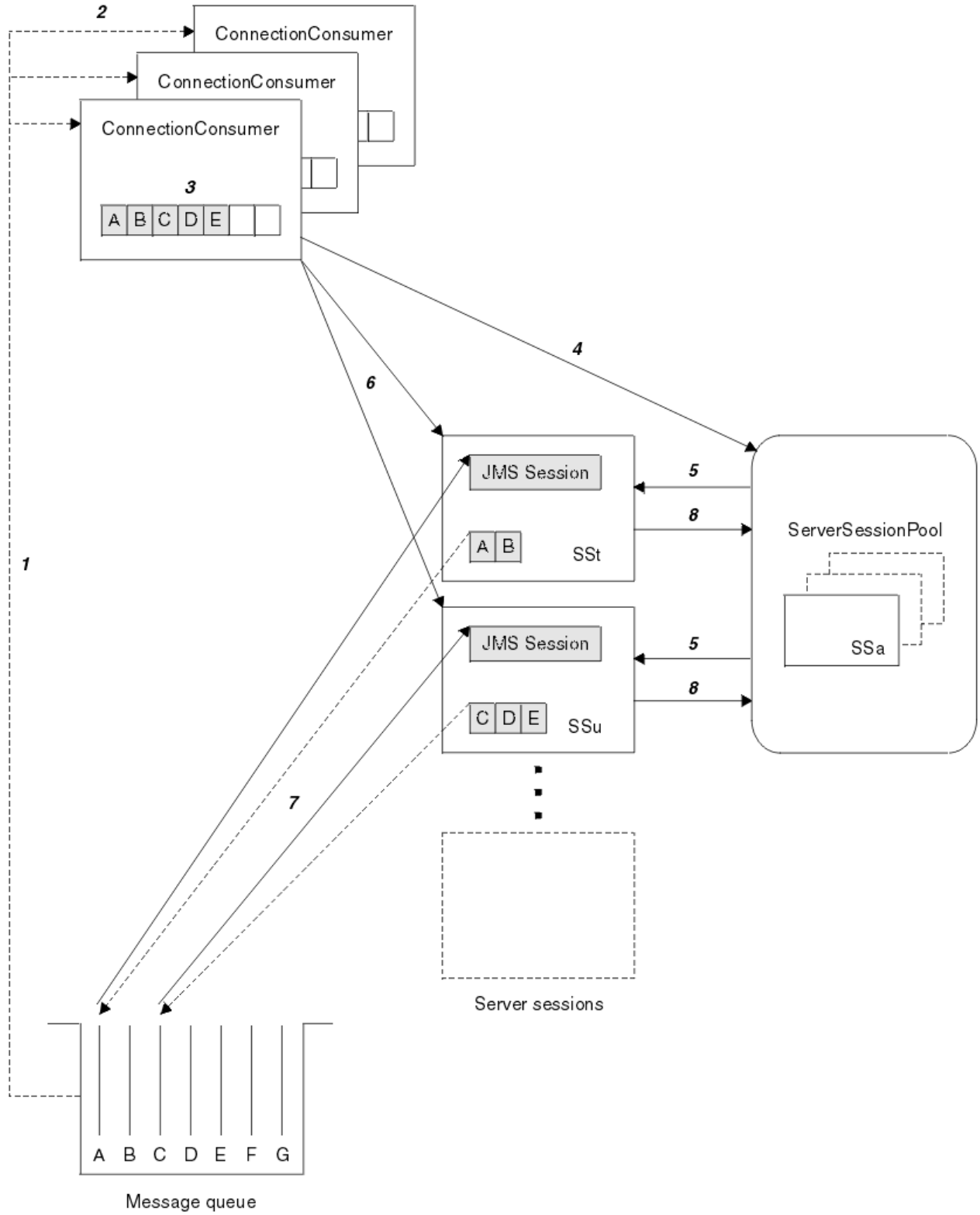


Figura 45. Funzionalità ServerSessionPool e ServerSession

1. I ConnectionConsumers ricevono i riferimenti ai messaggi dalla coda.

2. Ogni ConnectionConsumer seleziona riferimenti di messaggi specifici.
3. Il buffer ConnectionConsumer contiene i riferimenti ai messaggi selezionati.
4. ConnectionConsumer richiede una o più ServerSessions dal pool ServerSession.
5. Le ServerSessions vengono assegnate dal pool ServerSession.
6. ConnectionConsumer assegna i riferimenti dei messaggi a ServerSessions e avvia l'esecuzione dei thread ServerSession .
7. Ogni ServerSession richiama i relativi messaggi di riferimento dalla coda. Li passa al metodo onMessage dal MessageListener associato alla sessione JMS .
8. Una volta completata l'elaborazione, ServerSession viene restituito al pool.

Un server delle applicazioni normalmente fornisce la funzione ServerSessionPool e ServerSession .

z/OS Utilizzo di IBM MQ classes for JMS in un server JVM CICS Liberty

Da IBM MQ 9.1.0, i programmi Java in esecuzione in un server JVM CICS Liberty possono utilizzare IBM MQ classes for JMS per accedere a IBM MQ.

È necessario utilizzare una versione IBM MQ 9.1.0o successiva dell'adattatore di risorse IBM MQ . È possibile ottenere l'adattatore di risorse da Fix Central (vedere “Installazione dell'adattatore di risorse in Liberty” a pagina 447).

Ci sono due tipi di JVM del profilo Liberty disponibili in CICS 5.3 e successive, i tipi di connessione possibili per IBM MQ sono limitati come segue:

CICS Liberty Standard

- L'adattatore di risorse IBM MQ può connettersi a qualsiasi versione in servizio di IBM MQ in modalità CLIENT
- L'adattatore di risorse IBM MQ può connettersi a qualsiasi versione in servizio di IBM MQ for z/OS in modalità BINDINGS quando non è presente alcuna connessione CICS (definizione della risorsa MQCONN CICS attiva) allo stesso gestore code dalla stessa regione CICS .

CICS Liberty Integrato

- L'adattatore di risorse IBM MQ può collegarsi a qualsiasi versione in servizio di IBM MQ in modalità CLIENT.
- La connessione in modalità BINDINGS non è supportata.

Per i dettagli sull'impostazione e la configurazione del sistema, consultare Utilizzo di IBM MQ classes for JMS in un server JVM Liberty nella documentazione CICS .

Utilizzo di IBM MQ classes for JMS/ Jakarta Messaging in IMS

Il supporto della messaggistica basata sugli standard in un ambiente IMS viene fornito tramite l'utilizzo di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Controllare i requisiti di sistema per il sistema IMS utilizzato dall'azienda. Per ulteriori informazioni, vedere IMS 15.2 .

Questa serie di argomenti descrive come impostare IBM MQ classes for JMS in un ambiente IMS e le limitazioni API che si applicano quando si utilizzano le interfacce classiche (JMS 1.1) e semplificate (JMS 2.0). Per un elenco delle informazioni specifiche dell'API, consultare “Limitazioni API JMS” a pagina 348 .

Nota: Limitazioni simili si applicano alle interfacce legacy (JMS 1.0.2) specifiche del dominio, ma non sono descritte in modo specifico.

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, consultare Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging.

Regioni dipendenti IMS supportate

Sono supportati i seguenti tipi di region dipendenti:

- MPR
- BMP
- IFP
- JVM (JMP 31 and 64 bit Java virtual machines)
- JVM a 31 e 64 bit JBP

A meno che non venga specificamente indicato nei seguenti argomenti, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging si comportano allo stesso modo in tutti i tipi di region.

Macchine virtuali Java supportate

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging richiedono IBM Runtime Environment, Java Technology Edition 8. IBM Semeru Runtime Certified Edition per z/OS, Versione 11 non è supportato.

Altre restrizioni

Le seguenti limitazioni si applicano quando si utilizza IBM MQ classes for JMS in un ambiente IMS :

- Le connessioni in modalità client non sono supportate.
- Le connessioni sono supportate solo per i gestori code IBM MQ 8.0 che utilizzano la modalità IBM MQ provider di messaggistica `Normal`.

  L'attributo **`PROVIDERVERSION`** sul factory di connessione deve essere non specificato o un valore maggiore o uguale a sette.
- L'utilizzo di qualsiasi factory di connessione XA, ad esempio `com.ibm.mq.jms.MQXAConnectionFactory`, non è supportato.

Attività correlate

Definizione di IBM MQ in IMS

Impostazione dell'adattatore IMS per l'utilizzo con IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging utilizzano lo stesso adattatore IBM MQ-IMS utilizzato da altri linguaggi di programmazione. Questo adattatore utilizza ESAF (External Subsystem Attach Facility) IMS .

Prima di iniziare

Prima di completare la seguente procedura, è necessario configurare l'adattatore IMS per i gestori code rilevanti e il controllo IMS e le aree dipendenti, come descritto in Impostazione dell'adattatore IMS.

Attenzione: Non è necessario eseguire il passo che descrive la generazione di uno stub dinamico, a meno che lo stub dinamico non sia necessario per altri scopi.

Una volta configurato l'adattatore IMS , attenersi alla seguente procedura.

Procedura

1. Aggiornare la variabile LIBPATH nel membro di IMS PROCLIB cui fa riferimento il parametro ENVIRON nel JCL della regione dipendente (ad esempio, DFSJVMEV) in modo che includa le librerie native IBM MQ classes for JMS .

Ovvero, la directory zFS che contiene `libmqjims.so`. Ad esempio, DFSJVMEV potrebbe essere simile al seguente, dove l'ultima riga è la directory contenente le librerie native IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging :

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. Aggiungere IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging al percorso classe della JVM, utilizzato dalla regione dipendente IMS , aggiornando l'opzione `java.class.path` .

   Eseguire questa operazione seguendo le istruzioni riportate in Membro DFSJVMMS del dataset IMS PROCLIB.

   Ad esempio, è possibile utilizzare quanto segue, dove la riga in grassetto indica l'aggiornamento:

V 9.3.0 JM 3.0 V 9.3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   Nota: Mentre ci sono molti file jar differenti disponibili nella directory contenente IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, è necessario solo `com.ibm.mq.allclient.jar` (JMS 2.0) o `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0).

3. Arrestare e riavviare le regioni dipendenti da IMS che utilizzeranno IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Operazioni successive

Creare e configurare factory di connessione e destinazioni.

Esistono tre possibili approcci per la creazione di istanze delle implementazioni IBM MQ di factory di connessione e destinazioni. Vedi “Creazione e configurazione di factory di connessione e destinazioni” a pagina 207 per i dettagli.

Notare che questi tre approcci sono tutti validi in un ambiente IMS .

Concetti correlati

Impostazione dell'adattatore IMS

Definizione di IBM MQ in IMS

Comportamento transazionale

I messaggi inviati e ricevuti da IBM MQ classes for JMS in un ambiente IMS sono sempre associati all'unità di lavoro (UOW) IMS attiva sull'attività corrente.

Tale UOW può essere completata solo richiamando i metodi di commit o di rollback su un'istanza dell'oggetto `com.ibm.ims.dli.tm.Transaction` o dall'attività IMS che termina normalmente, nel qual caso l'UOW viene implicitamente sottoposta a commit. Se l'attività IMS termina in modo anomalo, viene eseguito il rollback della UOW.

Di conseguenza, i valori degli argomenti **`transacted`** e **`acknowledgeMode`** vengono ignorati durante il richiamo di uno dei metodi `Connection.createSession` o `ConnectionFactory.createContext`. Inoltre, non sono supportati i seguenti metodi. Il richiamo di uno dei seguenti metodi comporta un `IllegalStateException` nel caso della sessione:

- `javax.jms.Session.commit()`

- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

e una `IllegalStateRuntimeSession` nel caso di contesto JMS:

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

Esiste un'eccezione a questo comportamento. Se una sessione o un contesto JMS viene creato utilizzando uno dei seguenti meccanismi:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

il funzionamento di tale sessione o contesto JMS è il seguente:

- I messaggi inviati vengono inviati al di fuori della UOW IMS . Cioè, saranno disponibili sulla destinazione di destinazione immediatamente o quando l'intervallo di ritardo di consegna fornito è stato completato.
- Tutti i messaggi non persistenti verranno ricevuti al di fuori della IMS UOW, a meno che la proprietà SYNCPOINTALLGETS non sia stata specificata sulla factory di connessione che ha creato la sessione o il contesto JMS .
- I messaggi persistenti verranno sempre ricevuti all'interno della UOW IMS .

Ciò potrebbe essere utile se, ad esempio, si desidera scrivere un messaggio di verifica in una coda anche se la UOW esegue il rollback.

Implicazioni dei punti di sincronizzazione IMS

Il IBM MQ classes for JMS si basa sul supporto dell'adattatore IBM MQ esistente che utilizza ESAF. Ciò significa che si applica il comportamento documentato, inclusi tutti gli handle aperti chiusi dall'adattatore IMS quando si verifica un punto di sincronizzazione.

Per ulteriori informazioni, consultare “Punti di sincronizzazione nelle applicazioni IMS” a pagina 72.

Per illustrare questo punto, considerare il seguente codice in esecuzione in un ambiente JMP. La seconda chiamata a `mp.send()` risulta in un `JMSException` poiché il codice `messageQueue.getUnique(inputMessage)` risulta in tutte le connessioni aperte IBM MQ e gli handle di oggetto vengono chiusi.

Un comportamento simile viene osservato se la chiamata `getUnique()` è stata sostituita con `Transaction.commit()`, ma non se è stato utilizzato `Transaction.rollback()` .

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

Il codice corretto da utilizzare in questo scenario è il seguente. In questo caso la connessione a IBM MQ viene chiusa prima di richiamare `getUnique()`. La connessione e la sessione vengono quindi ricreate per inviare un altro messaggio.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

Considerazioni sull'utilizzo dell'adattatore IMS

È necessario essere consapevoli delle seguenti restrizioni. È possibile disporre di un solo handle di connessione per ciascun gestore code. Ci sono implicazioni nell'interazione con IBM MQ quando si utilizza sia il JMS che il codice nativo. Esistono limitazioni all'autenticazione e all'autorizzazione della connessione.

Un handle di connessione per ciascun gestore code

Solo un handle di connessione alla volta per un gestore code specifico è consentito nelle aree dipendenti IMS . Qualsiasi tentativo successivo di connessione allo stesso gestore code riutilizza l'handle esistente.

Mentre questo comportamento non dovrebbe causare problemi in un'applicazione che utilizza solo IBM MQ classes for JMS, questo comportamento può causare problemi nelle applicazioni che interagiscono con IBM MQ, quando si utilizzano sia IBM MQ classes for JMS che MQI nel codice nativo scritto in linguaggi, come COBOL o C.

Implicazioni dell'interagire con IBM MQ quando si utilizza sia il codice JMS che il codice nativo

I problemi possono verificarsi quando si lascia il codice Java e il codice nativo che utilizzano la funzionalità IBM MQ e quando la connessione a IBM MQ non viene chiusa prima di lasciare il codice nativo o Java.

Ad esempio, nel seguente pseudo codice, un handle di connessione a un gestore code viene originariamente stabilito in codice Java utilizzando IBM MQ classes for JMS. L'handle di connessione viene riutilizzato in codice COBOL e invalidato da una chiamata a MQDISC.

La volta successiva in cui IBM MQ classes for JMS utilizza la gestione della connessione `JMSException` con un codice motivo di risultati MQRC_HCONN_ERROR.

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
       Get session from static variable
       Create a message consumer - fails as connection handle invalidated
```

Esistono altri modelli di utilizzo simili che possono risultare in MQRC_HCONN_ERROR.

Sebbene sia possibile condividere gli handle di connessione IBM MQ tra il codice nativo e Java (ad esempio, l'esempio precedente funzionerebbe se non vi fosse stata una chiamata MQDISC) in generale, la procedura ottimale consiste nel chiudere tutti gli handle di connessione prima di passare da Java al codice nativo o viceversa.

Autenticazione e autorizzazione della connessione

La specifica JMS consente di specificare un nome utente e una password per l'autenticazione e l'autorizzazione durante la creazione di una connessione o di un oggetto di contesto JMS .

Ciò non è supportato in un ambiente IMS . Il tentativo di creazione di un collegamento durante la specifica di un nome utente e di una password comporta l'emissione di `JMS Exception` . Il tentativo di creare un contesto JMS , durante la specifica di nome utente e password, comporta l'emissione di un `JMSRuntimeException` .

Al contrario, è necessario utilizzare i meccanismi esistenti per l'autenticazione e l'autorizzazione durante la connessione a IBM MQ da un ambiente IMS .

Per ulteriori informazioni, consultare Configurazione della sicurezza su z/OS. Nello specifico, fare riferimento a ID utente per il controllo di sicurezza, che descrive gli ID utente che possono essere utilizzati.

Attività correlate

Impostazione della sicurezza su z/OS

Limitazioni API JMS

Da una prospettiva di specifica JMS , IBM MQ classes for JMS considera IMS come un server delle applicazioni conforme a Java EE o Jakarta EE , che ha sempre una transazione JTA in corso.

Ad esempio, non è mai possibile richiamare `javax.jms.Session.commit()` in IMS, poiché la specifica JMS indica che non è possibile richiamarlo in un EJB JEE o in un contenitore Web, mentre è in corso una transazione JTA.

Ciò comporta le seguenti limitazioni all'API JMS , oltre a quelle descritte in “Comportamento transazionale” a pagina 345.

Limitazioni API classica

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` genera sempre un `JMSException`.
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` genera sempre un `JMSException`.
- Tutte e tre le varianti di `javax.jms.Connection.createSession` generano sempre una `JMSException` se la connessione ha già una sessione esistente attiva.

- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` genera sempre un `JMSException`.
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` genera sempre un `JMSException`.
- `javax.jms.Connection.setClientID()` genera sempre un `JMSException`.
- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` genera sempre un `JMSException`.
- `javax.jms.Connection.stop()` genera sempre un `JMSException`.
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` genera sempre un `JMSException`.
- `javax.jms.MessageConsumer.getMessageListener()` genera sempre un `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` genera sempre un `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` genera sempre un `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` genera sempre un `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener)` genera sempre un `JMSException`.
- `javax.jms.Session.run()` genera sempre un `JMSRuntimeException`.
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` genera sempre un `JMSException`.
- `javax.jms.Session.getMessageListener()` genera sempre un `JMSException`.

Limitazioni API semplificate

- `javax.jms.JMSContext.createContext(int)` genera sempre un `JMSRuntimeException`.
- `javax.jms.JMSContext.setClientID(String)` genera sempre un `JMSRuntimeException`.
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` genera sempre un `JMSRuntimeException`.
- `javax.jms.JMSContext.stop()` genera sempre un `JMSRuntimeException`.
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` genera sempre un `JMSRuntimeException`.

Utilizzo di IBM MQ classes for Java

Utilizzare IBM MQ in un ambiente Java . IBM MQ classes for Java consenti a un'applicazione Java di connettersi a IBM MQ come client IBM MQ o di connettersi direttamente a un gestore code IBM MQ .

Nota:

Stabilized IBM non apporterà ulteriori miglioramenti a IBM MQ classes for Java e saranno funzionalmente stabilizzati al livello fornito in IBM MQ 8.0. Le applicazioni esistenti che utilizzano IBM MQ classes for Java continuano ad essere completamente supportate, ma le nuove funzioni non verranno aggiunte e le richieste di miglioramenti verranno rifiutate. Completamente supportato significa che i difetti verranno corretti insieme a tutte le modifiche richieste dalle modifiche ai requisiti di sistema IBM MQ .

Le IBM MQ classes for Java non sono supportate in IMS.

Le IBM MQ classes for Java non sono supportate in WebSphere Liberty. Non devono essere utilizzati con la funzione di messaggistica IBM MQ Liberty o con il supporto JCA generico. Per ulteriori informazioni, consultare Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

IBM MQ classes for Java è una delle tre API alternative che le applicazioni Java possono utilizzare per accedere alle risorse IBM MQ . Le altre API sono:

- V 9.3.0 JM 3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

Per ulteriori informazioni, consultare “Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88.

Da IBM MQ 9.3, i IBM MQ classes for Java vengono creati con Java 8. L'ambiente runtime Java 8 supporta l'esecuzione di versioni di file di classe precedenti.

IBM MQ classes for Java incapsula l'interfaccia MQI (Message Queue Interface), l'API IBM MQ nativa e utilizza un modello oggetto simile alle interfacce C++ e .NET a IBM MQ.

Le opzioni programmabili consentono a IBM MQ classes for Java di connettersi a IBM MQ in uno dei seguenti modi:

- In modalità client come IBM MQ MQI client utilizzando TCP/IP (Transmission Control Protocol/Internet Protocol)
- In modalità bind, la connessione diretta a IBM MQ utilizzando JNI (Java Native Interface)

Nota: La riconnessione automatica del client non è supportata da IBM MQ classes for Java.

Connessione in modalità client

Un'applicazione IBM MQ classes for Java può connettersi a qualsiasi gestore code supportato utilizzando la modalità client.

Per connettersi a un gestore code in modalità client, un'applicazione IBM MQ classes for Java può essere eseguita sullo stesso sistema su cui è in esecuzione il gestore code o su un sistema differente. In ogni caso IBM MQ classes for Java si connette al gestore code tramite TCP/IP.

Per ulteriori informazioni su come scrivere le applicazioni per utilizzare le connessioni in modalità client, consultare “Modalità di connessione IBM MQ classes for Java” a pagina 375.

Connessione in modalità bind

Quando viene utilizzato in modalità bind, IBM MQ classes for Java utilizza JNI (Java Native Interface) per richiamare direttamente l'API del gestore code esistente, piuttosto che comunicare attraverso una rete. Nella maggior parte degli ambienti, la connessione in modalità di bind fornisce prestazioni migliori per le applicazioni IBM MQ classes for Java rispetto alla connessione in modalità client, evitando il costo della comunicazione TCP/IP.

Le applicazioni che utilizzano IBM MQ classes for Java per connettersi in modalità bind devono essere eseguite sullo stesso sistema del gestore code a cui si stanno collegando.

Java Runtime Environment, utilizzato per eseguire l'applicazione IBM MQ classes for Java , deve essere configurato per caricare le librerie IBM MQ classes for Java ; consultare “IBM MQ classes for Java librerie” a pagina 360 per ulteriori informazioni.

Per ulteriori informazioni su come scrivere le applicazioni per utilizzare le connessioni in modalità bind, consultare “Modalità di connessione IBM MQ classes for Java” a pagina 375.

Concetti correlati

Interfacce di lingua IBM MQ Java

“Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

Attività correlate

Traccia delle applicazioni IBM MQ classes for Java

Perché dovrei utilizzare IBM MQ classes for Java?

Un'applicazione Java può utilizzare IBM MQ classes for Java o IBM MQ classes for JMS per accedere alle risorse IBM MQ .

Nota: V 9.3.0 V 9.3.0 Anche se le applicazioni esistenti che utilizzano IBM MQ classes for Java continuano ad essere completamente supportate, le nuove applicazioni devono utilizzare IBM MQ classes for Jakarta Messaging Le funzioni che sono state recentemente aggiunte a IBM MQ, come il consumo asincrono e la riconnessione automatica, non sono disponibili in IBM MQ classes for Java, ma in IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging. Per ulteriori informazioni, consultare “Perché dovrei utilizzare IBM MQ classes for JMS?” a pagina 86 e “Perché dovrei utilizzare IBM MQ classes for Jakarta Messaging?” a pagina 85.

Nota:

Stabilized IBM non apporterà ulteriori miglioramenti a IBM MQ classes for Java e saranno funzionalmente stabilizzati al livello fornito in IBM MQ 8.0. Le applicazioni esistenti che utilizzano IBM MQ classes for Java continuano ad essere completamente supportate, ma le nuove funzioni non verranno aggiunte e le richieste di miglioramenti verranno rifiutate. Completamente supportato significa che i difetti verranno corretti insieme a tutte le modifiche richieste dalle modifiche ai requisiti di sistema IBM MQ .

Le IBM MQ classes for Java non sono supportate in IMS.

Le IBM MQ classes for Java non sono supportate in WebSphere Liberty. Non devono essere utilizzati con la funzione di messaggistica IBM MQ Liberty o con il supporto JCA generico. Per ulteriori informazioni, consultare Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

Concetti correlati

“Accesso a IBM MQ da Java - Scelta dell'API” a pagina 88
IBM MQ fornisce tre interfacce di linguaggio Java .

Prerequisiti per IBM MQ classes for Java

Per utilizzare IBM MQ classes for Java, sono necessari alcuni prodotti software.

Per informazioni sui prerequisiti per IBM MQ classes for Java, consultare la pagina Web Requisiti di sistema per IBM MQ .

Per sviluppare applicazioni IBM MQ classes for Java , è necessario un JDK (Java Development Kit). I dettagli dei JDK supportati con il proprio sistema operativo sono riportati nelle informazioni Requisiti di sistema per IBM MQ .

Per eseguire applicazioni IBM MQ classes for Java , sono necessari i seguenti componenti software:

- Un gestore code IBM MQ , per le applicazioni che si connettono a un gestore code
- Un JRE (Java Runtime Environment) per ogni sistema su cui si eseguono le applicazioni. Un JRE adatto viene fornito con IBM MQ.
- IBM i Per IBM i, QShell, che è l'opzione 30 del sistema operativo
- z/OS Per z/OS, z/OS UNIX System Services (z/OS UNIX)

Se si richiedono connessioni TLS per utilizzare moduli crittografici certificati FIPS 140-2, è necessario il provider IBM Java JSSE FIPS (IBMJSSEFIPS). Ogni IBM JDK e JRE alla versione 1.4.2 o successiva contiene IBMJSSEFIPS.

È possibile utilizzare gli indirizzi Internet Protocol versione 6 (IPv6) nelle IBM MQ classes for Java applicazioni se IPv6 è supportato dalla propria JVM (Java virtual machine) e dall'implementazione TCP/IP sul proprio sistema operativo.

Esecuzione di applicazioni IBM MQ classes for Java in Java EE

Ci sono alcune limitazioni e considerazioni di progettazione che devono essere prese in considerazione prima di utilizzare IBM MQ classes for Java in Java EE.

IBM MQ classes for Java ha delle limitazioni quando viene utilizzato in un ambiente Java Platform, Enterprise Edition (Java EE). Ci sono anche ulteriori considerazioni da prendere in considerazione quando si progetta, si implementa e si gestisce un'applicazione IBM MQ classes for Java in esecuzione in un ambiente Java EE . Queste limitazioni e considerazioni sono descritte nelle seguenti sezioni.

Limitazioni transazioni JTA

L'unico gestore transazioni supportato per applicazioni che utilizzano IBM MQ classes for Java è IBM MQ stesso. Sebbene un'applicazione sotto il controllo JTA possa utilizzare IBM MQ classes for Java, qualsiasi lavoro eseguito attraverso queste classi non è controllato dalle unità di lavoro JTA. Formano invece unità di lavoro locali separate da quelle gestite dal server delle applicazioni tramite le interfacce JTA. In particolare, qualsiasi rollback della transazione JTA non risulta in un rollback dei messaggi inviati o ricevuti. Questa limitazione si applica alle transazioni gestite dall'applicazione o dal bean e alle transazioni gestite dal contenitore e a tutti i contenitori Java EE . Per eseguire la messaggistica direttamente con IBM MQ all'interno delle transazioni coordinate dal server delle applicazioni, è necessario utilizzare IBM MQ classes for JMS .

Creazione thread

IBM MQ classes for Java crea thread internamente per varie operazioni. Ad esempio, quando si esegue in modalità BINDINGS per richiamare direttamente un gestore code locale, le chiamate vengono effettuate su un thread 'worker' creato internamente da IBM MQ classes for Java. Altri thread possono essere creati internamente, ad esempio per cancellare le connessioni inutilizzate da un pool di connessioni o per rimuovere le sottoscrizioni per le applicazioni di pubblicazione / sottoscrizione terminate.

Alcune applicazioni Java EE (ad esempio quelle in esecuzione nei contenitori EJB e Web) non devono creare nuovi thread. Invece, tutto il lavoro deve essere eseguito sui thread dell'applicazione principale gestiti dal server delle applicazioni. Quando le applicazioni utilizzano IBM MQ classes for Java, il server delle applicazioni potrebbe non essere in grado di distinguere tra il codice dell'applicazione e il codice IBM MQ classes for Java , quindi i thread precedentemente descritti fanno sì che l'applicazione non sia conforme alla specifica del contenitore. IBM MQ classes for JMS non interrompe queste specifiche Java EE e quindi può essere utilizzato.

Limitazioni di sicurezza

Le politiche di sicurezza implementate da un server delle applicazioni potrebbero impedire alcune operazioni intraprese dall'API IBM MQ classes for Java , come la creazione e l'utilizzo di nuovi thread di controllo (come descritto nelle sezioni precedenti).

Ad esempio, i server delle applicazioni generalmente vengono eseguiti con la sicurezza Java disabilitata per impostazione predefinita e ne consentono l'abilitazione tramite una configurazione specifica del server delle applicazioni (alcuni server delle applicazioni consentono anche una configurazione più dettagliata delle politiche utilizzate in Java Security). Quando Java Sicurezza è abilitata, IBM MQ classes for Java potrebbe violare le regole di thread della politica di sicurezza Java definite per il server delle applicazioni e l'API potrebbe non essere in grado di creare tutti i thread necessari per il funzionamento. Per evitare problemi con la gestione dei thread, l'utilizzo di IBM MQ classes for Java non è supportato in ambienti in cui è abilitata la sicurezza Java .

Considerazioni sull'isolamento dell'applicazione

Un vantaggio previsto dell'esecuzione di applicazioni in un ambiente Java EE è l'isolamento dell'applicazione. La progettazione e l'implementazione di IBM MQ classes for Java precedano l'ambiente Java EE . IBM MQ classes for Java può essere utilizzato in modo da non supportare il concetto di isolamento dell'applicazione. Esempi specifici di considerazioni in questo settore sono:

- L'utilizzo delle impostazioni statiche (a livello di processo JVM) all'interno della classe MQEnvironment, come:
  - l'ID utente e la password da utilizzare per l'identificazione e l'autenticazione della connessione
  - il nome host, la porta e il canale utilizzati per le connessioni client
  - Configurazione TLS per le connessioni client protette

  La modifica delle proprietà MQEnvironment a vantaggio di un'applicazione influisce anche su altre applicazioni che utilizzano le stesse proprietà. Quando viene eseguita in un ambiente con più applicazioni come Java EE, ciascuna applicazione deve utilizzare la propria configurazione distinta mediante la creazione di oggetti MQQueueManager con una serie specifica di proprietà, piuttosto che utilizzare le proprietà configurate nella classe MQEnvironment a livello di processo.
- La classe MQEnvironment introduce un certo numero di metodi statici che agiscono globalmente su tutte le applicazioni che utilizzano IBM MQ classes for Java all'interno dello stesso processo JVM e non c'è modo di sovrascrivere questo comportamento per particolari applicazioni. Ecco alcuni esempi:
  - configurazione delle proprietà TLS, come ad esempio l'ubicazione del keystore
  - configurazione uscite canale client
  - abilitazione o disabilitazione della traccia diagnostica
  - gestione del pool di connessioni predefinito utilizzato per ottimizzare l'uso delle connessioni ai gestori code

  Il richiamo di tali metodi influisce su tutte le applicazioni in esecuzione nello stesso ambiente Java EE .
- Il pool di connessioni è abilitato per ottimizzare il processo di creazione di più connessioni allo stesso gestore code. Il gestore pool di connessioni predefinito è a livello di processo e condiviso da più applicazioni. Le modifiche alla configurazione del pool di connessioni, come la sostituzione del gestore connessioni predefinito per un'applicazione utilizzando il metodo MQEnvironment.setDefaultConnectionManager(), influiscono quindi su altre applicazioni in esecuzione nello stesso server di applicazioni Java EE.
- TLS è configurato per applicazioni che utilizzano IBM MQ classes for Java utilizzando la classe MQEnvironment e le proprietà dell'oggetto MQQueueManager . Non è integrato con la configurazione della sicurezza gestita del server delle applicazioni stesso. È necessario assicurarsi di configurare IBM MQ classes for Java in modo appropriato per fornire il livello di sicurezza richiesto e non utilizzare la configurazione del server delle applicazioni.

Limitazioni della modalità di bind

IBM MQ e WebSphere Application Server possono essere installati sulla stessa macchina in modo che le versioni principali del gestore code e dell'adattatore di risorse IBM MQ (RA) fornito in WebSphere Application Server siano differenti.

Se le versioni principali del gestore code e dell'adattatore di risorse sono differenti, non è possibile utilizzare le connessioni di bind. Tutte le connessioni da WebSphere Application Server al gestore code che utilizzano l'adattatore di risorse devono utilizzare connessioni di tipo client. Le connessioni dei collegamenti possono essere utilizzate se le versioni sono le stesse.

Conversioni di stringhe di caratteri in IBM MQ classes for Java

IBM MQ classes for Java utilizza CharsetEncoders e CharsetDecoders direttamente per la conversione della stringa di caratteri. Il funzionamento predefinito per la conversione della stringa di caratteri può essere configurato con due proprietà di sistema. La gestione dei messaggi che contengono caratteri non associabili può essere configurata tramite `com.ibm.mq.MQMD`.

Prima di IBM MQ 8.0, le conversioni di stringhe in IBM MQ classes for Java venivano eseguite richiamando i metodi `java.nio.charset.Charset.decode(ByteBuffer)` e `Charset.encode(CharBuffer)` .

L'utilizzo di uno di questi metodi determina una sostituzione predefinita (REPLACE) di dati non corretti o non convertibili. Questo comportamento può oscurare gli errori nelle applicazioni e causare caratteri imprevisti, ad esempio ?, nei dati tradotti.

Da IBM MQ 8.0, per rilevare tali problemi prima e in modo più efficace, IBM MQ classes for Java utilizzare CharsetEncoders e CharsetDecoders direttamente e configurare esplicitamente la gestione di dati con formato non corretto e non convertibili. Il comportamento predefinito è di REPORT tali problemi generando un `MQException`adatto.

Configurazione

La conversione da UTF-16 (la rappresentazione dei caratteri utilizzata in Java) a una serie di caratteri nativo, come UTF-8, viene definita *codifica*, mentre la conversione nella direzione opposta viene definita *decodifica*.

La decodifica assume il comportamento predefinito per `CharsetDecoders`, riportando errori generando un'eccezione.

Un'impostazione viene utilizzata per specificare un `java.nio.charset.CodingErrorAction` per controllare la gestione degli errori sia sulla codifica che sulla decodificazione. Un'altra impostazione viene utilizzata per controllare il byte o i byte di sostituzione durante la codifica. La stringa di sostituzione Java predefinita verrà utilizzata nelle operazioni di decodifica.

Configurazione della gestione dei dati non traducibili in IBM MQ classes for Java

Da IBM MQ 8.0, `com.ibm.mq.MQMD` include i seguenti due campi:

byte [] unMappableSostituzione
La sequenza di byte che verrà scritta in una stringa codificata se non è possibile convertire un carattere di input ed è stato specificato REPLACE.

Valore predefinito: "?".getBytes()
La stringa di sostituzione Java predefinita viene utilizzata nelle operazioni di decodifica.

java.nio.charset.CodingErrorAction unmappableAction
Specifica l'azione da intraprendere per i dati non convertibili sulla codifica e la decodifica:

Valore predefinito: CodingErrorAction.REPORT;

Proprietà di sistema per l'impostazione dei valori predefiniti di sistema

Da IBM MQ 8.0, le seguenti proprietà di sistema Java sono disponibili per configurare il funzionamento predefinito relativo alla conversione della stringa di caratteri.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Specifica l'azione da intraprendere per i dati non convertibili sulla codifica e la decodifica. Il valore può essere REPORT, REPLACEo IGNORE.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Imposta o richiama i byte di sostituzione da applicare quando un carattere non può essere mappato in un'operazione di codifica. La stringa di sostituzione predefinita Java viene utilizzata nelle operazioni di decodifica.

Per evitare confusione tra rappresentazioni di byte nativi e di caratteri Java , è necessario specificare `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` come numero decimale che rappresenta il byte di sostituzione nella serie di caratteri nativi.

Ad esempio, il valore decimale di ?, come byte nativo, è 63 se la serie di caratteri nativi è basata su ASCII, ad esempio ISO-8859-1, mentre è 111 se la serie di caratteri nativi è EBCDIC.

Nota: Si noti che se un oggetto MQMD o MQMessage ha i campi **`unmappableAction`** o **`unMappableReplacement`** impostati, i valori di questi campi hanno la precedenza sulle proprietà di sistema Java . Ciò consente ai valori specificati dalle proprietà di sistema Java di essere sovrascritti per ogni messaggio, se necessario.

Concetti correlati
“Conversioni di stringhe di caratteri in IBM MQ classes for JMS” a pagina 140

IBM MQ classes for JMS utilizza CharsetEncoders e CharsetDecoders direttamente per la conversione della stringa di caratteri. Il funzionamento predefinito per la conversione della stringa di caratteri può essere configurato con due proprietà di sistema. La gestione dei messaggi che contengono caratteri non associabili può essere configurata mediante le proprietà del messaggio per impostare l'azione UnmappableCharactere i byte di sostituzione.

Installazione e configurazione di IBM MQ classes for Java

Questa sezione descrive le directory e i file creati quando si installa IBM MQ classes for Javae indica come configurare IBM MQ classes for Java dopo l'installazione.

Cosa è installato per IBM MQ classes for Java

L'ultima versione di IBM MQ classes for Java è installata con IBM MQ. Potrebbe essere necessario sovrascrivere le opzioni di installazione predefinite per assicurarsi che ciò avvenga.

Per ulteriori informazioni sull'installazione di IBM MQ , consultare:

-

Installazione di IBM MQ
-

Installazione del prodotto IBM MQ for z/OS

IBM MQ classes for Java sono contenuti nei file JAR (Java archive), `com.ibm.mq.jar`e `com.ibm.mq.jmqi.jar`.

Il supporto per le intestazioni di messaggi standard, ad esempio PCF (Programmable Command Format), è contenuto nel file JAR `com.ibm.mq.headers.jar`.

Il supporto PCF (Programmable Command Format) è contenuto nel file JAR `com.ibm.mq.pcf.jar`.

Nota: Si consiglia di non utilizzare IBM MQ classes for Java all'interno di un server delle applicazioni. Per informazioni sulle limitazioni che si applicano durante l'esecuzione in questo ambiente, consultare “Esecuzione di applicazioni IBM MQ classes for Java in Java EE” a pagina 352. Per ulteriori informazioni, consultare Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

Importante: A parte i file JAR riposizionabili descritti in “File JAR IBM MQ classes for Java riposizionabili” a pagina 355, la copia dei file JAR IBM MQ classes for Java o delle librerie native su altre macchine o in un'altra ubicazione su una macchina su cui è stato installato IBM MQ classes for Java non è supportata.

File JAR IBM MQ classes for Java riposizionabili

I file JAR rilocabili possono essere spostati su sistemi che devono eseguire IBM MQ classes for Java.

Importante:

- A parte i file JAR riposizionabili descritti in File JAR riposizionabili, la copia dei file JAR IBM MQ classes for Java o delle librerie native su altre macchine o su un'ubicazione diversa su una macchina su cui è stato installato il IBM MQ classes for Java non è supportata.
- Per evitare conflitti del classloader, non è consigliabile raggruppare i file JAR riposizionabili all'interno di più applicazioni all'interno dello stesso runtime Java . In questo scenario, considerare la possibilità di rendere disponibili i file JAR IBM MQ rilocabili nel percorso classi del runtime Java .
- Non includere i file JAR riposizionabili all'interno delle applicazioni distribuite nei server delle applicazioni Java EE , come WebSphere Application Server. In questi ambienti, l'adattatore di risorse IBM MQ deve essere distribuito e utilizzato, poiché contiene il IBM MQ classes for Java. Tenere presente che WebSphere Application Server integra l'adattatore di risorse IBM MQ , quindi non è necessario distribuirlo manualmente in questo ambiente. Inoltre, IBM MQ classes for Java non sono supportate in WebSphere Liberty. Per ulteriori informazioni, consultare “Liberty e l'adattatore risorse IBM MQ” a pagina 443.
- Se si stanno raggruppando i file JAR riposizionabili all'interno delle applicazioni, assicurarsi di includere tutti i file JAR prerequisiti come descritto in File JAR riposizionabili. Inoltre, è necessario accertarsi di disporre delle procedure appropriate per aggiornare i file JAR integrati come parte della manutenzione dell'applicazione, per garantire che IBM MQ classes for Java rimanga aggiornato e che i problemi noti vengano rimediati.

File JAR rilocabili

All'interno di un'azienda, i seguenti file possono essere spostati su sistemi che devono eseguire applicazioni IBM MQ classes for Java :

- JMS 2.0 `com.ibm.mq.allclient.jar` "1" a pagina 356
- V 9.3.0 JM 3.0 V 9.3.0 `com.ibm.mq.jakarta.client.jar` "2" a pagina 356
- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`
- V 9.3.5 `bcpkix-jdk18on.jar` "3" a pagina 356
- V 9.3.0 V 9.3.0 `bcpkix-jdk15to18.jar` "4" a pagina 356
- V 9.3.5 `bcprov-jdk18on.jar` "3" a pagina 356
- V 9.3.0 V 9.3.0 `bcprov-jdk15to18.jar` "4" a pagina 356
- V 9.3.5 `bcutil-jdk18on.jar` "3" a pagina 356
- V 9.3.0 V 9.3.0 `bcutil-jdk15to18.jar` "4" a pagina 356
- V 9.3.3 `jackson-annotations.jar`
- V 9.3.3 `jackson-core.jar`
- V 9.3.3 `jackson-databind.jar`
- `org.json.jar`

Note:

1. JMS 2.0 e JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery da IBM MQ 9.3.5
4. Long Term Support e Continuous Delivery prima IBM MQ 9.3.5

Provider di sicurezza Bouncy Castle e file JAR di supporto CMS

Sono necessari il provider di sicurezza Bouncy Castle e i file JAR di supporto CMS . Per ulteriori informazioni, vedi Support for non -IBM JREs with AMS.

V 9.3.5 Per Continuous Delivery da IBM MQ 9.3.5, sono obbligatori i seguenti file JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS Per Long Term Support e Continuous Delivery prima di IBM MQ 9.3.5, sono obbligatori i seguenti file JAR:

- `bcpkix-jdk15to18.jar`
- `bcprov-jdk15to18.jar`
- `bcutil-jdk15to18.jar`

org.json.jar

Il file `org.json.jar` è richiesto se la propria applicazione IBM MQ classes for Java utilizza una CCDT in formato JSON.

com.ibm.mq.allclient.jar e com.ibm.mq.jakarta.client.jar

I file `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar` contengono le classi IBM MQ classes for JMS, IBM MQ classes for Javae PCF e Headers. Se si spostano questi file in una nuova ubicazione, assicurarsi di eseguire le operazioni necessarie per mantenere questa nuova ubicazione con i nuovi Fix Pack IBM MQ . Inoltre, assicurarsi che l'utilizzo dei file sia reso noto al supporto IBM se si sta ottenendo una fix temporanea.

Per determinare la versione del file `com.ibm.mq.allclient.jar` o del file `com.ibm.mq.jakarta.client.jar` , utilizzare il seguente comando:

V 9.3.0 JM 3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

Il seguente esempio mostra alcuni output di esempio di questo comando:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar, jackson-core.jar e jackson-databind.jar

V 9.3.3

I tre file Jackson JAR sono richiesti se l'applicazione IBM MQ classes for Java crea connessioni TLS sicure a un gestore code.

Directory di installazione per IBM MQ classes for Java
I file e gli esempi di IBM MQ classes for Java vengono installati in ubicazioni differenti in base alla piattaforma. L'ubicazione di JRE (Java Runtime Environment) installato con IBM MQ varia anche in base alla piattaforma.

Directory di installazione per file IBM MQ classes for Java

La Tabella 49 a pagina 357 mostra dove sono installati i file IBM MQ classes for Java .

Tabella 49. Directory di installazione IBM MQ classes for Java

Piattaforma	Directory
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib

Tabella 49. Directory di installazione IBM MQ classes for Java (Continua)	
Piattaforma	**Directory**
	/QIBM/ProdData/mqm/java/lib
Linux	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Directory di installazione per gli esempi

Alcune applicazioni di esempio, come IVP (Installation Verification Programs), vengono fornite con IBM MQ. Tabella 50 a pagina 358 mostra dove sono installate le applicazioni di esempio. Gli esempi IBM MQ classes for Java si trovano in una sottodirectory denominata `wmqjava`. Gli esempi PCF si trovano in una sottodirectory denominata `pcf`.

Tabella 50. Directory di esempi	
Piattaforma	**Directory**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i	/QIBM/ProdData/mqm/java/samples
Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Directory di installazione per JRE

IBM MQ classes for JMS richiede un JRE (Java Runtime Environment) Java 7 (o superiore). Un JRE adatto è installato con IBM MQ. Tabella 51 a pagina 358 mostra dove è installato questo JRE. Per eseguire i programmi Java come gli esempi forniti, utilizzando questo JRE, richiamare esplicitamente *JRE_LOCATION*`/bin/java` o aggiungere *JRE_LOCATION*`/bin` all'ambiente PATH (o equivalente) per la propria piattaforma, dove *JRE_LOCATION* è la directory fornita in Tabella 51 a pagina 358.

Tabella 51. Directory JRE	
Piattaforma	**Directory**
AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i	/QIBM/ProdData/mqm/java/jre
Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Variabili di ambiente relative a IBM MQ classes for Java
Se si desidera eseguire le applicazioni IBM MQ classes for Java , il percorso classi deve includere le directory IBM MQ classes for Java e degli esempi.

Per le applicazioni IBM MQ classes for Java da eseguire, il percorso classe deve includere la directory IBM MQ classes for Java appropriata. Per eseguire le applicazioni di esempio, il percorso classe deve includere anche le directory degli esempi appropriate. Queste informazioni possono essere fornite nel comando di richiamo Java o nella variabile di ambiente **CLASSPATH** .

Importante: L'impostazione dell' Java opzione `-Xbootclasspath` per includere il file IBM MQ classes for Java non è supportata.

Tabella 52 a pagina 359 mostra l'impostazione **CLASSPATH** appropriata da utilizzare su ciascuna piattaforma per eseguire applicazioni IBM MQ classes for Java , incluse le applicazioni di esempio.

*Tabella 52. **CLASSPATH** impostazione per eseguire applicazioni IBM MQ classes for Java , incluse le applicazioni di esempio IBM MQ classes for Java*

Piattaforma	CLASSPATH impostazione
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/sample / wmqjava/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/pcf

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Se si esegue la compilazione utilizzando l'opzione `-Xlint` , potrebbe essere visualizzato un messaggio che indica che `com.ibm.mq.ese.jar` non è presente. È possibile ignorare l'avvertenza. Questo file è presente solo se è stato installato Advanced Message Security.

Gli script forniti con IBM MQ classes for JMS utilizzano le variabili di ambiente riportate di seguito:

MQ_JAVA_DATA_PATH
Questa variabile di ambiente specifica la directory per l'output di log e traccia.

PERCORSO_INSTALL_JAVA_MQ
Questa variabile di ambiente specifica la directory in cui è installato IBM MQ classes for Java , come mostrato in Directory di installazione IBM MQ classes for Java.

MQ_JAVA_LIB_PATH
Questa variabile di ambiente specifica la directory in cui sono memorizzate le librerie IBM MQ classes for Java , come mostrato in L'ubicazione delle librerie IBM MQ classes for Java per ogni piattaforma. Alcuni script forniti con IBM MQ classes for Java, come IVTRun, utilizzano questa variabile di ambiente.

Windows In Windows, tutte le variabili di ambiente vengono impostati automaticamente durante l'installazione.

Linux AIX Su AIX and Linux, è possibile utilizzare lo script `setjmsenv` (se si utilizza una JVM a 32 bit) o `setjmsenv64` (se si utilizza una JVM a 64 bit) per impostare le variabili di ambiente. Questi script si trovano nella directory *MQ_INSTALLATION_PATH*/java/bin .

IBM i in IBM i, la variabile di ambiente **QIBM_MULTI_THREADED** deve essere impostata su Y. È quindi possibile eseguire le applicazioni a più thread nello stesso modo in cui si eseguono le applicazioni a singolo thread. Per ulteriori informazioni, consultare Impostazione di IBM MQ con Java e JMS.

IBM MQ classes for Java richiede un JRE (Java 7 Java Runtime Environment). Per informazioni sull'ubicazione di un JRE adatto installato con IBM MQ, consultare “Directory di installazione per IBM MQ classes for Java” a pagina 357.

IBM MQ classes for Java librerie
L'ubicazione delle librerie IBM MQ classes for Java varia in base alla piattaforma. Specificare questa ubicazione quando si avvia una applicazione.

Per specificare il percorso delle librerie JNI (Java Native Interface), avviare l'applicazione utilizzando un comando **java** con il seguente formato:

```
java -Djava.library.path= library_path application_name
```

dove *library_path* è il percorso di IBM MQ classes for Java, che include le librerie JNI. Tabella 53 a pagina 360 mostra l'ubicazione delle librerie IBM MQ classes for Java per ogni piattaforma. In questa tabella *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ .

Tabella 53. L'ubicazione delle librerie IBM MQ classes for Java per ciascuna piattaforma

Piattaforma	**Directory contenente le librerie IBM MQ classes for Java**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib (librerie a 32 bit) *MQ_INSTALLATION_PATH*/java/lib64 (librerie a 64 bit)
Linux Linux (piattaformax86)	*MQ_INSTALLATION_PATH*/java/lib
Linux Linux (piattaforme POWER, x86-64 e zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (librerie a 32 bit) *MQ_INSTALLATION_PATH*/java/lib64 (librerie a 64 bit)
Windows Windows	*MQ_INSTALLATION_PATH*\Java\lib (librerie a 32 bit) *MQ_INSTALLATION_PATH*\Java\lib64 (librerie a 64 bit)
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (librerie a 32 bit e a 64 bit)

Nota:

1. Linux AIX Su AIX o Linux (piattaforma Power), utilizzare librerie a 32 bit o a 64 bit. Utilizzare le librerie a 64 bit solo se si sta eseguendo l'applicazione in una JVM (Java virtual machine) a 64 bit su una piattaforma a 64 bit. In caso contrario, utilizzare le librerie a 32 bit.
2. Windows In Windows, è possibile utilizzare la variabile di ambiente PATH per specificare l'ubicazione delle librerie di IBM MQ classes for Java invece di specificarne l'ubicazione nel comando **java** .
3. IBM i Per utilizzare IBM MQ classes for Java in modalità bind su IBM i, assicurarsi che la libreria QMQMJAVA sia presente nell'elenco librerie.

4. z/OS Su z/OS, è possibile utilizzare una JVM (Java virtual machine) a 32 bit o a 64 bit. Non è necessario specificare quali librerie utilizzare; IBM MQ classes for Java può stabilire da solo quali librerie JNI caricare.

Concetti correlati

Utilizzo di IBM MQ classes for Java
Dopo l'installazione di IBM MQ classes for Java, è possibile configurare l'installazione in modo da eseguire le proprie applicazioni.

Supporto per OSGi con IBM MQ classes for Java
OSGi fornisce un framework che supporta la distribuzione delle applicazioni come bundle. Tre bundle OSGi vengono forniti come parte di IBM MQ classes for Java.

OSGi fornisce un framework Java generico, sicuro e gestito, che supporta la distribuzione di applicazioni fornite sotto forma di bundle. I dispositivi conformi a OSGi possono scaricare e installare i bundle e rimuoverli quando non sono più necessari. Il framework gestisce l'installazione e l'aggiornamento dei bundle in modo dinamico e scalabile.

IBM MQ classes for Java include i seguenti bundle OSGi.

com.ibm.mq.osgi.java_*version_number*.jar
I file JAR per consentire alle applicazioni di utilizzare IBM MQ classes for Java.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, questo file JAR consente alle applicazioni di utilizzare sia IBM MQ classes for JMS che IBM MQ classes for Javae include anche il codice per gestire i messaggi PCF.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Per JMS 2.0, questo file JAR consente alle applicazioni di utilizzare sia IBM MQ classes for JMS che IBM MQ classes for Javae include anche il codice per gestire i messaggi PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, questo file JAR fornisce i prerequisiti per `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Per JMS 2.0, questo file JAR fornisce i prerequisiti per `com.ibm.mq.osgi.allclient_version_number.jar`.

dove *version_number* è il numero di versione di IBM MQ installato.

I bundle sono installati nella sottodirectory `java/lib/OSGi` dell'installazione di IBM MQ o nella cartella `java\lib\OSGi` su Windows.

Da IBM MQ 8.0, utilizzare i bundle `com.ibm.mq.osgi.allclient_8.0.0.0.jare com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` per le nuove applicazioni. L'uso di questi bundle rimuove la limitazione di non essere in grado di eseguire IBM MQ classes for JMS e IBM MQ classes for Java all'interno dello stesso framework OSGi. Tuttavia, tutte le altre restrizioni sono ancora applicabili. Per le versioni di IBM MQ precedenti a IBM MQ 8.0, si applica la limitazione dell'utilizzo di IBM MQ classes for JMS o IBM MQ classes for Java .

Nove altri bundle vengono installati anche nella sottodirectory `java/lib/OSGi` dell'installazione di IBM MQ o nella cartella `java\lib\OSGi` su Windows. Questi bundle fanno parte di IBM MQ classes for JMSe non devono essere caricati in un ambiente di runtime OSGi che ha il bundle IBM MQ classes for Java caricato. Se il bundle OSGi IBM MQ classes for Java è caricato in un ambiente di runtime OSGi che ha anche i bundle IBM MQ classes for JMS caricati, si verificano errori come mostrato nel seguente esempio quando vengono eseguite le applicazioni che utilizzano il bundle IBM MQ classes for Java o i bundle IBM MQ classes for JMS :

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

Il bundle OSGi per IBM MQ classes for Java è stato scritto nella specifica OSGi Release 4; non funziona in un ambiente OSGi Release 3.

È necessario impostare correttamente il percorso di sistema o il percorso della libreria in modo che l'ambiente di runtime OSGi possa trovare i file DLL o le librerie condivise richiesti.

Se si utilizza il bundle OSGi per IBM MQ classes for Java, le classi di uscita del canale scritte in Java non sono supportate a causa di un problema intrinseco nel caricamento delle classi in un ambiente con più programmi di caricamento classi come OSGi. Un bundle utente può essere a conoscenza del bundle IBM MQ classes for Java , ma il bundle IBM MQ classes for Java non è a conoscenza di alcun bundle utente. Di conseguenza, il programma di caricamento classi utilizzato in un bundle IBM MQ classes for Java non può caricare una classe di uscita del canale che si trova in un bundle utente.

Per ulteriori informazioni su OSGi, consultare il sito Web OSGi alliance .

z/OS *Installazione di IBM MQ classes for Java su z/OS*

Su z/OS, la STEPLIB utilizzata al runtime deve contenere le librerie IBM MQ SCSQAUTH e SCSQANLE.

Da z/OS UNIX System Services, è possibile aggiungere queste librerie utilizzando una riga nel `.profile` come mostrato nel seguente esempio, sostituendo `thlqual` con il qualificatore del dataset di alto livello scelto durante l'installazione di IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In altri ambienti, è generalmente necessario modificare il JCL di avvio per includere SCSQAUTH nella concatenazione STEPLIB:

```
 STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

Il file di configurazione IBM MQ classes for Java

Un file di configurazione IBM MQ classes for Java specifica le proprietà utilizzate per configurare IBM MQ classes for Java.

Il formato di un file di configurazione IBM MQ classes for Java è quello di un file delle proprietà Java standard.

Un file di configurazione di esempio, `mqjava.config`, viene fornito nella sottodirectory `bin` della directory di installazione IBM MQ classes for Java . Questo file documenta tutte le proprietà supportate e i relativi valori predefiniti.

Nota: Il file di configurazione di esempio viene sovrascritto quando l'installazione di IBM MQ viene aggiornata a un fix pack futuro. Di conseguenza, si consiglia di creare una copia del file di configurazione di esempio da utilizzare con le applicazioni.

È possibile scegliere il nome e l'ubicazione di un file di configurazione IBM MQ classes for Java . Quando si avvia l'applicazione, utilizzare un comando **java** con il seguente formato:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

Nel comando, *config_file_url* è un URL (uniform resource locator) (URL) che specifica il nome e l'ubicazione del file di configurazione IBM MQ classes for Java . Sono supportati URL dei seguenti tipi: `http`, `file`, `ftpe` `jar`.

Il seguente esempio mostra un comando **java** :

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

Questo comando identifica il file di configurazione IBM MQ classes for Java come file `D:\mydir\mqjava.config` sul sistema Windows locale.

All'avvio di un'applicazione, IBM MQ classes for Java legge il contenuto del file di configurazione e memorizza le proprietà specificate in un archivio delle proprietà interno. Se il comando **java** non identifica un file di configurazione o se non è possibile trovare il file di configurazione, IBM MQ classes for Java utilizzerà i valori predefiniti per tutte le proprietà. Se necessario, è possibile sovrascrivere qualsiasi proprietà nel file di configurazione specificandola come proprietà di sistema nel comando **java** .

Un file di configurazione IBM MQ classes for Java può essere utilizzato con uno qualsiasi dei trasporti supportati tra un'applicazione e un gestore code o broker.

Sostituzione delle proprietà specificate in un file di configurazione IBM MQ MQI client

Un file di configurazione IBM MQ MQI client può specificare anche le proprietà utilizzate per configurare IBM MQ classes for Java. Tuttavia, le proprietà specificate in un file di configurazione IBM MQ MQI client vengono applicate solo quando un'applicazione si connette a un gestore code in modalità client.

Se richiesto, è possibile sovrascrivere qualsiasi attributo in un file di configurazione IBM MQ MQI client specificandolo come proprietà in un file di configurazione IBM MQ classes for Java . Per sovrascrivere un attributo in un file di configurazione IBM MQ MQI client , utilizzare una voce con il formato seguente nel file di configurazione IBM MQ classes for Java :

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Le variabili nella voce hanno i seguenti significati:

stanza
Il nome della sezione nel file di configurazione IBM MQ MQI client che contiene l'attributo.

propName
Il nome dell'attributo come specificato nel file di configurazione IBM MQ MQI client .

propValue
Il valore della proprietà che sovrascrive il valore dell'attributo specificato nel file di configurazione IBM MQ MQI client .

In alternativa, è possibile sovrascrivere un attributo in un file di configurazione IBM MQ MQI client specificando la proprietà come proprietà di sistema nel comando **java** . Utilizzare il formato precedente per specificare la proprietà come proprietà di sistema.

Solo i seguenti attributi in un file di configurazione IBM MQ MQI client sono rilevanti per IBM MQ classes for Java. Se si specificano o si sovrascrivono altri attributi, non ha alcun effetto. In particolare, notare che `ChannelDefinitionFile` e `ChannelDefinitionDirectory` nella stanza CHANNELS del file di configurazione client non vengono utilizzati. Consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for Java” a pagina 379 per i dettagli su come utilizzare CCDT con IBM MQ classes for Java.

Tabella 54. Quale stanza del file di configurazione client contiene quale attributo

stanza	Attributo
Stanza CHANNELS del file di configurazione del client	Put1DefaultAlwaysSync
Stanza CHANNELS del file di configurazione del client	PasswordProtection
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath64
ClientExitStanza del percorso del file di configurazione client	Percorso classi JavaExits

Tabella 54. Quale stanza del file di configurazione client contiene quale attributo (Continua)	
stanza	**Attributo**
Stanza JMQI del file di configurazione del client	useMQCSPauthentication
Stanza MessageBuffer del file di configurazione client	MaximumSize
Stanza MessageBuffer del file di configurazione client	PurgeTime
Stanza MessageBuffer del file di configurazione client	UpdatePercentage
Stanza TCP del file di configurazione client	ClntRcvBuffSize
Stanza TCP del file di configurazione client	ClntSndBuffSize
Stanza TCP del file di configurazione client	Timeout connessione
Stanza TCP del file di configurazione client	KeepAlive

Per ulteriori informazioni sulla configurazione di IBM MQ MQI client , consultare il file di configurazione di IBM MQ MQI client , `mqclient.ini`.

Attività correlate

Traccia delle classi IBM MQ per le applicazioni Java

Utilizzo della traccia dell'ambiente standard Java per configurare la traccia Java

Utilizzare la stanza Java Standard Environment Trace Settings per configurare la funzione di traccia IBM MQ classes for Java .

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputNome*

traceOutputName è la directory e il nome file a cui viene inviato l'output di traccia.

Per impostazione predefinita, le informazioni di traccia vengono scritte in un file di traccia nella directory di lavoro corrente dell'applicazione. Il nome del file di traccia dipende dall'ambiente in cui è in esecuzione l'applicazione:

- Da IBM MQ 9.1.5 e IBM MQ 9.1.0 Fix Pack 5:
  - Se l'applicazione ha caricato il IBM MQ classes for Java dal file JAR riposizionabile `com.ibm.mq.allclient.jar`, la traccia viene scritta in un file denominato `mqjavaclient_%PID%.cl%u.trc`.
  - Se l'applicazione ha caricato IBM MQ classes for Java dal file JAR `com.ibm.mq.jar`, la traccia viene scritta in un file denominato `mqjava_%PID%.cl%u.trc`.
- Da IBM MQ 9.0.0 Fix Pack 2:
  - Se l'applicazione ha caricato il IBM MQ classes for Java dal file JAR riposizionabile `com.ibm.mq.allclient.jar`, la traccia viene scritta in un file denominato `mqjavaclient_%PID%.trc`.
  - Se l'applicazione ha caricato IBM MQ classes for Java dal file JAR `com.ibm.mq.jar`, la traccia viene scritta in un file denominato `mqjava_%PID%.trc`.
- Per IBM MQ classes for Java per IBM MQ 9.0.0 Fix Pack 1 o versioni precedenti, la traccia viene scritta in un file denominato `mqjms_%PID%.trc`.

dove *%PID%* è l'identificativo del processo dell'applicazione di cui viene eseguita la traccia e *%u* è un numero univoco per differenziare i file tra i thread che eseguono la traccia in diversi programmi di caricamento classi Java .

Se un ID processo non è disponibile, viene generato un numero casuale con la lettera `f`come prefisso. Per includere l'ID processo in un nome file specificato, utilizzare la stringa `%PID%`.

Se si specifica una directory alternativa, è necessario che esista e che si disponga dell'autorizzazione di scrittura per questa directory. Se non si dispone dell'autorizzazione di scrittura, l'output di traccia viene scritto in `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList è un elenco di package e classi di cui viene eseguita la traccia o i valori speciali `ALL` o `NONE`.

Separare i nomi pacchetto o classe con un punto e virgola, `;`. *includeList* assume il valore predefinito `ALL`e traccia tutti i pacchetti e le classi in IBM MQ classes for Java.

Nota: È possibile includere un package, ma escludere i package secondari di tale package. Ad esempio, se si include il pacchetto `a.b` e si esclude il pacchetto `a.b.x`, la traccia include tutto in `a.b.y` e `a.b.z`, ma non in `a.b.x` o `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList è un elenco di package e classi di cui non viene eseguita la traccia o i valori speciali `ALL` o `NONE`.

Separare i nomi pacchetto o classe con un punto e virgola, `;`. *excludeList* assume il valore predefinito `NONE`e pertanto non esclude dalla traccia alcun package e classe in IBM MQ classes for JMS .

Nota: È possibile escludere un package ma includere i package secondari di tale package. Ad esempio, se si esclude il pacchetto `a.b` e si include il pacchetto `a.b.x`, la traccia include tutto in `a.b.x` e `a.b.x.1`, ma non `a.b.y` o `a.b.z`.

Sono inclusi tutti i package o le classi specificati, allo stesso livello, inclusi ed esclusi.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayByte*
maxArrayBytes è il numero massimo di byte tracciati da qualsiasi array di byte.

Se *maxArrayBytes* è impostato su un numero intero positivo, limita il numero di byte in una schiera di byte scritti nel file di traccia. Tronca la schiera di byte dopo la scrittura di *maxArrayBytes* . L'impostazione *maxArrayBytes* riduce la dimensione del file di traccia risultante e l'effetto della traccia sulle prestazioni dell'applicazione.

Un valore di `0` per questa proprietà indica che nessuno dei contenuti delle schiere di byte viene inviato al file di traccia.

Il valore predefinito è `-1`, che elimina qualsiasi limite sul numero di byte in una schiera di byte inviati al file di traccia.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceByte*
maxTraceBytes è il numero massimo di byte scritti in un file di output di traccia.

maxTraceBytes funziona con *traceCycles*. Se il numero di byte di traccia scritti è vicino al limite, il file viene chiuso e viene avviato un nuovo file di output di traccia.

Un valore `0` indica che un file di output di traccia ha lunghezza zero. Il valore predefinito è `-1`, che significa che la quantità di dati da scrivere in un file di output di traccia è illimitata.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles è il numero di file di output di traccia da scorrere.

Se il file di output di traccia corrente raggiunge il limite specificato da *maxTraceBytes*, il file viene chiuso. Un ulteriore output di traccia viene scritto nel successivo file di output di traccia in sequenza. Ogni file di output di traccia è distinto da un suffisso numerico aggiunto al nome file. Il file di output di traccia corrente o più recente è `mqjms.trc.0`, il successivo file di output di traccia più recente è `mqjms.trc.1`. I file di traccia più vecchi seguono lo stesso schema di numerazione fino al limite.

Il valore predefinito di *traceCycles* è 1. Se *traceCycles* è 1, quando il file di output di traccia corrente raggiunge la dimensione massima, il file viene chiuso ed eliminato. Viene avviato un nuovo file di output di traccia con lo stesso nome. Pertanto, esiste un solo file di output di traccia alla volta.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters controlla se i parametri del metodo e i valori di ritorno sono inclusi nella traccia.

traceParameters assume il valore predefinito TRUE. Se *traceParameters* è impostato su FALSE, viene eseguita la traccia solo delle firme del metodo.

com.ibm.msg.client.commonservices.trace.startup = *avvio*
C'è una fase di inizializzazione di IBM MQ classes for Java durante la quale vengono assegnate le risorse. La funzione di traccia principale viene inizializzata durante la fase di allocazione delle risorse.

Se *startup* è impostato su TRUE, viene utilizzata la traccia di avvio. Le informazioni di traccia vengono prodotte immediatamente e includono la configurazione di tutti i componenti, inclusa la funzionalità di traccia stessa. Le informazioni di traccia di avvio possono essere utilizzate per diagnosticare i problemi di configurazione. Le informazioni sulla traccia di avvio vengono sempre scritte in `System.err`.

startup assume il valore predefinito FALSE.

startup viene controllato prima che l'inizializzazione sia completa. Per questo motivo, specificare la proprietà sulla riga comandi solo come proprietà di sistema Java . Non specificarla nel file di configurazione IBM MQ classes for Java .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Impostare *compressedTrace* su TRUE per comprimere l'output di traccia.

Il valore predefinito di *compressedTrace* è FALSE.

Se *compressedTrace* è impostato su TRUE, l'output di traccia viene compresso. Il nome file di output di traccia predefinito ha l'estensione `.trz`. Se la compressione è impostata su FALSE, il valore predefinito, il file ha l'estensione `.trc` per indicare che è decompresso. Tuttavia, se il nome file per l'output di traccia è stato specificato in *traceOutputName* , viene utilizzato tale nome; al file non viene applicato alcun suffisso.

L'output della traccia compressa è più piccolo di quello decompresso. Poiché c'è meno I/O, può essere scritto più velocemente della traccia non compressa. La traccia compressa ha un effetto minore sulle prestazioni di IBM MQ classes for Java rispetto alla traccia non compressa.

Se *maxTraceBytes* e *traceCycles* sono impostati, vengono creati più file di traccia compressi invece di più file flat.

Se IBM MQ classes for Java termina in modo non controllato, un file di traccia compresso potrebbe non essere valido. Per questo motivo, la compressione della traccia deve essere utilizzata solo quando IBM MQ classes for Java si chiude in modo controllato. Utilizzare la compressione della traccia solo se i problemi esaminati non causano l'arresto imprevisto della stessa JVM. Non utilizzare la compressione di traccia quando si diagnosticano problemi che possono causare `System.Halt()` arresti o terminazioni JVM anomale e non controllate.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel specifica un livello di filtro per la traccia. I livelli di traccia definiti sono i seguenti:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Ogni livello di traccia include tutti i livelli inferiori. Ad esempio, se il livello di traccia è impostato su `TRACE_INFO`, qualsiasi punto di traccia con un livello definito `TRACE_EXCEPTION`, `TRACE_WARNING`o `TRACE_INFO` viene scritto nella traccia. Tutti gli altri punti di traccia sono esclusi.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace controlla se il servizio di traccia del client IBM MQ classes for Java viene utilizzato in un ambiente WebSphere Application Server .

Se *standaloneTrace* è impostato su TRUE, le proprietà di traccia del client IBM MQ classes for Java vengono utilizzate per determinare la configurazione della traccia.

Se *standaloneTrace* è impostato su FALSEe il client IBM MQ classes for Java è in esecuzione in un contenitore WebSphere Application Server , viene utilizzato il servizio di traccia WebSphere Application Server . Le informazioni di traccia generate dipendono dalle impostazioni di traccia del server delle applicazioni.

Il valore predefinito di *standaloneTrace* è FALSE.

IBM MQ classes for Java e strumenti di gestione software
Gli strumenti di gestione software come Apache Maven possono essere utilizzati con IBM MQ classes for Java.

Molte grandi organizzazioni di sviluppo utilizzano questi strumenti per gestire centralmente i repository di librerie di terze parti.

I IBM MQ classes for Java sono composti da un numero di file JAR. Quando si sviluppano le applicazioni del linguaggio Java utilizzando questa API, è richiesta un'installazione di IBM MQ Server, IBM MQ Client o IBM MQ Client SupportPac sulla macchina su cui si sta sviluppando l'applicazione.

Se si desidera utilizzare uno strumento di gestione software e aggiungere i file JAR che costituiscono IBM MQ classes for Java a un repository gestito centralmente, è necessario osservare i seguenti punti:

- Un repository o un contenitore deve essere reso disponibile solo agli sviluppatori all'interno della propria organizzazione. Non è consentita alcuna distribuzione al di fuori dell'organizzazione.
- Il repository deve contenere una serie completa e coerente di file JAR da una singola release IBM MQ o Fix Pack.
- L'utente è responsabile dell'aggiornamento del repository con qualsiasi manutenzione fornita dal supporto IBM .

Da IBM MQ 8.0, il file JAR `com.ibm.mq.allclient.jar` deve essere installato nel repository.

Da IBM MQ 9.0, il provider di sicurezza Bouncy Castle e i file JAR di supporto CMS sono obbligatori. Per ulteriori informazioni, vedi “File JAR IBM MQ classes for Java riposizionabili” a pagina 355 e Support for non -IBM JREs.

Configurazione post - installazione per applicazioni IBM MQ classes for Java

Dopo l'installazione di IBM MQ classes for Java, è possibile configurare l'installazione in modo da eseguire le proprie applicazioni.

Verificare il file readme del prodotto IBM MQ per le informazioni più recenti o per informazioni più specifiche sull'ambiente. L'ultima versione del file readme del prodotto è disponibile sulla pagina Web IBM MQ, WebSphere MQ, e i readme del prodotto MQSeries .

Prima di tentare l'esecuzione di un'applicazione IBM MQ classes for Java in modalità bind, assicurarsi di aver configurato IBM MQ come descritto in Configurazione.

Configurazione del gestore code per accettare le connessioni client da IBM MQ classes for Java
Per configurare il gestore code per accettare le richieste di connessione in entrata dai client, definire e consentire l'utilizzo di un canale di connessione server e avviare un programma listener.

Vedi “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072 per i dettagli.

Esecuzione delle applicazioni IBM MQ classes for Java in Java security manager
IBM MQ classes for Java può essere eseguito con Java security manager abilitato. Per eseguire correttamente le applicazioni con Java security manager abilitato, è necessario configurare Java Virtual Machine (JVM) con un appropriato file di definizione delle politiche.

Il modo più semplice per creare un file di definizione della politica adatto consiste nel modificare il file della politica fornito con Java runtime environment (JRE). Sulla maggior parte dei sistemi, questo file è

memorizzato in `path lib/security/java.policy`, relativo alla directory JRE. È possibile modificare i file delle politiche utilizzando l'editor preferito o il programma policytool fornito con JRE.

È necessario concedere l'autorizzazione al file `com.ibm.mq.jmqi.jar` in modo che possa:

- Crea socket (in modalità client)
- Carica la libreria nativa (in modalità bind)
- Leggi varie proprietà dall'ambiente

La proprietà di sistema **os.name** deve essere disponibile per IBM MQ classes for Java quando viene eseguita in Java security manager.

Se l'applicazione Java utilizza Java security manager, è necessario aggiungere la seguente autorizzazione al file `java.security.policy` utilizzato dall'applicazione, altrimenti verranno generate delle eccezioni all'applicazione:

```
permission java.lang.RuntimePermission "modifyThread";
```

Questa RuntimePermission è richiesta dal client come parte della gestione dell'assegnazione e della chiusura di conversazioni multiplex su connessioni TCP/IP ai gestori code.

Voce del file della politica di esempio

Questo è un esempio di una voce del file delle politiche che consente a IBM MQ classes for Java di essere eseguito correttamente con il gestore della sicurezza predefinito. Sostituire la stringa *MQ_INSTALLATION_PATH* in questo esempio con l'ubicazione in cui è installato IBM MQ classes for Java sul proprio sistema.

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";
```

```
//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

Questo esempio di un file di politica consente a IBM MQ classes for Java di lavorare correttamente sotto il gestore della sicurezza, ma potrebbe essere ancora necessario abilitare il proprio codice per essere eseguito correttamente prima che le applicazioni funzionino.

Il codice di esempio fornito con IBM MQ classes for Java non è stato specificamente abilitato per l'utilizzo con il gestore della sicurezza; tuttavia, i test IVT vengono eseguiti con questo file delle politiche e con il gestore della sicurezza predefinito.

Importante:

La funzione di traccia IBM MQ classes for Java richiede ulteriori autorizzazioni in quanto esegue ulteriori query delle proprietà di sistema e ulteriori operazioni del file system.

Un file della politica di sicurezza del template adatto per l'esecuzione in un gestore sicurezza con la traccia abilitata viene fornito nella directory `samples/wmqjava` dell'installazione di IBM MQ come `example.security.policy`.

Per un'installazione predefinita, il file `example.security.policy` si trova in:

Windows
In `C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`

Linux
In `/opt/mqm/samp/wmqjava/samples/example.security.policy`

Solaris
In `/opt/mqm/samp/wmqjava/samples/example.security.policy`

AIX
In `/usr/mqm/samp/wmqjava/samples/example.security.policy`

Esecuzione di applicazioni IBM MQ classes for Java in CICS Transaction Server
Un'applicazione IBM MQ classes for Java può essere eseguita come transazione in CICS Transaction Server.

Per eseguire un'applicazione IBM MQ classes for Java come transazione in CICS Transaction Server per z/OS, effettuare le seguenti operazioni:

1. Definire l'applicazione e la transazione in CICS utilizzando la transazione CEDA fornita.
2. Accertarsi che l'adattatore IBM MQ CICS sia installato nel proprio sistema CICS . z/OS (Per i dettagli, consultare Utilizzo di IBM MQ con CICS .)
3. Verificare che l'ambiente JVM specificato in CICS includa le voci CLASSPATH e LIBPATH appropriate.
4. Avviare la transazione utilizzando uno dei normali processi.

Per ulteriori informazioni sull'esecuzione delle transazioni CICS Java , fare riferimento alla documentazione del sistema CICS .

Verifica dell'installazione di IBM MQ classes for Java

Un programma di verifica dell'installazione, MQIVP, viene fornito con IBM MQ classes for Java. È possibile utilizzare questo programma per verificare tutte le modalità di connessione di IBM MQ classes for Java.

Il programma richiede un numero di scelte e altri dati per determinare la modalità di connessione che si desidera verificare. Utilizzare la seguente procedura per verificare l'installazione:

1. Se si sta per eseguire il programma in modalità client, configurare il gestore code come descritto in “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072. La coda da utilizzare è SYSTEM.DEFAULT.LOCAL.QUEUE
2. Se si sta per eseguire il programma in modalità client, consultare anche “Utilizzo di IBM MQ classes for Java” a pagina 349.

   Eseguire i passi rimanenti di questa procedura sul sistema su cui si sta per eseguire il programma.
3. Accertarsi di aver aggiornato la propria variabile di ambiente CLASSPATH in conformità con le istruzioni in “Variabili di ambiente relative a IBM MQ classes for Java” a pagina 359.
4. Modificare la directory in *MQ_INSTALLATION_PATH*`/mqm/samp/wmqjava/samples`, dove *MQ_INSTALLATION_PATH* è il percorso dell'installazione di IBM MQ . Quindi, nel prompt dei comandi, immettere:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   dove *library_path* è il percorso delle librerie IBM MQ classes for Java (consultare “IBM MQ classes for Java librerie” a pagina 360).

   Al prompt contrassegnato (1):

   - Per utilizzare una connessione TCP/IP, immettere un nome host server IBM MQ .
   - Per utilizzare la connessione nativa (modalità bind), lasciare il campo vuoto (non immettere un nome).

   Il programma tenta di:

   - 1. Collegarsi al gestore code
   - 2. Aprire la coda SYSTEM.DEFAULT.LOCAL.QUEUE, inserire un messaggio nella coda, richiamare un messaggio dalla coda e chiudere la coda
   - 3. Disconnettersi dal gestore code
   - 4. Restituire un messaggio se le operazioni hanno esito positivo

Di seguito è riportato un esempio delle richieste e delle risposte che potrebbero essere visualizzate. Le richieste e le risposte effettive dipendono dalla rete IBM MQ .

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
```

```
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

Nota:

1. z/OS In z/OS, lasciare vuoto il campo alla richiesta contrassegnata con (1).
2. Se si sceglie la connessione server, non vengono visualizzati i prompt contrassegnati con (2).
3. IBM i Su IBM i, è possibile immettere solo il comando `java MQIVP` da QShell. In alternativa, è possibile eseguire l'applicazione utilizzando il comando CL `RUNJVA CLASS(MQIVP)`.

Utilizzo di applicazioni di esempio IBM MQ classes for Java

Le applicazioni di esempio IBM MQ classes for Java forniscono una panoramica delle funzioni comuni dell'API IBM MQ classes for Java . È possibile utilizzarle per verificare l'installazione e la configurazione del server di messaggistica e per creare le proprie applicazioni.

Informazioni su questa attività

Se si ha bisogno di aiuto per creare le proprie applicazioni, è possibile utilizzare le applicazioni di esempio come punto di partenza. Sia l'origine che una versione compilata vengono fornite per ciascuna applicazione. Esaminare il codice di origine di esempio e identificare i passi chiave per creare ogni oggetto richiesto per la propria applicazione (MQQueueManager, MQConstants, MQMessage, MQPutMessageOptions e MQDestination) e per impostare le proprietà specifiche necessarie per specificare il modo in cui si desidera che la propria applicazione funzioni. Per ulteriori informazioni, consultare “Scrittura di applicazioni IBM MQ classes for Java” a pagina 374. Gli esempi potrebbero essere soggetti a modifiche nelle release future di IBM MQ Java.

Tabella 55 a pagina 371 mostra dove sono installate le IBM MQ classes for Java applicazioni di esempio su ciascuna piattaforma:

Tabella 55. Directory di installazione per le applicazioni di esempio IBM MQ classes for Java

Piattaforma	Directory
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\wmqjava\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/wmqjava/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/wmqjava`

Tabella 56 a pagina 371 mostra le serie di applicazioni di esempio fornite con IBM MQ classes for Java.

Tabella 56. IBM MQ classes for JavaApplicazioni di esempio

Nome del campione	Descrizione
IMSBridgeSample.java	Programma semplice per dimostrare l'utilizzo di IMS Bridge con IBM MQ classes for Java.
MQIVP.java	Programma di verifica dell'installazione di IBM MQ Java .

Tabella 56. IBM MQ classes for JavaApplicazioni di esempio (Continua)	
Nome del campione	**Descrizione**
MQMessagePropertiesSample.java	Dimostra l'utilizzo dell'API Proprietà messaggio.
MQPubSubApiSample.java	Dimostra di utilizzare l'API di pubblicazione / sottoscrizione.
MQSample.java	Semplice programma per dimostrare l'inserimento e il richiamo di un messaggio da una coda.
MQSampleMessageManager.java	Classe di utilità per la gestione dei messaggi negli esempi IBM MQ base Java .
mqjcivp.properties	Questo bundle di risorse contiene i messaggi utilizzati dal programma di verifica dell'installazione di IBM MQ classes for Java (`MQIVP.java`).

Il IBM MQ classes for Java fornisce uno script denominato `runjms` che può essere utilizzato per eseguire le applicazioni di esempio. Questo script imposta l'ambiente IBM MQ per consentire l'esecuzione delle applicazioni di esempio IBM MQ classes for Java .

Tabella 57 a pagina 372 mostra l'ubicazione dello script su ciascuna piattaforma:

Tabella 57. Ubicazione dello script `runjms`	
Piattaforma	**Directory**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	o
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`java/bin/runjms`

Per utilizzare uno script `runjms` per richiamare un'applicazione di esempio, completare la seguente procedura:

Procedura

1. Visualizzare un prompt dei comandi e passare alla directory contenente l'applicazione di esempio che si desidera eseguire.
2. Immettere il seguente comando:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   L'applicazione di esempio visualizza un elenco di parametri necessari.
3. Immettere il seguente comando per eseguire l'esempio con questi parametri:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Esempio

Linux Ad esempio, per eseguire l'esempio MQIVP su Linux, immettere i seguenti comandi:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

Concetti correlati

“Cosa è installato per IBM MQ classes for JMS” a pagina 91
Una serie di file e directory vengono creati quando si installa IBM MQ classes for JMS. Su Windows, alcune configurazioni vengono eseguite durante l'installazione impostando automaticamente le variabili di ambiente. Su altre piattaforme e in determinati ambienti Windows , è necessario impostare le variabili di ambiente prima di eseguire applicazioni IBM MQ classes for JMS .

Risoluzione dei problemi di IBM MQ classes for Java

Inizialmente, eseguire il programma di verifica dell'installazione. Potrebbe anche essere necessario utilizzare la funzionalità di traccia.

Se un'applicazione non viene completata correttamente, eseguire il programma di verifica dell'installazione e seguire i consigli forniti nei messaggi diagnostici. Il programma di verifica dell'installazione è descritto in “Verifica dell'installazione di IBM MQ classes for Java” a pagina 370.

Se i problemi persistono ed è necessario contattare il team di assistenza IBM , potrebbe essere richiesto di attivare la funzionalità di traccia. Eseguire questa operazione come mostrato nel seguente esempio.

Per tracciare il programma `MQIVP` :

- Creare un file delle proprietà `com.ibm.mq.commonservices` (consultare Utilizzo di com.ibm.mq.commonservices.
- Immettere il seguente comando:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

dove:

- *commonservices_properties_file* è il percorso (incluso il nome file) del file delle proprietà `com.ibm.mq.commonservices` .
- *library_path* è il percorso delle librerie IBM MQ classes for Java (consultare “IBM MQ classes for Java librerie” a pagina 360).

Per ulteriori informazioni su come utilizzare la traccia, consultare Traccia delle applicazioni IBM MQ classes for Java.

z/OS MQ Adv. VUE Java connettività client alle applicazioni batch in esecuzione su z/OS

In determinate condizioni, un'applicazione di IBM MQ classes for Java su z/OS può connettersi a un gestore code su z/OS utilizzando una connessione client. L'utilizzo di una connessione client può semplificare le topologie IBM MQ .

Utilizzando una connessione client, se un'applicazione IBM MQ classes for Java è in esecuzione in un ambiente batch e si applica una delle condizioni riportate di seguito, l'applicazione può connettersi a un gestore code z/OS remoto:

- V 9.3.4 LTS Il codice IBM MQ classes for Java è all'indirizzo IBM MQ 9.3.4 o successivo oppure Long Term Support con APAR PH56722 applicato. Il gestore code può essere di qualsiasi versione supportata.

- Il gestore code a cui ci si connette è in esecuzione con la titolarità IBM MQ Advanced for z/OS Value Unit Edition e pertanto il parametro **ADVCAP** è impostato su ENABLED. Il gestore code può essere di qualsiasi versione supportata.

Per ulteriori informazioni su IBM MQ Advanced for z/OS Value Unit Edition , consultare IBM MQ product identifier and export information.

Consultare DISPLAY QMGR per ulteriori informazioni su **ADVCAP** e START QMGR per ulteriori informazioni su **QMGRPROD**.

Un'applicazione IBM MQ classes for Java su z/OS non può utilizzare una connessione in modalità client per connettersi a un gestore code che non è in esecuzione su z/OS

Se un'applicazione IBM MQ classes for Java su z/OS tenta di connettersi utilizzando la modalità client e non è consentita, viene restituito MQRC_ENVIRONMENT_ERROR .

Supporto Advanced Message Security (AMS)

Le applicazioni client IBM MQ classes for Java possono utilizzare AMS durante la connessione ai gestori code z/OS remoti, in base alle condizioni precedentemente descritte in questo argomento.

Per utilizzare AMS in questo modo, le applicazioni client devono utilizzare un tipo di keystore `jceracfks` in `keystore.conf`, dove:

- Il prefisso del nome della proprietà è `jceracfks` e questo prefisso del nome non è sensibile al maiuscolo / minuscolo.
- Il keystore è un keyring RACF .
- Le password non sono richieste e verranno ignorate. Questo perché i keyring RACF non utilizzano le password.
- Se si specifica il fornitore, il fornitore deve essere `IBMJCE`.

Quando si utilizza `jceracfks` con AMS, il keystore deve essere nel formato: `safkeyring://`*`user`*`/`*`keyring`*, dove:

- `safkeyring` è un valore letterale e questo nome non è sensibile al maiuscolo / minuscolo
- *`user`* è l'ID utente RACF che possiede il keyring
- *`keyring`* è il nome del keyring RACF e il nome del keyring è sensibile al maiuscolo / minuscolo

Il seguente esempio utilizza il keyring standard AMS per l'utente `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Concetti correlati

“JMS/Jakarta Messaging connettività client alle applicazioni batch in esecuzione su z/OS” a pagina 128
In determinate condizioni, un'applicazione IBM MQ classes for JMS/Jakarta Messaging su z/OS può connettersi a un gestore code su z/OS utilizzando una connessione client. L'utilizzo di una connessione client può semplificare le topologie IBM MQ .

Scrittura di applicazioni IBM MQ classes for Java

Questa raccolta di argomenti fornisce informazioni utili per la scrittura di applicazioni Java per interagire con i sistemi IBM MQ .

Per utilizzare IBM MQ classes for Java per accedere a code IBM MQ , si scrivono applicazioni Java che contengono chiamate che inserono messaggi e ricevono messaggi da code IBM MQ . Per dettagli sulle singole classi, vedere IBM MQ classes for Java.

Nota: La riconnessione automatica del client non è supportata da IBM MQ classes for Java.

L'interfaccia IBM MQ classes for Java

L'interfaccia di programmazione dell'applicazione IBM MQ procedurale utilizza i verbi, che agiscono sugli oggetti. L'interfaccia di programmazione Java utilizza gli oggetti, su cui si agisce richiamando i metodi.

L'API (application programming interface) procedurale IBM MQ si basa su verbi come questi:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Questi verbi prendono tutti, come parametro, un handle per l'oggetto IBM MQ su cui devono operare. Il programma consiste in una serie di oggetti IBM MQ , su cui si agisce richiamando i metodi su tali oggetti.

Quando si utilizza l'interfaccia procedurale, ci si disconnette da un gestore code utilizzando la chiamata MQDISC (Hconn, CompCode, Reason), dove *Hconn* è un handle per il gestore code.

Nell'interfaccia Java , il gestore code è rappresentato da un oggetto della classe MQQueueManager. È possibile disconnettersi dal gestore code richiamando il metodo disconnect () su tale classe.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

Modalità di connessione IBM MQ classes for Java

Il modo in cui si programma per IBM MQ classes for Java ha alcune dipendenze sulle modalità di connessione che si desidera utilizzare.

Se si utilizzano le connessioni client, esistono diverse differenze rispetto a IBM MQ MQI client , ma è concettualmente simile. Se si utilizza la modalità di bind, è possibile utilizzare i bind del percorso rapido e immettere il comando MQBEGIN. Specificare la modalità da utilizzare impostando le variabili nella classe MQEnvironment.

IBM MQ classes for Java Connessioni client

Quando IBM MQ classes for Java viene utilizzato come un client, è simile a IBM MQ MQI client, ma presenta una serie di differenze.

Se si sta programmando per *IBM MQ classes for Java* da utilizzare come client, tenere presente le seguenti differenze:

- Supporta solo TCP/IP.
- Non legge alcuna variabile di ambiente IBM MQ all'avvio.
- Le informazioni memorizzate in una definizione di canale e nelle variabili di ambiente possono essere memorizzate in una classe denominata Ambiente. In alternativa, queste informazioni possono essere trasmesse come parametri quando viene effettuata la connessione.
- Le condizioni di errore e di eccezione vengono scritte in un log specificato nella classe `MQException` . La destinazione errori predefinita è la console Java .
- Solo i seguenti attributi in un file di configurazione del client IBM MQ sono rilevanti per IBM MQ classes for Java. Se si specificano altri attributi, essi sono inefficaci.

stanza	**Attributo**
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath64

stanza	Attributo
ClientExitStanza del percorso del file di configurazione client	JavaExitsClasspath
Stanza MessageBuffer del file di configurazione client	MaximumSize
Stanza MessageBuffer del file di configurazione client	PurgeTime
Stanza MessageBuffer del file di configurazione client	UpdatePercentage
Stanza TCP del file di configurazione client	ClntRcvBuffSize
Stanza TCP del file di configurazione client	ClntSndBuffSize
Stanza TCP del file di configurazione client	Timeout connessione
Stanza TCP del file di configurazione client	KeepAlive

- Se ci si connette a un gestore code che richiede la conversione dei dati carattere, il client V7 Java è ora in grado di eseguire la conversione se il gestore code non è in grado di eseguire tale operazione. La JVM client deve supportare la conversione tra il CCSID del client e quello del gestore code.
- La riconnessione automatica del client non è supportata da IBM MQ classes for Java.

Se utilizzato in modalità client, *IBM MQ classes for Java* non supporta la chiamata MQBEGIN.

Modalità bind IBM MQ classes for Java
La modalità di bind di IBM MQ classes for Java differisce dalla modalità del client in tre modi principali.

Quando viene utilizzato in modalità bind, IBM MQ classes for Java utilizza JNI (Java Native Interface) per richiamare direttamente l'API del gestore code esistente, piuttosto che comunicare attraverso una rete.

Per impostazione predefinita, le applicazioni che utilizzano IBM MQ classes for Java in modalità di bind si connettono a un gestore code utilizzando *ConnectOption*, MQCNO_STANDARD_BINDINGS.

IBM MQ classes for Java supporta le seguenti *ConnectOptions*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

Per ulteriori informazioni su *ConnectOptions*, consultare “Connessione a un gestore code utilizzando la chiamata MQCONNX” a pagina 738.

La modalità bind supporta la chiamata MQBEGIN per avviare le unità di lavoro globali coordinate dal gestore code, su tutte le piattaforme ad eccezione di IBM MQ for IBM i e IBM MQ for z/OS.

La maggior parte dei parametri forniti dalla classe `MQEnvironment` non sono rilevanti per la modalità di bind e vengono ignorati.

Definizione della connessione IBM MQ classes for Java da utilizzare
Il tipo di connessione da utilizzare è determinato dall'impostazione delle variabili nella classe MQEnvironment.

Vengono utilizzate due variabili:

MQEnvironment.properties
Il tipo di connessione è determinato dal valore associato con il nome chiave `CMQC.TRANSPORT_PROPERTY`. I valori possibili sono i seguenti:

CMQC.TRANSPORT_MQSERIES_BINDINGS
Connetti in modalità bind

CMQC.TRANSPORT_MQSERIES_CLIENT
Connetti in modalità client

CMQC.TRANSPORT_MQSERIES
La modalità di connessione è determinata dal valore della proprietà *hostname*

MQEnvironment.hostname
Impostare il valore di questa variabile come segue:

- Per le connessioni client, impostare il valore di questa variabile sul nome host del server IBM MQ a cui si desidera connettersi
- Per la modalità di bind, non impostare questa variabile o impostarla su null

Operazioni sui gestori code

Questa raccolta di argomenti descrive come connettersi e disconnettersi da un gestore code utilizzando IBM MQ classes for Java.

Configurazione dell'ambiente di IBM MQ per IBM MQ classes for Java
Perché un'applicazione si connetta a un gestore code in modalità client, l'applicazione deve specificare il nome canale, il nome host e il numero di porta.

Nota: Le informazioni contenute in questo argomento sono rilevanti solo se l'applicazione si connette a un gestore code in modalità client. Non Š rilevante se si connette in modalit ... bind. Consultare: “Modalità di connessione per IBM MQ classes for JMS” a pagina 112

È possibile specificare il nome del canale, il nome host e il numero di porta in due modi: come campi nella classe MQEnvironment o come proprietà dell'oggetto MQQueueManager .

Se si impostano i campi nella classe MQEnvironment, essi si applicano all'intera applicazione, tranne dove vengono sovrascritti da una tabella hash delle proprietà. Per specificare il nome canale e il nome host in MQEnvironment, utilizzare il seguente codice:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

Ciò equivale all'impostazione di una variabile di ambiente **MQSERVER** :

```
"java.client.channel/TCP/host.domain.com".
```

Per impostazione predefinita, i client Java tentano di connettersi a un listener IBM MQ sulla porta 1414. Per specificare una porta diversa, utilizzare il seguente codice:

```
MQEnvironment.port = nnnn;
```

dove nnnn è il numero di porta richiesto

Se si passano le proprietà a un oggetto gestore code al momento della sua creazione, esse si applicano solo a tale gestore code. Creare le voci in un oggetto Hashtable con chiavi di **hostname**, **channel**e, facoltativamente, **port**e con valori appropriati. Per utilizzare la porta predefinita 1414, è possibile omettere la voce **port** . Creare l'oggetto MQQueueManager utilizzando un costruttore che accetti la tabella hash delle proprietà.

Identificazione di una connessione al gestore code impostando un nome applicazione

Un'applicazione può impostare un nome che identifica la connessione al gestore code. Questo nome applicazione viene visualizzato dal comando **DISPLAY CONN MQSC/PCF** (dove il campo è denominato **APPLTAG**) o nella visualizzazione IBM MQ Explorer **Connessioni alle applicazioni** (dove il campo è denominato **App name**).

I nomi delle applicazioni sono limitati a 28 caratteri, quindi i nomi più lunghi vengono troncati. Se non viene specificato un nome applicazione, viene fornito un nome predefinito. Il nome predefinito si basa sulla classe di richiamo (principale), ma se queste informazioni non sono disponibili, viene utilizzato il testo `IBM MQ Client for Java` .

Se viene utilizzato il nome della classe di richiamo, viene adattato per adattarlo rimuovendo i nomi di pacchetto iniziali, se necessario. Ad esempio, se la classe di richiamo è `com.example.MainApp`, viene utilizzato il nome completo, ma se la classe di richiamo è `com.example.dictionaryAndThesaurus.multilingual.mainApp`, viene utilizzato il nome `multilingual.mainApp` , poiché è la combinazione più lunga di nome classe e nome pacchetto più a destra che si adatta alla lunghezza disponibile.

Se il nome della classe è più lungo di 28 caratteri, viene troncato per adattarlo. Ad esempio, `com.example.mainApplicationForSecondTestCase` diventa `mainApplicationForSecondTest`.

Per impostare un nome applicazione nella classe MQEnvironment, aggiungere il nome alla tabella hash MQEnvironment.properties , con chiave **`MQConstants.APPNAME_PROPERTY`**, utilizzando il seguente codice:

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

Per impostare un nome dell'applicazione nella tabella hash delle proprietà passata al costruttore MQQueueManager , aggiungere il nome alla tabella hash delle proprietà con una chiave **`MQConstants.APPNAME_PROPERTY`**.

Sovrascrittura delle proprietà specificate in un file di configurazione client IBM MQ

Un file di configurazione del client IBM MQ può specificare anche le proprietà utilizzate per configurare IBM MQ classes for Java. Tuttavia, le proprietà specificate in un file di configurazione IBM MQ MQI client si applicano solo quando un'applicazione si connette a un gestore code in modalità client.

Se necessario, è possibile sovrascrivere qualsiasi attributo in un file di configurazione IBM MQ in uno dei seguenti modi. Le opzioni vengono visualizzate in ordine di precedenza.

- Impostare una proprietà di sistema Java per la proprietà di configurazione.
- Impostare la proprietà nella mappa MQEnvironment.properties .
- Su Java5 e release successivi, impostare una variabile di ambiente di sistema.

Solo i seguenti attributi in un file di configurazione del client IBM MQ sono rilevanti per IBM MQ classes for Java. Se si specificano o si sovrascrivono altri attributi, non ha alcun effetto.

stanza	**Attributo**
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath
ClientExitStanza del percorso del file di configurazione client	ExitsDefaultPath64
ClientExitStanza del percorso del file di configurazione client	JavaExitsClasspath
Stanza MessageBuffer del file di configurazione client	MaximumSize
Stanza MessageBuffer del file di configurazione client	PurgeTime
Stanza MessageBuffer del file di configurazione client	UpdatePercentage
Stanza TCP del file di configurazione client	ClntRcvBufSize

stanza	Attributo
Stanza TCP del file di configurazione client	ClntSndBufSize
Stanza TCP del file di configurazione client	Timeout connessione
Stanza TCP del file di configurazione client	KeepAlive

Connessione a un gestore code in IBM MQ classes for Java
Connettersi a un gestore code creando una nuova istanza della classe MQQueueManager . Disconnettersi da un gestore code richiamando il metodo disconnect ().

Sei ora pronto a connetterti a un gestore code creando una nuova istanza della classe MQQueueManager :

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Per disconnettersi da un gestore code, richiamare il metodo disconnect () sul gestore code:

```
queueManager.disconnect();
```

Se si richiama il metodo di disconnessione, tutte le code aperte e i processi a cui è stato effettuato l'accesso tramite tale gestore code vengono chiusi. Tuttavia, è buona pratica di programmazione chiudere esplicitamente queste risorse quando si finisce di utilizzarle. Per eseguire questa operazione, utilizzare il metodo close () sugli oggetti pertinenti.

I metodi commit () e backout () su un gestore code sono equivalenti ai richiami MQCMIT e MQBACK utilizzati con l'interfaccia procedurale.

Utilizzo di una tabella di definizione di canale client con IBM MQ classes for Java
Un'applicazione client IBM MQ classes for Java può utilizzare le definizioni di canale di connessione client memorizzate in una CCDT (client channel definition table).

Come alternativa alla creazione di una definizione di canale di connessione client impostando determinati campi e proprietà di ambiente nella classe `MQEnvironment` o trasmettendoli a un `MQQueueManager` in una tabella hash delle proprietà, un'applicazione client IBM MQ classes for Java può utilizzare le definizioni di canale di connessione client memorizzate in una tabella di definizione di canale client. Queste definizioni vengono create dai comandi MQSC (IBM MQ Script) o PCF (IBM MQ Programmable Command Format) oppure utilizzando IBM MQ Explorer .

Quando l'applicazione crea un oggetto `MQQueueManager` , il client IBM MQ classes for Java ricerca nella tabella di definizione di canale client una definizione di canale di connessione client adatta e utilizza la definizione di canale per avviare un canale MQI. Per ulteriori informazioni sulle tabelle di definizione dei canali client e su come crearne una, consultare Tabella di definizione dei canali client.

Per utilizzare una tabella di definizione di canale client, un'applicazione deve prima creare un oggetto URL . L'oggetto URL incapsula un URL (Uniform Resource Locator) (URL) che identifica il nome e l'ubicazione del file contenente la tabella di definizione del canale client e specifica come è possibile accedere al file.

Ad esempio, se il file `ccdt1.tab` contiene una tabella di definizione del canale client ed è memorizzato sullo stesso sistema su cui è in esecuzione l'applicazione, l'applicazione può creare un oggetto URL nel modo seguente:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

Come altro esempio, si supponga che il file `ccdt2.tab` contenga una tabella di definizione del canale client e che sia memorizzato su un sistema diverso da quello su cui è in esecuzione l'applicazione. Se è

possibile accedere al file utilizzando il protocollo FTP, l'applicazione può creare un oggetto URL nel modo seguente:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Una volta creato un oggetto URL , l'applicazione può creare un oggetto `MQQueueManager` utilizzando uno dei costruttori che utilizza un oggetto URL come parametro. Di seguito è riportato un esempio:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Questa istruzione fa sì che il client IBM MQ classes for Java acceda alla tabella di definizioni del canale client identificata dall'oggetto URL chanTab2, ricerchi nella tabella una definizione del canale di connessione client adatta e quindi utilizzi la definizione del canale per avviare un canale MQI sul gestore code denominato MARS.

Notare i seguenti punti che si applicano se un'applicazione utilizza una tabella di definizione del canale client:

- Quando l'applicazione crea un oggetto `MQQueueManager` utilizzando un costruttore che utilizza un oggetto URL come parametro, non è necessario impostare alcun nome di canale nella classe `MQEnvironment` , come campo o come proprietà di ambiente. Se è impostato un nome canale, il client IBM MQ classes for Java genera un `MQException`. La proprietà del campo o dell'ambiente che specifica il nome del canale viene considerata impostata se il suo valore è diverso da null, una stringa vuota o una stringa contenente tutti caratteri vuoti.
- Il parametro **queueManagerName** nel costruttore `MQQueueManager` può avere uno dei seguenti valori:
  - Il nome di un gestore code
  - Un asterisco (*) seguito dal nome di un gruppo di gestori code
  - Un asterisco (*)
  - Null, una stringa vuota o una stringa contenente tutti i caratteri vuoti

  Si tratta degli stessi valori che possono essere utilizzati per il parametro **QMgrName** in una chiamata MQCONN emessa da un'applicazione client che utilizza MQI (Message Queue Interface). Per ulteriori informazioni sul significato di questi valori, consultare“Panoramica su Message Queue Interface” a pagina 722.

  Se l'applicazione utilizza il pool di connessioni, consultare“Controllo del pool di connessioni predefinito in IBM MQ classes for Java” a pagina 400.
- Quando il client IBM MQ classes for Java trova una definizione di canale di connessione client adatta nella tabella di definizioni di canali client, utilizza solo le informazioni estratte da questa definizione di canale per avviare un canale MQI. Tutti i campi relativi al canale o le proprietà dell'ambiente che l'applicazione potrebbe aver impostato nella classe `MQEnvironment` vengono ignorate.

  In particolare, notare i seguenti punti se si utilizza TLS (Transport Layer Security):
  - Un canale MQI utilizza TLS solo se la definizione del canale estratta dalla tabella di definizione del canale client specifica il nome di una CipherSpec supportata dal client IBM MQ classes for Java .
  - Una tabella di definizione del canale client contiene anche informazioni sull'ubicazione dei server LDAP (Lightweight Directory Access Protocol) che contengono i CRL (Certificate Revocation List). Il client IBM MQ classes for Java utilizza solo queste informazioni per accedere ai server LDAP che contengono i CRL.
  - Una tabella di definizione del canale client può contenere anche la posizione di un responder OCSP. IBM MQ classes for Java non è in grado di utilizzare le informazioni OCSP contenute in un file della tabella di definizione di canale client. Tuttavia, è possibile configurare OCSP come descritto nella sezione Utilizzo di Online Certificate Protocol

  Per ulteriori informazioni sull'utilizzo di TLS con una tabella di definizione del canale client, consultare Specifica che un canale MQI utilizza TLS.

Notare anche i seguenti punti se si utilizzano le uscite del canale:

- Un canale MQI utilizza le uscite canale e i dati utente associati specificati dalla definizione del canale estratta dalla tabella di definizione del canale client, preferendo le uscite canale e i dati specificati utilizzando altri metodi.
- Una definizione di canale estratta da una tabella di definizione di canale client può specificare le uscite di canale scritte in Java, C o C + +. Per ulteriori informazioni su come scrivere un'uscita di canale in Java , consultare"Creazione di un'uscita di canale in IBM MQ classes for Java" a pagina 394. Per ulteriori informazioni su come scrivere un'uscita canale in altre lingue, consultare "Utilizzo delle uscite canale non scritte in Java con IBM MQ classes for Java" a pagina 397.

Specifica di un intervallo di porte per connessioni client IBM MQ classes for Java
È possibile specificare una porta o un intervallo di porte a cui un'applicazione può collegarsi in uno dei due modi.

Quando un'applicazione IBM MQ classes for Java tenta di connettersi a un gestore code IBM MQ in modalità client, un firewall potrebbe consentire solo le connessioni che hanno origine da porte specificate o da un intervallo di porte. In questa situazione, è possibile specificare una porta o un intervallo di porte a cui l'applicazione può collegarsi. È possibile specificare le porte nei modi seguenti:

- È possibile impostare il campo di impostazione localAddressnella classe MQEnvironment. Di seguito è riportato un esempio:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- È possibile impostare la proprietà di ambiente CMQC.LOCAL_ADDRESS_PROPERTY. Di seguito è riportato un esempio:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Quando è possibile creare l'oggetto MQQueueManager , è possibile passare una tabella di hash delle proprietà contenente una LOCAL_ADDRESS_PROPERTY con il valore "192.0.2.0(2000,3000)"

In ognuno di questi esempi, quando l'applicazione si connette successivamente a un gestore code, l'applicazione si collega a un indirizzo IP locale e a un numero di porta compresi nell'intervallo tra 192.0.2.0(2000) e 192.0.2.0(3000).

In un sistema con più di un'interfaccia di rete, è anche possibile utilizzare il campo Impostazione localAddresso la proprietà di ambiente CMQC.LOCAL_ADDRESS_PROPERTY, per specificare quale interfaccia di rete deve essere utilizzata per una connessione.

Gli errori di connessione potrebbero verificarsi se si limita l'intervallo di porte. Se si verifica un errore, viene generata una MQException contenente il codice motivo IBM MQ MQRC_Q_MGR_NOT_AVAILABLE e il seguente messaggio:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Si potrebbe verificare un errore se tutte le porte nell'intervallo specificato sono in uso o se l'indirizzo IP, il nome host o il numero di porta specificati non sono validi (ad esempio, un numero di porta negativo).

Accesso a code, argomenti e processi in IBM MQ classes for Java

Per accedere a code, argomenti e processi, utilizzare i metodi della classe MQQueueManager . MQOD (object descriptor structure) è compresso nei parametri di questi metodi.

Code

Per aprire una coda è possibile utilizzare il metodo accessQueue della classe MQQueueManager . Ad esempio, su un gestore code denominato queueManager, utilizzare il seguente codice:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

Il metodo accessQueue restituisce un nuovo oggetto della classe MQQueue.

Una volta terminato di utilizzare la coda, utilizzare il metodo close () per chiuderla, come nel seguente esempio:

```
queue.close();
```

È anche possibile creare una coda utilizzando il costruttore MQQueue. I parametri sono esattamente gli stessi del metodo accessQueue , con l'aggiunta di un parametro del gestore code. Ad esempio:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

È possibile specificare un certo numero di opzioni quando si creano code. Per i dettagli, consultare Class.com.ibm.mq.MQQueue. La creazione di un oggetto coda in questo modo consente di scrivere le proprie sottoclassi di MQQueue.

Argomenti

Allo stesso modo, è possibile aprire un topic utilizzando il metodo accessTopic della classe MQQueueManager . Ad esempio, su un gestore code denominato queueManager, utilizzare il seguente codice per creare un sottoscrittore e un publisher:

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

Una volta terminato di utilizzare l'argomento, utilizzare il metodo close () per chiuderlo.

È anche possibile creare un argomento utilizzando il costruttore MQTopic. I parametri sono esattamente gli stessi del metodo accessTopic , con l'aggiunta di un parametro del gestore code. Ad esempio:

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

È possibile specificare un certo numero di opzioni quando si creano argomenti. Per i dettagli, consultare Class com.ibm.mq.MQTopic. La creazione di un oggetto argomento in questo modo consente di scrivere le proprie sottoclassi di MQTopic.

Un argomento deve essere aperto per la pubblicazione o per la sottoscrizione. La classe MQQueueManager ha otto metodi accessTopic e la classe Topic ha otto costruttori. In ogni caso, quattro di questi hanno un parametro **destination** e quattro hanno un parametro **subscriptionName** (inclusi due che hanno entrambi). Questi possono essere utilizzati solo per aprire l'argomento per le sottoscrizioni. I due metodi restanti hanno un parametro **openAs** e l'argomento può essere aperto per la pubblicazione o la sottoscrizione a seconda del valore del parametro **openAs** .

Per creare un argomento come sottoscrittore durevole, utilizzare un metodo accessTopic della classe MQQueueManager o un costruttore MQTopic che accetta un nome di sottoscrizione e, in entrambi i casi, impostare CMQC.MQSO_DURABLE .

Processi

Per accedere a un processo, utilizzare il metodo accessProcess di MQQueueManager. Ad esempio, su un gestore code denominato queueManager, utilizzare il codice riportato di seguito per creare un oggetto MQProcess:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Per accedere a un processo, utilizzare il metodo accessProcess di MQQueueManager.

Il metodo accessProcess restituisce un nuovo oggetto della classe MQProcess.

Una volta terminato di utilizzare l'oggetto processo, utilizzare il metodo close () per chiuderlo, come nel seguente esempio:

```
process.close();
```

È anche possibile creare un processo utilizzando il costruttore MQProcess. I parametri sono esattamente gli stessi del metodo accessProcess , con l'aggiunta di un parametro del gestore code. Ad esempio:

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

La costruzione di un oggetto processo in questo modo consente di scrivere le proprie sottoclassi di MQProcess.

Gestione dei messaggi in IBM MQ classes for Java

I messaggi sono rappresentati dalla classe MQMessage. I messaggi vengono inseriti e ricevuti utilizzando i metodi della classe MQDestination, che dispone di sottoclassi MQQueue e MQTopic.

Inserire i messaggi nelle code o negli argomenti utilizzando il metodo put () della classe MQDestination. I messaggi vengono ricevuti dalle code o dagli argomenti utilizzando il metodo get () della classe MQDestination. A differenza dell'interfaccia procedurale, dove MQPUT e MQGET immettono e ottengono matrici di byte, il linguaggio di programmazione Java immette e richiama le istanze della classe MQMessage. La classe MQMessage incapsula il buffer di dati che contiene i dati del messaggio effettivi, insieme a tutti i parametri MQMD (descrittore del messaggio) e le proprietà del messaggio che descrivono tale messaggio.

Per generare un nuovo messaggio, creare una nuova istanza della classe MQMessage e utilizzare i metodi writeXXX per inserire i dati nel buffer del messaggio.

Quando viene creata la nuova istanza di messaggio, tutti i parametri MQMD vengono impostati automaticamente sui loro valori predefiniti, come definito in Valori iniziali e dichiarazioni della lingua per MQMD. Il metodo put () di MQDestination utilizza anche un'istanza della classe di opzioni MQPutMessagecome parametro. Questa classe rappresenta la struttura MQPMO. Il seguente esempio crea un messaggio e lo inserisce in una coda:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage,pmo);
```

Il metodo get () di MQDestination restituisce una nuova istanza di MQMessage, che rappresenta il messaggio appena preso dalla coda. Inoltre, utilizza un'istanza della classe di opzioni MQGetMessagecome parametro. Questa classe rappresenta la struttura MQGMO.

Non è necessario specificare una dimensione massima del messaggio, poiché il metodo get () regola automaticamente la dimensione del buffer interno in modo che si adatti al messaggio in entrata. Utilizzare i metodi readXXX della classe MQMessage per accedere ai dati nel messaggio restituito.

Il seguente esempio mostra come ottenere un messaggio da una coda:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

È possibile modificare il formato numerico utilizzato dai metodi di lettura e scrittura impostando la variabile membro *encoding* .

È possibile modificare la serie di caratteri da utilizzare per la lettura e la scrittura di stringhe impostando la variabile membro *characterSet* .

Per ulteriori informazioni, consultare Classe MQMessage .

Nota: Il metodo writeUTF() di MQMessage codifica automaticamente la lunghezza della stringa e i byte Unicode che contiene. Quando il messaggio viene letto da un altro programma Java (utilizzando readUTF()), questo è il metodo più semplice per inviare le informazioni sulla stringa.

Miglioramento delle prestazioni dei messaggi non persistenti in IBM MQ classes for Java
Per migliorare le prestazioni durante l'esplorazione dei messaggi o l'utilizzo di messaggi non persistenti da un'applicazione client, è possibile utilizzare la *lettura anticipata*. Le applicazioni client che utilizzano MQGET o il consumo asincrono beneficeranno dei miglioramenti delle prestazioni durante l'esplorazione dei messaggi o l'utilizzo di messaggi non persistenti.

Per informazioni generali sulla funzione di lettura anticipata, consultare l'argomento correlato.

In IBM MQ classes for Java, si utilizza CMQC.MQSO_READ_AHEAD e CMQC.MQSO_NO_READ_AHEAD di un oggetto MQQueue o MQTopic per stabilire se i consumer dei messaggi e i browser delle code possono utilizzare la lettura anticipata su tale oggetto.

Inserimento asincrono dei messaggi utilizzando IBM MQ classes for Java
Per inserire un messaggio in modo asincrono, impostare MQPMO_ASYNC_RESPONSE.

I messaggi vengono inseriti nelle code o negli argomenti utilizzando il metodo put () della classe MQDestination. Per inserire un messaggio in modo asincrono, ossia consentendo il completamento dell'operazione senza attendere una risposta dal gestore code, è possibile impostare MQPMO_ASYNC_RESPONSE nel campo delle opzioni MQPutMessageOptions. Per determinare l'esito positivo o negativo delle operazioni di immissione asincrone, utilizzare la chiamata di stato MQQueueManager.getAsync.

Pubblicazione / sottoscrizione in IBM MQ classes for Java

In IBM MQ classes for Java, l'argomento è rappresentato dalla classe MQTopic e si pubblica su di esso utilizzando i metodi MQTopic.put().

Per informazioni generali sulla pubblicazione / sottoscrizione IBM MQ , consultare Messaggistica di pubblicazione / sottoscrizione.

Gestione delle intestazioni dei messaggi IBM MQ con IBM MQ classes for Java

Vengono fornite classi Java che rappresentano diversi tipi di intestazione del messaggio. Sono fornite anche due classi helper.

L'interfaccia MQHeader

Gli oggetti di intestazione sono descritti dall'interfaccia MQHeader, che fornisce metodi di uso generico per accedere ai campi di intestazione e per leggere e scrivere il contenuto del messaggio. Ogni tipo di intestazione ha la propria classe che implementa l'interfaccia MQHeader e aggiunge metodi getter e setter per i singoli campi. Ad esempio, il tipo di intestazione MQRFH2 è rappresentato dalla classe MQRFH2 ; il tipo di intestazione MQDLH dalla classe MQDLH e così via. Le classi di intestazione eseguono automaticamente qualsiasi conversione di dati necessaria e possono leggere o scrivere i dati in qualsiasi CCSID (serie di caratteri o codifica numerica specificata).

Importante: Le classi di intestazioni MQRFH2 trattano il messaggio come un file ad accesso casuale, il che significa che il cursore deve essere posizionato all'inizio del messaggio. Prima di utilizzare una classe di intestazione del messaggio interno come MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH o MQXMIT, assicurarsi di aggiornare la posizione del cursore del messaggio nella posizione corretta prima di passare il messaggio alla classe.

Classi helper

Due classi helper, MQHeaderIterator e MQHeaderList, assistono nella lettura e decodifica (analisi) del contenuto dell'intestazione nei messaggi:

- La classe MQHeaderIterator funziona come un java.util.Iterator. Finché ci sono più intestazioni nel messaggio, il metodo next () restituisce true e il metodo nextHeader() o next () restituisce l'oggetto intestazione successivo.
- MQHeaderList funziona come un java.util.List. Come MQHeaderIterator, analizza il contenuto dell'intestazione, ma consente anche di ricercare intestazioni particolari, aggiungere nuove intestazioni, rimuovere intestazioni esistenti, aggiornare i campi di intestazione e quindi scrivere il contenuto dell'intestazione in un messaggio. In alternativa, è possibile creare un MQHeaderListvuoto, quindi popolarlo con le istanze di intestazione e scriverlo in un messaggio una volta o ripetutamente.

Le classi MQHeaderIterator e MQHeaderList utilizzano le informazioni in MQHeaderRegistry per sapere quali classi di intestazioni IBM MQ sono associate a determinati tipi e formati di messaggi. MQHeaderRegistry viene configurato con la conoscenza di tutti i tipi di intestazione e formati IBM MQ correnti e delle relative classi di implementazione ed è anche possibile registrare i tipi di intestazione.

Il supporto viene fornito per le seguenti intestazioni IBM MQ comunemente utilizzate

- MQRFH - Regole e intestazione di formattazione
- MQRFH2 - Come MQRFH, utilizzato per trasmettere messaggi a e da un broker di messaggi appartenente a IBM Integration Bus. Utilizzato anche per contenere le proprietà del messaggio
- MQCIH - Bridge CICS
- MQDLH - Intestazione lettera non instradabile
- MQIIH - Intestazione informazioni IMS
- MQRMH - intestazione messaggio di riferimento
- MQSAPH - Intestazione SAP
- MQWIH - Intestazione informazioni di lavoro
- MQXQH - Intestazione Coda di trasmissione
- MQDH - Intestazione di distribuzione
- MQEPH - Intestazione PCF incapsulata

È anche possibile definire classi che rappresentano le proprie intestazioni.

Per utilizzare un MQHeaderIterator per richiamare un'intestazione RFH2 , impostare MQGMO_PROPERTIES_FORCE_MQRFH2 nelle opzioni GetMessageoppure impostare la proprietà della coda PROPCTL su FORCE.

Stampa di tutte le intestazioni in un messaggio utilizzando IBM MQ classes for Java
In questo esempio, un'istanza di MQHeaderIterator analizza le intestazioni in un MQMessage ricevuto da una coda. Gli oggetti MQHeader restituiti dal metodo nextHeader() visualizzano la loro struttura e il loro contenuto quando viene richiamato il loro metodo toString .

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

Ignorare le intestazioni in un messaggio utilizzando IBM MQ classes for Java
In questo esempio, il metodo skipHeaders() di MQHeaderIterator posiziona il cursore di lettura del messaggio immediatamente dopo l'ultima intestazione.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

Ricerca del codice di errore in un messaggio non instradabile utilizzando IBM MQ classes for Java
In questo esempio, il metodo read popola l'oggetto MQDLH leggendo dal messaggio. Dopo l'operazione di lettura, il cursore di lettura del messaggio viene posizionato immediatamente dopo il contenuto dell'intestazione MQDLH.

I messaggi nella coda di messaggi non instradabili del gestore code hanno come prefisso un'intestazione di messaggi non instradabili (MQDLH). Per decidere come gestire questi messaggi - ad esempio, per determinare se riprovare o eliminarli - un'applicazione di gestione dei messaggi non recapitabili deve esaminare il codice di errore contenuto in MQDLH.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

Tutte le classi di intestazione forniscono anche un costruttore di convenienza per inizializzarsi direttamente dal messaggio in un singolo passo. Quindi, il codice in questo esempio potrebbe essere semplificato come segue:

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);
```

```
System.out.println ("Reason: " + dlh.getReason ());
```

Lettura e rimozione dell'intestazione da un messaggio non recapitabile utilizzando IBM MQ classes for Java
In questo esempio, MQDLH viene utilizzato per eliminare l'intestazione da un messaggio non instradabile.

Un'applicazione di gestione dei messaggi non instradabili in genere reinoltra i messaggi che sono stati rifiutati se il relativo codice motivo indica un errore transitorio. Prima di inoltrare nuovamente il messaggio, è necessario rimuovere l'intestazione MQDLH.

Questo esempio esegue la seguente procedura (vedere i commenti nel codice di esempio):

1. MQHeaderList legge l'intero messaggio e ogni intestazione rilevata nel messaggio diventa un elemento nell'elenco.
2. I messaggi non instradabili contengono un MQDLH come prima intestazione, quindi è possibile trovarlo nella prima voce dell'elenco di intestazioni. MQDLH è già stato popolato dal messaggio quando viene creato MQHeaderList , quindi non è necessario richiamarne il metodo di lettura.
3. Il codice motivo viene estratto utilizzando il metodo getReason() fornito dalla classe MQDLH.
4. Il codice di errore è stato ispezionato e indica che è appropriato inoltrare nuovamente il messaggio. MQDLH viene rimosso utilizzando il metodo MQHeaderList remove ().
5. MQHeaderList scrive il contenuto rimanente in un nuovo oggetto messaggio. Il nuovo messaggio ora contiene tutto il contenuto del messaggio originale tranne MQDLH e può essere scritto in una coda. L'argomento **true** per il costruttore e per il metodo di scrittura indica che il corpo del messaggio deve essere contenuto in MQHeaderListe scritto di nuovo.
6. Il campo formato nel descrizione del messaggio del nuovo messaggio ora contiene il valore che era precedentemente nel campo formato MQDLH. I dati del messaggio corrispondono alla codifica numerica e CCSID impostati nel descrittore del messaggio.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

Stampa del contenuto di un messaggio utilizzando IBM MQ classes for Java
In questo esempio viene utilizzato MQHeaderList per stampare il contenuto di un messaggio, incluse le intestazioni.

L'output contiene una vista di tutto il contenuto dell'intestazione e del testo del messaggio. La classe MQHeaderList decodifica tutte le intestazioni in una sola volta, mentre MQHeaderIterator le espone una alla volta sotto il controllo dell'applicazione. È possibile utilizzare questa tecnica per fornire uno strumento di debug semplice durante la scrittura di applicazioni WebSphere MQ .

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

Questo esempio stampa anche i campi descrittore del messaggio, utilizzando la classe MQMD. Il metodo copyFrom() della classe com.ibm.mq.headers.MQMD popola l'oggetto intestazione dai campi del descrittore del messaggio di MQMessage piuttosto che leggendo il contenuto del messaggio.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

Ricerca di un tipo specifico di intestazione in un messaggio utilizzando IBM MQ classes for Java
Questo esempio utilizza il metodo indexOf(String) di MQHeaderList per trovare un'intestazione MQRFH2 in un messaggio, se presente.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
        ...
}
```

Analisi di una intestazione MQRFH2 utilizzando IBM MQ classes for Java
Questo esempio mostra come accedere a un valore di campo noto in una cartella denominata utilizzando la classe MQRFH2 .

La classe MQRFH2 fornisce una serie di modi per accedere non solo ai campi nella parte fissa della struttura, ma anche al contenuto della cartella codificata XML contenuto nel campo NameValueData. Questo esempio mostra come accedere ad un valore di campo noto in una cartella denominata - in questa istanza, il campo Rto nella cartella jms, che rappresenta il nome della coda di risposta in un messaggio MQ JMS .

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

Per scoprire il contenuto di MQRFH2 (invece di richiedere campi specifici direttamente), è possibile utilizzare il metodo getFolders per restituire un elenco di MQRFH2.Element, che rappresenta la struttura di una cartella che potrebbe contenere campi e altre cartelle. L'impostazione di un campo o di una cartella su null lo rimuove da MQRFH2. Quando si manipola il contenuto della cartella dati NameValuein questo modo, il campo StrucLength viene aggiornato automaticamente.

Lettura e scrittura di flussi di byte diversi dagli oggetti MQMessage utilizzando IBM MQ classes for Java
Questi esempi utilizzano le classi di intestazione per analizzare e gestire il contenuto dell'intestazione IBM MQ quando l'origine dati non è un oggetto MQMessage.

È possibile utilizzare le classi di intestazione per analizzare e gestire il contenuto dell'intestazione IBM MQ anche quando l'origine dati è qualcosa di diverso da un oggetto MQMessage. L'interfaccia MQHeader implementata da ogni classe di intestazione fornisce i metodi `int read (java.io.DataInput message, int encoding, int characterSet)` e `int write (java.io.DataOutput message, int encoding, int characterSet)`. La classe com.ibm.mq.MQMessage implementa le interfacce java.io.DataInput e java.io.DataOutput . Ciò significa che è possibile utilizzare i due metodi MQHeader per leggere e scrivere il contenuto di MQMessage, sostituendo la codifica e il CCSID specificati nel descrittore del messaggio. È utile per i messaggi che contengono una catena di intestazioni in codifiche differenti.

È inoltre possibile ottenere gli oggetti DataInput e DataOutput da altri flussi di dati, ad esempio flussi di file o socket o array di byte trasportati nei messaggi JMS . Le classi java.io.DataInputStream implementano DataInput e le classi java.io.DataOutputStream implementano DataOutput. Questo esempio legge il contenuto dell'intestazione IBM MQ da un array di byte:

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

La riga che inizia con MQHeaderIterator potrebbe essere sostituita con

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

Questo esempio scrive in un array di byte utilizzando il flusso DataOutput:

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

Quando si utilizzano gli stream in questo modo, fare attenzione a utilizzare i valori corretti per gli argomenti di codifica e characterSet . Durante la lettura delle intestazioni, specificare la codifica e il CCSID con cui il contenuto byte è stato originariamente scritto. Durante la scrittura delle intestazioni, specificare la codifica e il CCSID che si desidera produrre. La conversione dei dati viene eseguita automaticamente dalle classi di intestazione.

Creazione di classi per nuovi tipi di intestazione utilizzando IBM MQ classes for Java
È possibile creare classi Java per tipi di intestazione non forniti con IBM MQ classes for Java.

Per aggiungere una classe Java che rappresenta un nuovo tipo di intestazione che è possibile utilizzare allo stesso modo di qualsiasi classe di intestazione fornita con IBM MQ classes for Java, creare una classe che implementi l'interfaccia MQHeader. L'approccio più semplice consiste nell'estendere la classe com.ibm.mq.headers.impl.Header . Questo esempio produce una classe completamente funzionale che rappresenta la struttura dell'intestazione MQTM. Non è necessario aggiungere singoli metodi getter e setter per ogni campo, ma si tratta di una convenienza utile per gli utenti della classe intestazione. I metodi getValue e setValue generici che utilizzano una stringa per il nome campo funzioneranno per tutti i campi definiti nel tipo di intestazione. I metodi di lettura, scrittura e dimensione ereditati consentono alle istanze del nuovo tipo di intestazione di essere lette e scritte e calcolano correttamente la dimensione dell'intestazione in base alla definizione del campo. La definizione del tipo viene creata solo una volta, tuttavia vengono create molte istanze di questa classe di intestazione. Per rendere la nuova definizione di intestazione disponibile per la decodifica utilizzando le classi MQHeaderIterator o MQHeaderList , è necessario registrarla utilizzando MQHeaderRegistry. Notare, tuttavia, che la classe di intestazione MQTM è già fornita in questo pacchetto e registrata nel registro predefinito.

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
```

```
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);

    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

Gestione dei messaggi PCF con IBM MQ classes for Java

Le classi Java vengono fornite per creare e analizzare messaggi strutturati PCF e per facilitare l'invio di richieste PCF e la raccolta di risposte PCF.

Le classi PCFMessage & MQCFGR rappresentano schiere di strutture di parametri PCF. Forniscono metodi di convenienza per aggiungere e richiamare i parametri PCF.

Le strutture dei parametri PCF sono rappresentate dalle classi MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL e MQCFGR. Queste condividono le interfacce operative di base:

- Metodi per leggere e scrivere il contenuto del messaggio: read (), write () e size ()
- Metodi per manipolare i parametri getValue (), setValue (), getParameter () e altri
- Il metodo enumerator.nextParameter (), che analizza il contenuto PCF in un MQMessage

Il parametro di filtro PCF viene utilizzato nei comandi inquire per fornire una funzione di filtro. Esso è incapsulato nelle seguenti classi:

- MQCFIF - filtro numero intero
- MQCFSF - filtro stringa
- MQCFBF - filtro byte

Vengono fornite due classi di agent, PCFagent e PCFMessageAgent , per gestire la connessione a un gestore code, la coda del server dei comandi e una coda di risposta associata. PCFMessageAgent estende PCFAgent e deve essere normalmente utilizzato di preferenza. La classe PCFMessageAgent converte i messaggi MQMessage ricevuti e li restituisce al chiamante come un array PCFMessage. PCFAgent restituisce un array di messaggi MQMessage, che è necessario analizzare prima dell'utilizzo.

Gestione delle proprietà del messaggio in IBM MQ classes for Java

Le chiamate di funzione per elaborare gli handle del messaggio non hanno un equivalente in IBM MQ classes for Java. Per impostare, restituire o eliminare le proprietà dell'handle del messaggio, utilizzare i metodi della classe MQMessage.

Per informazioni generali sulle proprietà dei messaggi, consultare “Nomi proprietà” a pagina 28.

In IBM MQ classes for Java l'accesso ai messaggi avviene tramite la classe MQMessage. Gli handle dei messaggi non vengono pertanto forniti in ambiente Java e non esiste alcun equivalente alle chiamate di funzione IBM MQ MQCRTMH, MQDLTMH, MQMHBUF e MQBUFMH

Per impostare le proprietà della gestione messaggi nell'interfaccia procedurale, utilizzare la chiamata MQSETMP. In IBM MQ classes for Java, utilizzare il metodo appropriato della classe MQMessage:

- Proprietà setBoolean
- Proprietà setByte
- Proprietà setBytes

- Proprietà setShort
- Proprietà setInt
- setInt2Property
- setInt4Property
- setInt8Property
- Proprietà setLong
- Proprietà setFloat
- Proprietà setDouble
- Proprietà setString
- Proprietà setObject

Talvolta si fa riferimento a tali metodi collettivamente come metodi *set*property* .

Per restituire il valore delle proprietà dell'handle del messaggio nell'interfaccia procedurale, utilizzare la chiamata MQINQMP. In IBM MQ classes for Java, utilizzare il metodo appropriato della classe MQMessage:

- Proprietà getBoolean
- Proprietà getByte
- Proprietà getBytes
- Proprietà getShort
- Proprietà getInt
- getInt2Property
- getInt4Property
- getInt8Property
- Proprietà getLong
- Proprietà getFloat
- Proprietà getDouble
- Proprietà getString
- Proprietà getObject

Talvolta si fa riferimento a tali metodi collettivamente come ai metodi *get*property* .

Per eliminare il valore delle proprietà dell'handle del messaggio nell'interfaccia procedurale, utilizzare la chiamata MQDLTMP. In IBM MQ classes for Java, utilizzare il metodo deleteProperty della classe MQMessage.

Gestione degli errori in IBM MQ classes for Java

Gestire gli errori derivanti dall' IBM MQ classes for Java utilizzo dei blocchi Java `try` e `catch` .

I metodi nell'interfaccia Java non restituiscono un codice di completamento e un codice motivo. Al contrario, generano un'eccezione ogni volta che il codice di completamento e il codice motivo risultanti da una chiamata IBM MQ non sono entrambi zero. Ciò semplifica la logica del programma in modo che non sia necessario controllare i codici di ritorno dopo ogni chiamata a IBM MQ. È possibile decidere in quali punti del programma si desidera affrontare la possibilità di errore. In questi punti, puoi racchiudere il tuo codice con blocchi `try` e `catch` , come nel seguente esempio:

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
```

```
                            "CC = " + ex.completionCode +
                            "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

I codici motivo della chiamata IBM MQ riportati nelle eccezioni Java per z/OS sono documentati in Codici motivo e completamento API.

Le eccezioni generate durante l'esecuzione di un'applicazione IBM MQ classes for Java vengono anche scritte nel log. Tuttavia, un'applicazione può richiamare il metodo MQException.logExclude() per impedire la registrazione delle eccezioni associate a uno specifico codice motivo. È possibile eseguire questa operazione in situazioni in cui si prevedono molte eccezioni associate a uno specifico codice di errore e non si desidera che il log venga riempito con tali eccezioni. Ad esempio, se l'applicazione tenta di richiamare un messaggio da una coda ogni volta che esegue l'iterazione di un loop e, per la maggior parte di questi tentativi, si prevede che non vi sia alcun messaggio adatto nella coda, è possibile impedire la registrazione delle eccezioni associate al codice di errore MQRC_NO_MSG_AVAILABLE. Se un'applicazione ha precedentemente impedito la registrazione delle eccezioni associate a un codice motivo specifico, può consentire la registrazione di tali eccezioni richiamando il metodo MQException.logInclude().

A volte il codice di errore non trasmette tutti i dettagli associati all'errore. Per ogni eccezione generata, un'applicazione deve controllare l'eccezione collegata. L'eccezione collegata stessa potrebbe avere un'altra eccezione collegata, e quindi le eccezioni collegate formano una catena che riporta al problema sottostante originale. Un'eccezione collegata viene implementata utilizzando il meccanismo di eccezione concatenato della classe java.lang.Throwable , e un'applicazione ottiene un'eccezione collegata richiamando il metodo Throwable.getCause(). Da un'eccezione che è un'istanza MQException, MQException.getCause() richiama l'istanza sottostante di com.ibm.mq.jmqi.JmqiExceptione getCause da questa eccezione richiama l' java.lang.Exception sottostante che ha causato l'errore.

Richiamo e impostazione dei valori di attributo in IBM MQ classes for Java

I metodi getXXX() e setXXX() vengono forniti per molti attributi comuni. È possibile accedere ad altri utilizzando i metodi inquire () e set () generici.

Per molti degli attributi comuni, le classi MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess e MQQueueManager contengono metodi getXXX() e setXXX(). Questi metodi consentono di ottenere e impostare i loro valori di attributi. Tenere presente che per MQDestination, MQQueue e MQTopic, i metodi funzionano solo se si specificano gli indicatori inquire e set appropriati quando si apre l'oggetto.

Per gli attributi meno comuni, le classi MQQueueManager, MQDestination, MQQueue, MQTopic e MQProcess ereditano tutte da una classe denominata MQManagedObject. Questa classe definisce le interfacce inquire () e set ().

Quando si crea un nuovo oggetto gestore code utilizzando il *nuovo* operatore, viene automaticamente aperto per l'interrogazione. Quando si utilizza il metodo accessProcess() per accedere a un oggetto processo, tale oggetto viene automaticamente aperto per l'interrogazione. Quando si utilizza il metodo accessQueue() per accedere a un oggetto coda, tale oggetto non viene automaticamente aperto per le operazioni di interrogazione o di impostazione. Questo perché l'aggiunta automatica di queste opzioni può causare problemi con alcuni tipi di code remote. Per utilizzare i metodi inquire, set, getXXXe setXXX su una coda, è necessario specificare gli indicatori inquire e set appropriati nel parametro openOptions del metodo accessQueue(). Lo stesso vale per gli oggetti di destinazione e argomento.

I metodi inquire e set utilizzano tre parametri:

- array selettori
- array intAttrs
- Array charAttrs

Non sono necessari i parametri SelectorCount, IntAttrCount e CharAttrLength rilevati in MQINQ, poiché la lunghezza di un array in Java è sempre nota. Il seguente esempio mostra come eseguire un'interrogazione su una coda:

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Programmi multithread in Java

L'ambiente di runtime Java è intrinsecamente multithread. IBM MQ classes for Java consente a un oggetto del gestore code di essere condiviso da più thread, ma garantisce che tutti gli accessi al gestore code di destinazione siano sincronizzati.

I programmi multithread sono difficili da evitare in Java. Considerare un programma semplice che si connette a un gestore code e apre una coda all'avvio. Il programma visualizza un singolo pulsante sullo schermo. Quando un utente fa clic su quel pulsante, il programma richiama un messaggio dalla coda.

L'ambiente di runtime Java è intrinsecamente multithread. Pertanto, l'inizializzazione dell'applicazione avviene in un thread e il codice che viene eseguito in seguito alla pressione del pulsante viene eseguito in un thread separato (il thread dell'interfaccia utente).

Con IBM MQ MQI clientbasato su C, ciò causerebbe un problema, poiché esistono limitazioni alla condivisione di handle da parte di più thread. IBM MQ classes for Java rimuove questo vincolo, consentendo a un oggetto del gestore code (e agli oggetti coda, argomento e processo associati) di essere condiviso da più thread.

L'implementazione di IBM MQ classes for Java garantisce che, per una particolare connessione (istanza dell'oggettoMQQueueManager), tutti gli accessi al gestore code IBM MQ di destinazione siano sincronizzati. Un thread che desidera emettere una chiamata a un gestore code viene bloccato finché non vengono completate tutte le altre chiamate in corso per tale connessione. Se si richiede l'accesso simultaneo allo stesso gestore code da più thread all'interno del programma, creare un nuovo oggetto MQQueueManager per ogni thread che richiede l'accesso simultaneo. (Ciò equivale all'emissione di una chiamata MQCONN separata per ciascun thread.)

Nota: Le istanze della classe `com.ibm.mq.MQGetMessageOptions` non devono essere condivise tra i thread che richiedono messaggi contemporaneamente. Le istanze di questa classe vengono aggiornate con i dati durante la richiesta MQGET corrispondente, il che può causare conseguenze impreviste quando più thread operano contemporaneamente nella stessa istanza dell'oggetto.

Utilizzo delle uscite di canale in IBM MQ classes for Java

Una panoramica su come utilizzare le uscite di canale in un'applicazione utilizzando IBM MQ classes for Java.

I seguenti argomenti descrivono come scrivere un'uscita del canale in Java, come assegnarla e come passare i dati ad essa. Descrivono quindi come utilizzare le uscite canale scritte in C e come utilizzare una sequenza di uscite canale.

L'applicazione deve disporre dell'autorizzazione di protezione corretta per caricare la classe di uscita canale.

Creazione di un'uscita di canale in IBM MQ classes for Java
È possibile fornire le proprie uscite canale definendo una classe Java che implementa un'interfaccia appropriata.

Per implementare un'uscita, definire una nuova classe Java che implementi l'interfaccia appropriata. Tre interfacce di uscita sono definite nel pacchetto com.ibm.mq.exits :

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

Nota: Le uscite del canale sono supportate solo per connessioni client; non sono supportate per connessioni di bind. Non è possibile utilizzare un' Java uscita del canale all'esterno di IBM MQ classes for Java, ad esempio se si utilizza un'applicazione client scritta in C.

Qualsiasi crittografia TLS definita per una connessione viene eseguita *dopo* che sono state richiamate le uscite di invio e di sicurezza. Analogamente, la decrittografia viene eseguita *prima che* vengano richiamate le uscite di sicurezza e di ricezione.

L'esempio riportato di seguito definisce una classe che implementa tutte e tre le interfacce:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                  MQCD channelDefinition,
                                  ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                     MQCD channelDefinition,
                                     ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                      MQCD channelDefinition,
                                      ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

Ad ogni uscita viene passato un oggetto MQCXP e un oggetto MQCD. Questi oggetti rappresentano le strutture MQCXP e MQCD definite nell'interfaccia procedurale.

Qualsiasi classe di uscita scritta deve avere un costruttore. Può essere il costruttore predefinito o uno che utilizza un argomento stringa. Se richiede una stringa, i dati utente verranno passati nella classe di uscita quando vengono creati. Se la classe di uscita contiene sia un costruttore predefinito che un costruttore di argomento singolo, il costruttore di argomento singolo ha la priorità.

Per le uscite di invio e di sicurezza, il codice di uscita deve restituire i dati che si desidera inviare al server. Per un'uscita di ricezione, il codice di uscita deve restituire i dati modificati che si desidera che IBM MQ interpreti.

Il corpo di uscita più semplice possibile è:

```
{ return agentBuffer; }
```

Non chiudere il gestore code dall'interno di un'uscita canale.

Utilizzo delle classi di uscita canale esistenti

Nelle versioni di IBM MQ precedenti a 7.0, è necessario implementare queste uscite utilizzando le interfacce MQSendExit, MQReceiveExite MQSecurityExit, come nel seguente esempio. Questo metodo rimane valido, ma il nuovo metodo è preferito per migliorare la funzionalità e le prestazioni.

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

Assegnazione di un'uscita canale in IBM MQ classes for Java
È possibile assegnare un'uscita canale utilizzando IBM MQ classes for Java.

Non esiste un equivalente diretto del canale IBM MQ in IBM MQ classes for Java. Le uscite dei canali vengono assegnate a un MQQueueManager. Ad esempio, dopo aver definito una classe che implementa l'interfaccia WMQSecurityExit , un'applicazione può utilizzare l'uscita di sicurezza in quattro modi:

- Assegnando un'istanza della classe al campo MQEnvironment.channelSecurityExit prima di creare un oggetto MQQueueManager
- Impostando il campo MQEnvironment.channelSecurityExit su una stringa che rappresenta la classe di uscita di sicurezza prima di creare un oggetto MQQueueManager
- Creando una coppia chiave / valore nell'hashtable delle proprietà passata a MQQueueManager con una chiave di CMQC.SECURITY_EXIT_PROPERTY
- Utilizzo di una tabella di definizione del canale client (CCDT)

Qualsiasi uscita assegnata impostando il campo MQEnvironment.channelSecurityExit su una stringa, creando una coppia chiave / valore nell'hashtable delle proprietà o utilizzando una CCDT, deve essere scritta con un costruttore predefinito. Un'uscita assegnata come istanza di una classe non necessita di un costruttore predefinito, a seconda dell'applicazione.

Un'applicazione può utilizzare un'uscita di invio o di ricezione in modo simile. Ad esempio, il seguente frammento di codice mostra come utilizzare le uscite di sicurezza, di invio e di ricezione implementate nella classe MyMQExits, definita in precedenza, utilizzando MQEnvironment:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Se viene utilizzato più di un metodo per assegnare un'uscita canale, l'ordine di precedenza è il seguente:

1. Se l' URL di una CCDT viene passato a MQQueueManager, il contenuto della CCDT determina le uscite del canale da utilizzare e tutte le definizioni di uscita in MQEnvironment o nella tabella hash delle proprietà vengono ignorate.
2. Se non viene passato alcun URL CCDT, le definizioni di uscita da MQEnvironment e l'hashtable vengono unite
   - Se lo stesso tipo di uscita è definito sia in MQEnvironment che nella tabella hash, viene utilizzata la definizione nella tabella hash.
   - Se vengono specificati nuovi e vecchi tipi equivalenti di uscita (ad esempio, il campo sendExit , che può essere utilizzato solo per il tipo di uscita utilizzato nelle versioni precedenti a IBM WebSphere MQ 7.0, e il campo channelSendExit, che può essere utilizzato per qualsiasi uscita di invio), viene utilizzata la nuova uscita (channelSendExit) anziché la vecchia uscita.

Se è stata dichiarata un'uscita canale come stringa, è necessario abilitare IBM MQ per individuare il programma di uscita canale. È possibile eseguire questa operazione in vari modi, a seconda dell'ambiente in cui l'applicazione è in esecuzione e della modalità di impacchettatura dei programmi di uscita del canale.

- Per un'applicazione in esecuzione in un server delle applicazioni, è necessario memorizzare i file nella directory mostrata in Tabella 58 a pagina 396 o impacchettarli nei file JAR a cui fa riferimento **exitClasspath**.
- Per un'applicazione non in esecuzione in un application server, si applicano le seguenti regole:
  - Se le classi di uscita del canale sono compresse in file JAR separati, tali file JAR devono essere inclusi in **exitClasspath**.
  - Se le classi di uscita del canale non sono compresse in file JAR, i file di classe possono essere memorizzati nella directory mostrata in Tabella 58 a pagina 396 o in qualsiasi directory nel percorso di classe del sistema JVM o in **exitClasspath**.

La proprietà **exitClasspath** può essere specificata in quattro modi. In ordine di priorità, queste modalità sono le seguenti:

1. La proprietà di sistema com.ibm.mq.exitClasspath (definita sulla riga comandi utilizzando l'opzione `-D`)
2. La sezione exitPath del file `mqclient.ini`
3. Una voce hashtable con la chiave CMQC.EXIT_CLASSPATH_PROPERTY
4. La variabile MQEnvironment **exitClasspath**

Separare più percorsi utilizzando il carattere java.io.File.pathSeparator .

Tabella 58. La directory per i programmi di uscita del canale

Piattaforma	Directory
AIX AIX Linux Linux	`/var/mqm/exits` (programmi di uscita canale a 32 bit) `/var/mqm/exits64` (programmi di uscita del canale a 64 bit)
Windows	*dir_dati_installazione*\exits

Nota: *install_data_dir* è la directory scelta per i file di dati IBM MQ durante l'installazione. La directory predefinita è `C:\ProgramData\IBM\MQ`.

Passaggio dei dati alle uscite del canale in IBM MQ classes for Java
È possibile passare i dati alle uscite del canale e restituire i dati dalle uscite del canale all'applicazione.

Parametro agentBuffer

Per un'uscita di invio, il parametro *agentBuffer* contiene i dati che si sta per inviare. Per un'uscita di ricezione o di sicurezza, il parametro *agentBuffer* contiene i dati appena ricevuti. Non è necessario un parametro di lunghezza, in quanto l'espressione agentBuffer.limit () indica la lunghezza dell'array.

Per le uscite di invio e di sicurezza, il codice di uscita deve restituire i dati che si desidera inviare al server. Per un'uscita di ricezione, il codice di uscita deve restituire i dati modificati che si desidera che IBM MQ interpreti.

Il corpo di uscita più semplice possibile è:

```
{ return agentBuffer; }
```

Le uscite del canale vengono richiamate con un buffer che ha un array di supporto. Per prestazioni ottimali, l'uscita deve restituire un buffer con un array di backup.

Dati utente

Se un'applicazione si connette a un gestore code impostando channelSecurityExit, channelSendExit o channelReceiveExit, 32 byte di dati utente possono essere passati alla classe di uscita canale appropriata quando viene richiamata, utilizzando i dati channelSecurityExitUser, channelSendExitUserData o channelReceiveExitUserData. Questi dati sono disponibili per la classe di uscita del canale ma vengono aggiornati ogni volta che viene richiamata l'uscita. Tutte le modifiche apportate ai dati utente nell'uscita del canale andranno quindi perse. Se si desidera apportare modifiche persistenti ai dati in un'uscita del canale, utilizzare l'area MQCXP exitUser. I dati in questo campo vengono conservati tra i richiami dell'uscita.

Se l'applicazione imposta securityExit, sendExito receiveExit, non è possibile trasmettere dati utente a queste classi di uscita del canale.

Se un'applicazione utilizza una tabella di definizione del canale client (CCDT) per connettersi a un gestore code, tutti i dati utente specificati in una definizione del canale di connessione client vengono passati alle classi di uscita del canale quando vengono richiamati. Per ulteriori informazioni sull'utilizzo di una tabella di definizione del canale client, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for Java” a pagina 379.

Utilizzo delle uscite canale non scritte in Java con IBM MQ classes for Java
Come utilizzare i programmi di uscita canale scritti in C da un'applicazione Java .

In IBM MQ, è possibile specificare il nome di un programma di uscita del canale scritto in C come stringa passata ai campi channelSecurityExit, channelSendExit o channelReceiveExit nell'oggetto MQEnvironment o nelle proprietà Hashtable. Tuttavia, non è possibile utilizzare un'uscita del canale scritta in Java in un'applicazione scritta in un'altra lingua.

Specificare il nome del programma di uscita nel formato `library(function)` e assicurarsi che l'ubicazione del programma di uscita venga specificata come descritto in Percorso delle uscite.

Per informazioni su come scrivere un'uscita canale in C, consultare “Programmi di uscita canale per canali di messaggistica” a pagina 963.

Utilizzo di una sequenza di uscite di invio o ricezione del canale in IBM MQ classes for Java
Un'applicazione IBM MQ classes for Java può utilizzare una sequenza di uscite di invio o ricezione del canale eseguite in successione.

Per utilizzare una sequenza di uscite di invio, un'applicazione può creare un elenco o una stringa contenente le uscite di invio. Se viene utilizzato un elenco, ogni elemento dell'elenco può essere uno dei seguenti:

- Un'istanza di una classe definita dall'utente che implementa l'interfaccia WMQSendExit
- Un'istanza di una classe definita dall'utente che implementa l'interfaccia MQSendExit (per un'uscita di invio scritta in Java)

- Un'istanza della classe di uscita MQExternalSend(per un'uscita di invio non scritta in Java)
- Un'istanza della classe Chain MQSendExit
- Un'istanza della classe String

Un elenco non può contenere un altro elenco.

L'applicazione può utilizzare una sequenza di uscite di ricezione in modo simile.

Se viene utilizzata una stringa, deve essere costituita da una o più definizioni di uscita separate da virgole, ognuna delle quali può essere il nome di una classe Java o un programma C nel formato `library(function)`.

L'applicazione quindi assegna l'oggetto List o String al campo MQEnvironment.channelSendExit prima di creare un oggetto MQQueueManager .

Il contesto delle informazioni trasmesse alle uscite si trova esclusivamente all'interno del dominio delle uscite. Ad esempio, se un'uscita Java e un'uscita C sono concatenate, la presenza dell'uscita Java non ha alcun effetto sull'uscita C.

Utilizzo delle classi della catena di uscita

Nelle versioni precedenti a IBM WebSphere MQ 7.0, erano fornite due classi per consentire sequenze di uscite:

- Catena MQSendExit, che implementa l'interfaccia MQSendExit
- Catena MQReceiveExit, che implementa l'interfaccia MQReceiveExit

L'utilizzo di queste classi rimane valido ma si preferisce il nuovo metodo. L'uso delle interfacce IBM MQ Classi per Java significa che l'applicazione ha ancora una dipendenza su `com.ibm.mq.jar` Se viene utilizzata la nuova serie di interfacce nel pacchetto `com.ibm.mq.exits` , non vi è alcuna dipendenza su `com.ibm.mq.jar`.

Per utilizzare una sequenza di uscite di invio, un'applicazione ha creato un elenco di oggetti, dove ogni oggetto era uno dei seguenti:

- Un'istanza di una classe definita dall'utente che implementa l'interfaccia MQSendExit (per un'uscita di invio scritta in Java)
- Un'istanza della classe di uscita MQExternalSend(per un'uscita di invio non scritta in Java)
- Un'istanza della classe Chain MQSendExit

L'applicazione ha creato un oggetto della catena MQSendExitinoltrando questo elenco di oggetti come parametro sul costruttore. L'applicazione avrebbe quindi assegnato l'oggetto catena MQSendExital campo MQEnvironment.sendExit prima di creare un oggetto MQQueueManager .

Compressione canale in IBM MQ classes for Java

La compressione dei dati che fluiscono su un canale può migliorare le prestazioni del canale e ridurre il traffico di rete. IBM MQ classes for Java utilizzare la funzione di compressione integrata in IBM MQ.

Utilizzando la funzione fornita con IBM MQ, è possibile comprimere i dati che fluiscono sui canali di messaggi e MQI e, su entrambi i tipi di canale, è possibile comprimere i dati di intestazione e i dati di messaggio indipendentemente l'uno dall'altro. Per impostazione predefinita, non viene compresso alcun dato su un canale. Per una descrizione completa della compressione del canale, inclusa la modalità di implementazione in IBM MQ, consultare Data compression (COMPMSG) e Header compression (COMPHDR).

Un'applicazione IBM MQ classes for Java specifica le tecniche che possono essere utilizzate per comprimere i dati di intestazione o di messaggio su una connessione client creando un oggetto java.util.Collection . Ogni tecnica di compressione è un oggetto intero nella raccolta e l'ordine in cui l'applicazione aggiunge le tecniche di compressione alla raccolta è l'ordine in cui le tecniche di compressione vengono negoziate con il gestore code quando viene avviata la connessione client. L'applicazione può quindi assegnare la raccolta al campo hdrCompList, per i dati di intestazione, o al

campo msgCompList, per i dati del messaggio, nella classe MQEnvironment. Quando l'applicazione è pronta, può avviare la connessione client creando un oggetto MQQueueManager .

I seguenti frammenti di codice illustreranno l'approccio descritto. Il primo frammento di codice mostra come implementare la compressione dei dati di intestazione:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Il secondo frammento di codice mostra come implementare la compressione dei dati del messaggio:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));

:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Nel secondo esempio, le tecniche di compressione vengono negoziate nell'ordine RLE, quindi ZLIBHIGH, all'avvio della connessione client. La tecnica di compressione selezionata non può essere modificata durante la durata dell'oggetto MQQueueManager .

Le tecniche di compressione per i dati dell'intestazione e del messaggio supportati sia dal client che dal gestore code su una connessione client vengono inoltrati a un'uscita del canale come raccolte nei campi Elenco hdrCompe Elenco msgCompdi un oggetto MQChannelDefinition . Le tecniche effettive attualmente utilizzate per comprimere i dati dell'intestazione e del messaggio su una connessione client vengono inoltrate a un'uscita del canale nei campi di compressione CurHdre CurMsgdi un oggetto MQChannelExit .

Se la compressione viene utilizzata su una connessione client, i dati vengono compressi prima che le uscite di invio del canale vengano elaborate ed estratte dopo l'elaborazione delle uscite di ricezione del canale. I dati passati per inviare e ricevere le uscite sono quindi in uno stato compresso.

Per ulteriori informazioni sulla specifica delle tecniche di compressione e sulle tecniche di compressione disponibili, consultare Class com.ibm.mq.MQEnvironment e Interface com.ibm.mq.MQC.

Condivisione di una connessione TCP/IP in IBM MQ classes for Java

È possibile creare più istanze di un canale MQI per condividere una connessione TCP/IP singola.

In IBM MQ classes for Java, utilizzare la variabile MQEnvironment.sharingConversations per controllare il numero di conversazioni che possono condividere una singola connessione TCP/IP.

L'attributo SHARECNV è un approccio ottimale alla condivisione delle connessioni. Pertanto, quando un valore SHARECNV maggiore di 0 viene utilizzato con IBM MQ classes for Java , non è garantito che una nuova richiesta di connessione condivida sempre una connessione già stabilita.

Pool di connessioni in IBM MQ classes for Java

IBM MQ classes for Java consente il pool di connessioni di riserva per il riutilizzo.

IBM MQ classes for Java fornisce ulteriore supporto per le applicazioni che gestiscono più connessioni ai gestori code IBM MQ . Quando una connessione non è più necessaria, invece di eliminarla, può essere raggruppata e successivamente riutilizzata. Ciò può fornire un miglioramento sostanziale delle prestazioni per applicazioni e middleware che si connettono in modo seriale a gestori code arbitrari.

IBM MQ fornisce un pool di connessioni predefinito. Le applicazioni possono attivare o disattivare questo pool di connessioni registrando e annullando la registrazione dei token tramite la classe MQEnvironment. Se il lotto è attivo quando IBM MQ classes for Java costruisce un oggetto MQQueueManager , ricerca questo lotto predefinito e riutilizza qualsiasi connessione adatta. Quando si verifica una chiamata MQQueueManager.disconnect (), la connessione sottostante viene restituita al lotto.

In alternativa, le applicazioni possono creare un pool di connessioni MQSimpleConnectionManager per un utilizzo particolare. Quindi, l'applicazione può specificare tale pool durante la creazione di un oggetto MQQueueManager oppure passare tale pool a MQEnvironment per utilizzarlo come pool di connessioni predefinito.

Per evitare che le connessioni utilizzino una quantità eccessiva di risorse, è possibile limitare il numero totale di connessioni che un oggetto gestore MQSimpleConnectionpuò gestire ed è possibile limitare la dimensione del lotto connessioni. L'impostazione dei limiti è utile se vi sono richieste in conflitto per le connessioni all'interno di una JVM.

Per impostazione predefinita, il metodo getMaxConnections () restituisce il valore zero, che indica che non esiste alcun limite al numero di connessioni che l'oggetto MQSimpleConnectionManager può gestire. È possibile impostare un limite utilizzando il metodo setMaxConnections (). Se si imposta un limite e il limite viene raggiunto, una richiesta per un'altra connessione potrebbe causare la generazione di un'eccezione MQException, con un codice motivo di MQRC_MAX_CONNS_LIMIT_REACHED.

Controllo del pool di connessioni predefinito in IBM MQ classes for Java
Questo esempio mostra come utilizzare il pool di connessione predefinito.

Considerare la seguente applicazione di esempio, MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 prende un elenco di gestori code locali dalla riga comandi, si connette a ciascuno di essi ed esegue alcune operazioni. Tuttavia, quando la riga comandi elenca lo stesso gestore code più volte, è più efficiente connettersi una sola volta e riutilizzare tale connessione più volte.

IBM MQ classes for Java fornisce un pool di connessioni predefinito che è possibile utilizzare a tale scopo. Per abilitare il pool, utilizzare uno dei metodi MQEnvironment.addConnectionPoolToken(). Per disabilitare il pool, utilizzare MQEnvironment.removeConnectionPoolToken().

La seguente applicazione di esempio, MQApp2, è funzionalmente identica a MQApp1, ma si connette solo una volta a ciascun gestore code.

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

La prima riga in grassetto attiva il pool di connessione predefinito registrando un oggetto MQPoolToken con MQEnvironment.

Il costruttore MQQueueManager ora ricerca in questo lotto una connessione appropriata e crea una connessione al gestore code solo se non riesce a trovare una connessione esistente. La chiamata qmgr.disconnect() restituisce la connessione al lotto per un riutilizzo successivo. Queste chiamate API sono le stesse dell'applicazione di esempio MQApp1.

La seconda riga evidenziata disattiva il pool di connessione predefinito, che elimina tutte le connessioni del gestore code memorizzate nel pool. Ciò è importante perché altrimenti l'applicazione terminerebbe con un numero di connessioni gestore code attive nel pool. Questa situazione potrebbe causare errori che vengono visualizzati nei log del gestore code.

Se un'applicazione utilizza una tabella di definizione di canale client (CCDT) per connettersi a un gestore code, il costruttore MQQueueManager ricerca prima nella tabella una definizione di canale di connessione client adatta. Se ne viene trovato uno, il costruttore ricerca il pool di connessione predefinito per una connessione che può essere utilizzata per il canale. Se il costruttore non riesce a trovare una connessione adatta nel pool, ricerca nella tabella di definizione del canale client la successiva definizione di canale di connessione client adatta e procede nel modo descritto in precedenza. Se il costruttore completa la ricerca della tabella di definizione del canale client e non riesce a trovare alcuna connessione adatta nel lotto, il costruttore avvia una seconda ricerca della tabella. Durante questa ricerca, il costruttore tenta di creare una nuova connessione per ogni definizione di canale di connessione client adatta e utilizza la prima connessione che riesce a creare.

Il pool di connessione predefinito memorizza un massimo di dieci connessioni inutilizzate e mantiene attive le connessioni inutilizzate per un massimo di cinque minuti. L'applicazione può modificarlo (per i dettagli, consultare “Fornitura di un pool di connessione differente in IBM MQ classes for Java” a pagina 402).

Invece di utilizzare MQEnvironment per fornire un MQPoolToken, l'applicazione può crearne uno proprio:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Alcuni fornitori di applicazioni o middleware forniscono sottoclassi di MQPoolToken per trasmettere le informazioni a un pool di connessioni personalizzato. Possono essere creati e passati a addConnectionPoolToken() in questo modo in modo che ulteriori informazioni possano essere trasmesse al pool di connessioni.

Il pool di connessioni predefinito e più componenti in IBM MQ classes for Java
Questo esempio mostra come aggiungere o rimuovere MQPoolTokens da una serie statica di oggetti MQPoolToken registrati.

MQEnvironment contiene una serie statica di oggetti MQPoolToken registrati. Per aggiungere o rimuovere MQPoolTokens da questa serie, utilizzare i seguenti metodi:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Un'applicazione potrebbe essere composta da molti componenti che esistono in modo indipendente e che eseguono il lavoro utilizzando un gestore code. In un'applicazione di questo tipo, ogni componente deve aggiungere un MQPoolToken alla serie MQEnvironment per la sua durata.

Ad esempio, l'applicazione di esempio MQApp3 crea dieci thread e li avvia. Ogni thread registra il proprio MQPoolToken, attende per un periodo di tempo e si connette al gestore code. Dopo la disconnessione del thread, rimuove il proprio MQPoolToken.

Il pool di connessione predefinito rimane attivo mentre è presente almeno un token nella serie di MQPoolTokens, quindi rimarrà attivo per la durata di questa applicazione. L'applicazione non ha bisogno di mantenere un oggetto master nel controllo generale dei thread.

```
import com.ibm.mq.*;
```

```
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

Fornitura di un pool di connessione differente in IBM MQ classes for Java
Questo esempio mostra come utilizzare la classe **com.ibm.mq.MQSimpleConnectionManager** per fornire un pool di connessione differente.

Questa classe fornisce funzioni di base per il pool di connessioni e le applicazioni possono utilizzare questa classe per personalizzare il comportamento del pool.

Una volta creata, è possibile specificare un gestore MQSimpleConnectionsul costruttore MQQueueManager . Il gestore MQSimpleConnectiongestisce quindi la connessione alla base del MQQueueManagercreato. Se il gestore MQSimpleConnectioncontiene una connessione in pool adatta, tale connessione viene riutilizzata e restituita al gestore MQSimpleConnectiondopo una chiamata MQQueueManager.disconnect ().

Il seguente frammento di codice dimostra questo comportamento:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

La connessione forgiata durante il primo costruttore MQQueueManager viene memorizzata in myConnMan dopo la chiamata qmgr.disconnect(). La connessione viene quindi riutilizzata durante la seconda chiamata al costruttore MQQueueManager .

La seconda riga abilita il gestore MQSimpleConnection. L'ultima riga disabilita MQSimpleConnectionManager, eliminando tutte le connessioni presenti nel lotto. Un gestore MQSimpleConnectionè, per impostazione predefinita, in MODE_AUTO, descritto più avanti in questa sezione.

Un gestore MQSimpleConnectionassegna le connessioni in base all'utilizzo più recente e distrugge le connessioni in base all'utilizzo meno recente. Per impostazione predefinita, una connessione viene eliminata se non è stata utilizzata per cinque minuti o se nel pool sono presenti più di dieci connessioni non utilizzate. È possibile modificare questi valori richiamando MQSimpleConnectionManager.setTimeout().

È inoltre possibile impostare un gestore MQSimpleConnectionda utilizzare come pool di connessioni predefinito, da utilizzare quando non viene fornito un gestore connessioni sul costruttore di MQQueueManager .

La seguente applicazione lo dimostra:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

Le righe in grassetto creano e configurano un oggetto MQSimpleConnectionManager. La configurazione effettua quanto segue:

- Termina le connessioni non utilizzate per un'ora
- Limita il numero di connessioni gestite da myConnMan a 75
- Limita il numero di connessioni non utilizzate nel pool a 50
- Imposta MODE_AUTO, che è il valore predefinito. Ciò significa che il pool è attivo solo se è il gestore connessioni predefinito e che è presente almeno un token nella serie di MQPoolTokens detenuti da MQEnvironment.

Il nuovo gestore MQSimpleConnectionviene quindi impostato come gestore connessioni predefinito.

Nell'ultima riga, l'applicazione richiama MQApp3.main(). Questo esegue un numero di thread, in cui ogni thread utilizza IBM MQ in modo indipendente. Questi thread utilizzano myConnMan quando creano connessioni.

Coordinamento JTA/JDBC utilizzando IBM MQ classes for Java

IBM MQ classes for Java supporta il metodo MQQueueManager.begin (), che consente a IBM MQ di agire come coordinatore per un database che fornisce un programma di controllo conforme a JDBC tipo 2 o JDBC tipo 4.

Questo supporto non è disponibile su tutte le piattaforme. Per controllare quali piattaforme supportano il coordinamento JDBC , consultare Requisiti di sistema per IBM MQ.

Per utilizzare il supporto XA - JTA, è necessario utilizzare la libreria di switch JTA speciale. Il metodo per utilizzare questa libreria varia a seconda che si stia utilizzando Windows o una delle altre piattaforme.

Configurazione del coordinamento JTA/JDBC su Windows

La libreria XA viene fornita come DLL con un nome del formato `jdbcxxx.dll`.

Il `jdbcora12.dll` fornito fornisce la compatibilità con Oracle 12C, per un'installazione del server IBM MQ per Windows .

Su sistemi Windows , la libreria XA viene fornita come una DLL completa. Il nome di questa DLL è `jdbcxxx.dll` dove `xxx` indica il database per cui è stata compilata la libreria switch. Questa libreria si trova nella directory `java\lib\jdbc` o `java\lib64\jdbc` dell'installazione di IBM MQ classes for Java . È necessario dichiarare la libreria XA, descritta anche come file di caricamento switch, al gestore code. Utilizzare IBM MQ Explorer. Specificare i dettagli del file di caricamento switch nel pannello delle proprietà del gestore code, nel gestore risorse XA. È necessario fornire solo il nome della libreria. Ad esempio:

Per un database Db2 impostare il campo SwitchFile su `dbcdb2`

Per un database Oracle impostare il campo SwitchFile su `jdbcora`

Note:

1. Oracle 12C è supportato da IBM MQ classes for Java, solo su IBM MQ per Windows.
2. La versione supportata di Oracle 12C è 12.1.0.1.0 Enterprise Edition e futuri fix pack.
3. Oracle a 64 bit su Windows a 64 bit richiedono il client Oracle a 32 bit.
4. Utilizzando IBM MQ classes for Java, IBM MQ può agire come coordinatore della transazione. Tuttavia, non è possibile partecipare a una transazione in stile JTA.

Configurazione della coordinazione JTA/JDBC su piattaforme diverse da Windows
Vengono forniti i file oggetto. Collegare quello appropriato utilizzando il makefile fornito e dichiararlo al gestore code utilizzando il file di configurazione.

Per ciascun sistema di gestione database, IBM MQ fornisce due file oggetto. È necessario collegare un file oggetto per creare una libreria di switch a 32 bit e collegare l'altro file oggetto per creare una libreria di switch a 64 bit. Per Db2, il nome di ciascun file oggetto è `jdbcdb2.o`e per Oracle il nome di ogni file oggetto è `jdbcora.o`.

È necessario collegare ogni file oggetto utilizzando il makefile appropriato fornito con IBM MQ. Una libreria switch richiede altre librerie, che potrebbero essere memorizzate in ubicazioni differenti su sistemi differenti. Tuttavia, una libreria switch non può utilizzare la variabile di ambiente del percorso libreria per individuare queste librerie perché la libreria switch viene caricata dal gestore code, che viene eseguito in un ambiente `setuid` . Il makefile fornito garantisce quindi che una libreria switch contenga i nomi percorso completi di tali librerie.

Per creare una libreria switch, immettere un comando **make** con il seguente formato. Per creare una libreria di switch a 32 bit, immettere il comando nella directory `/java/lib/jdbc` dell'installazione di IBM MQ . Per creare una libreria di switch a 64 bit, immettere il comando nella directory `/java/lib64/jdbc` .

```
make DBMS
```

dove *DBMS* è il sistema di gestione database per cui si sta creando la libreria switch. I valori validi sono db2 per Db2 e `oracle` per Oracle.

Nota:

- Per eseguire applicazioni a 32 bit, è necessario creare una libreria di switch a 32 bit e a 64 bit per ciascun sistema di gestione database che si sta utilizzando. Per eseguire applicazioni a 64 bit, è necessario creare solo una libreria di switch a 64 bit. Per Db2, il nome di ciascuna libreria di switch è `jdbcdb2` e, per Oracle, il nome di ogni libreria di switch è `jdbcora`. I makefile garantiscono che le librerie di switch a 32 bit e a 64 bit siano memorizzate in directory IBM MQ differenti. Una libreria di switch a 32 bit è memorizzata nella directory `/java/lib/jdbc` e una libreria di switch a 64 bit è memorizzata nella directory `/java/lib64/jdbc` .
- Poiché è possibile installare Oracle ovunque su un sistema, i makefile utilizzano la variabile di ambiente **ORACLE_HOME** per individuare dove è installato Oracle .
- Se IBM MQ è installato in un percorso diverso da quello predefinito, modificare il valore di **MQ_INSTALLATION_PATH** nel makefile.

Dopo aver creato le librerie di switch per Db2, Oracleo entrambe, è necessario dichiararle al proprio gestore code. Se il file di configurazione del gestore code (qm.ini) contiene già XAResourceManager stanze per database Db2 o Oracle , è necessario sostituire la voce SwitchFile in ogni stanza con una delle seguenti:

Per un database Db2

```
SwitchFile=jdbcdb2
```

Per un database Oracle

```
SwitchFile=jdbcora
```

Non specificare il nome percorso completo della libreria di commutazione a 32 bit o a 64 bit. Specificare solo il nome della libreria.

Se il file di configurazione del gestore code non contiene già le stanze XAResourceManager per i database Db2 o Oracle o se si desidera aggiungere ulteriori stanze XAResourceManager , consultare Amministrazione IBM MQ per informazioni su come creare una stanza XAResourceManager . Tuttavia, ogni voce SwitchFile in una nuova stanza XAResourceManager deve essere esattamente come descritto in precedenza per un database Db2 o Oracle . È necessario includere anche la voce ThreadOfControl=PROCESS.

Dopo aver aggiornato il file di configurazione del gestore code e essersi assicurati che siano state impostate tutte le variabili di ambiente del database appropriate, è possibile riavviare il gestore code.

Utilizzo del coordinamento JTA/JDBC
Codificare le chiamate API come nell'esempio fornito.

La sequenza di base delle chiamate API per un'applicazione utente è:

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

xads nella chiamata getJDBCConnection è un'implementazione specifica del database dell'interfaccia XADataSource , che definisce i dettagli del database a cui connettersi. Consultare la documentazione per il proprio database per determinare come creare un oggetto XADataSource appropriato da passare a getJDBCConnection.

È inoltre necessario aggiornare il percorso di classe con i file jar specifici del database appropriati per eseguire il lavoro JDBC .

Se è necessario connettersi a più database, è necessario richiamare getJDBCConnection più volte per eseguire la transazione su diverse connessioni.

Esistono due forme di getJDBCConnection, che riflettono le due forme di XADataSource.getXAConnection:

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

Questi metodi dichiarano l'eccezione nelle relative clausole throws per evitare problemi con il programma di verifica JVM per i clienti che non utilizzano le funzioni JTA. L'eccezione effettiva generata è

javax.transaction.xa.XAException che richiede l'aggiunta del file jta.jar al percorso di classe per i programmi che non lo richiedevano in precedenza.

Per utilizzare il supporto JTA/JDBC , è necessario includere la seguente istruzione nell'applicazione:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Problemi noti e limitazioni con il coordinamento di JTA/JDBC
Alcuni dei problemi e delle limitazioni del supporto JTA/JDBC dipendono dal sistema di gestione del database in uso, ad esempio, i driver JDBC testati si comportano in modo diverso quando il database viene arrestato mentre un'applicazione è in esecuzione. Se la connessione al database che un'applicazione sta utilizzando è interrotta, l'applicazione può eseguire delle operazioni per ristabilire una nuova connessione al gestore code e al database in modo che possa utilizzare tali nuove connessioni per eseguire il lavoro transazionale richiesto.

Poiché il supporto JTA/JDBC effettua chiamate ai driver JDBC , l'implementazione di tali driver JDBC può avere un effetto significativo sul funzionamento del sistema. In particolare, i driver JDBC verificati si comportano in modo diverso quando il database viene chiuso mentre un'applicazione è in esecuzione.

Importante: Evitare sempre la chiusura improvvisa di un database mentre vi sono applicazioni che detengono connessioni aperte.

Nota: Un'applicazione IBM MQ classes for Java deve connettersi utilizzando la modalità bind per fare in modo che IBM MQ agisca come coordinatore del database.

Più stanze XAResourceManager
L'uso di più di una stanza XAResourceManager in un file di configurazione del gestore code, `qm.ini`, non è supportato. Qualsiasi stanza XAResourceManager diversa dalla prima viene ignorata.

Db2

A volte, Db2 restituisce un errore SQL0805N . Questo problema può essere risolto con il seguente comando CLP:

```
DB2 bind @db2cli.lst blocking all grant public
```

Per ulteriori informazioni, fare riferimento alla documentazione di Db2.

La stanza XAResourceManager deve essere configurata per utilizzare ThreadOfControl = PROCESS. Per Db2 8.1 e versioni successive, non corrisponde al thread predefinito dell'impostazione di controllo per Db2, quindi toc=p deve essere specificato in XA Open String. Una stanza XAResourceManager di esempio per Db2 con coordinamento JTA/JDBC è la seguente:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

Ciò non impedisce alle applicazioni Java che utilizzano la coordinazione JTA/JDBC di essere multithread.

Oracle
Richiamando il metodo JDBC Connection.close() dopo che MQQueueManager.disconnect () ha generato una SQLException. Richiamare Connection.close() prima di MQQueueManager.disconnect () oppure omettere la chiamata a Connection.close().

Gestione dei problemi con le connessioni al database

Quando un'applicazione IBM MQ classes for Java usa il supporto JTA/JDBC fornito da IBM MQ, in genere effettua le seguenti operazioni:

1. Crea un nuovo oggetto MQQueueManager per rappresentare una connessione al gestore code che agirà come gestore transazioni.
2. Crea un oggetto XADataSource che contiene dettagli su come connettersi al database che verrà inserito nella transazione.
3. Richiama il metodo MQQueueManager.getJDBCConnection(XADataSource) inoltrando XADataSource creato precedentemente. Ciò fa sì che IBM MQ classes for Java stabilisca una connessione al database.
4. Richiama il metodo MQQueueManager.begin () per avviare la transazione XA.
5. Esegue il lavoro di messaggistica e database.
6. Quando tutto il lavoro richiesto è stato completato, richiama il metodo MQQueueManager.commit (). Questa operazione completa la transazione XA.
7. Se a questo punto è richiesta una nuova transazione XA, l'applicazione può ripetere i passi 4, 5 e 6.
8. Una volta terminata l'applicazione, è necessario chiudere la connessione al database creata al passo 3 e richiamare il metodo MQQueueManager.disconnect () per disconnettersi dal gestore code.

IBM MQ classes for Java conserva un elenco interno di tutte le connessioni al database che sono state create quando un'applicazione richiama MQQueueManager.getJDBCConnection(XADataSource). Se un gestore code deve comunicare con il database durante l'elaborazione della transazione XA, si verifica la seguente elaborazione:

1. Le chiamate del gestore code in IBM MQ classes for Java, passando i dettagli della chiamata XA che deve essere passata al database.
2. Il IBM MQ classes for Java , quindi, ricerca la connessione appropriata nell'elenco e utilizza tale connessione per passare la chiamata XA al database.

Se la connessione al database viene persa in qualsiasi momento durante questa elaborazione, l'applicazione deve:

1. Eseguire il backout di qualsiasi lavoro esistente eseguito nella transazione, richiamando il metodo MQQueueManager.backout ().
2. Chiudere la connessione al database. Ciò dovrebbe far sì che IBM MQ classes for Java rimuova i dettagli della connessione al database interrotta dal proprio elenco interno.
3. Disconnettersi dal gestore code, richiamando il metodo MQQueueManager.disconnect ().
4. Stabilire una nuova connessione al gestore code, creando un nuovo oggetto MQQueueManager .
5. Creare una nuova connessione al database, richiamando il metodo MQQueueManager.getJDBCConnection(XADataSource).
6. Eseguire nuovamente il lavoro di transazione.

Ciò consente all'applicazione di ristabilire una nuova connessione al gestore code e al database, quindi di utilizzare tali connessioni per eseguire il lavoro di transazione richiesto.

Supporto TLS (Transport Layer Security) in IBM MQ classes for Java

Le applicazioni client IBM MQ classes for Java supportano la codifica TLS. È necessario un fornitore JSSE per utilizzare la codifica TLS.

Le applicazioni client IBM MQ classes for Java che utilizzano TRANSPORT (CLIENT) supportano la codifica TLS. TLS fornisce la codifica di comunicazione, l'autenticazione e l'integrità del messaggio. Generalmente viene utilizzato per proteggere le comunicazioni tra due peer su Internet o all'interno di una intranet.

IBM MQ classes for Java utilizza JSSE (Java Secure Socket Extension) per gestire la cifratura TLS e quindi richiede un provider JSSE. Le JVM JSE v1.4 hanno un provider JSSE integrato. I dettagli su come gestire e memorizzare i certificati possono variare da fornitore a fornitore. Per informazioni, fare riferimento alla documentazione del provider JSSE.

Questa sezione presuppone che il provider JSSE sia installato e configurato correttamente e che i certificati appropriati siano stati installati e resi disponibili per il proprio provider JSSE.

Se l'applicazione client IBM MQ classes for Java utilizza una tabella di definizione del canale client (CCDT) per connettersi a un gestore code, consultare “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for Java” a pagina 379.

Abilitazione di TLS in IBM MQ classes for Java
Per abilitare TLS, si specifica una CipherSuite. Esistono due metodi per specificare una CipherSuite.

TLS è supportato solo per connessioni client. Per abilitare TLS, è necessario specificare la CipherSuite da utilizzare durante la comunicazione con il gestore code e questa CipherSuite deve corrispondere alla CipherSpec impostata sul canale di destinazione. Inoltre, la CipherSuite denominata deve essere supportata dal provider JSSE. Tuttavia, le CipherSuites sono distinte da CipherSpecs e hanno nomi differenti. “TLS CipherSpecs e CipherSuites in IBM MQ classes for Java” a pagina 412 contiene una tabella che associa i CipherSpecs supportati da IBM MQ ai CipherSuites equivalenti noti a JSSE.

Per abilitare TLS, specificare CipherSuite utilizzando la variabile del membro statico della suite sslCipherdi MQEnvironment. Il seguente esempio si collega a un canale SVRCONN denominato SECURE.SVRCONN.CHANNEL, che è stato impostato per richiedere TLS con un CipherSpec di TLS_RSA_WITH_AES_128_CBC_SHA256:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Anche se il canale dispone di una CipherSpec di TLS_RSA_WITH_AES_128_CBC_SHA256, l'applicazione Java deve specificare una CipherSuite di SSL_RSA_WITH_AES_128_CBC_SHA256. Consultare “TLS CipherSpecs e CipherSuites in IBM MQ classes for Java” a pagina 412 per un elenco di associazioni tra CipherSpecs e CipherSuites.

Un'applicazione può anche specificare una CipherSuite impostando la proprietà di ambiente CMQC.SSL_CIPHER_SUITE_PROPERTY.

In alternativa, utilizzare CCDT (Client Channel Definition Table). Per ulteriori informazioni, vedi “Utilizzo di una tabella di definizione di canale client con IBM MQ classes for Java” a pagina 379

Se si richiede una connessione client per utilizzare una CipherSuite supportata dal provider IBM Java JSSE FIPS (IBMJSSEFIPS), un'applicazione può impostare il campo obbligatorio sslFipsnella classe MQEnvironment su `true`. In alternativa, l'applicazione può impostare la proprietà di ambiente CMQC.SSL_FIPS_REQUIRED_PROPERTY. Il valore predefinito è `false`, che significa che una connessione client può utilizzare qualsiasi CipherSuite supportato da IBM MQ.

Se un'applicazione utilizza più di una connessione client, il valore del campo sslFipsObbligatorio utilizzato quando l'applicazione crea la prima connessione client determina il valore utilizzato quando l'applicazione crea una connessione client successiva. Pertanto, quando l'applicazione crea una successiva connessione client, il valore del campo sslFipsObbligatorio viene ignorato. È necessario riavviare l'applicazione se si desidera utilizzare un valore diverso per il campo sslFipsObbligatorio.

Per connettersi correttamente utilizzando TLS, il truststore JSSE deve essere configurato con i certificati root dell'autorità di certificazione da cui è possibile autenticare il certificato presentato dal gestore code. Allo stesso modo, se SSLClientAuth sul canale SVRCONN è stato impostato su MQSSL_CLIENT_AUTH_REQUIRED, il keystore JSSE deve contenere un certificato di identificazione ritenuto attendibile dal gestore code.

Riferimenti correlati
FIPS (Federal Information Processing Standards) per AIX, Linux, and Windows

Utilizzo del DN del gestore code in IBM MQ classes for Java
Il gestore code si identifica utilizzando un certificato TLS, che contiene un DN (distinguished name). Un'applicazione client IBM MQ classes for Java può utilizzare questo DN per assicurarsi che stia comunicando con il gestore code corretto.

Un modello DN viene specificato utilizzando la variabile nome sslPeerdi MQEnvironment. Ad esempio, l'impostazione:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

consente la riuscita della connessione solo se il gestore code presenta un certificato con un nome comune che inizia con QMGR., e almeno due nomi di unità organizzative, il primo dei quali deve essere IBM e il secondo WebSphere.

Se è impostato il nome sslPeer, le connessioni hanno esito positivo solo se sono impostate su un modello valido e il gestore code presenta un certificato corrispondente.

Un'applicazione può anche specificare il DN (Distinguished Name) del gestore code impostando la proprietà di ambiente CMQC.SSL_PEER_NAME_PROPERTY. Per ulteriori informazioni sui DN (distinguished name), consultare DN (distinguished name).

Utilizzo degli elenchi di revoca dei certificati in IBM MQ classes for Java
Specificare gli elenchi di revoca certificati da utilizzare tramite la classe java.security.cert.CertStore . IBM MQ classes for Java controlla quindi i certificati rispetto al CRL specificato.

Un CRL (Certificate Revocation List) è una serie di certificati che sono stati revocati, dall'autorità di certificazione emittente o dall'organizzazione locale. I CRL sono generalmente ospitati su server LDAP. Con Java 2 v1.4, un server CRL può essere specificato in fase di connessione e il certificato presentato dal gestore code viene controllato rispetto al CRL prima che la connessione sia consentita. Per ulteriori informazioni sugli elenchi di revoca dei certificati e IBM MQ, consultare Gestione degli elenchi di revoca dei certificati e degli elenchi di revoca delle autorità e Accesso ai CRL e agli ARL con IBM MQ classes for Java e IBM MQ classes for JMS.

Nota: Per utilizzare correttamente un CertStore con un CRL ospitato su un server LDAP, assicurati che il tuo SDK (Software Development Kit) Java sia compatibile con il CRL. Alcuni SDK richiedono che il CRL sia conforme a RFC 2587, che definisce uno schema per LDAP v2. La maggior parte dei server LDAP v3 utilizza invece RFC 2256.

I CRL da utilizzare vengono specificati tramite la classe java.security.cert.CertStore . Fare riferimento alla documentazione su questa classe per i dettagli completi su come ottenere le istanze di CertStore. Per creare un CertStore basato su un server LDAP, creare prima un'istanza di parametri LDAPCertStore, inizializzata con le impostazioni server e porta da utilizzare. Ad esempio:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Dopo aver creato un'istanza dei parametri CertStore, utilizzare il costruttore statico su CertStore per creare un CertStore di tipo LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Sono supportati anche altri tipi CertStore (ad esempio, Collection). Di solito ci sono diversi server CRL configurati con informazioni CRL identiche per fornire ridondanza. Quando si dispone di un oggetto CertStore per ciascuno di questi server CRL, collocarli tutti in una raccolta adatta. Il seguente esempio mostra gli oggetti CertStore inseriti in un ArrayList:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

Questa raccolta può essere impostata nella variabile statica MQEnvironment, sslCertStores, prima della connessione per abilitare il controllo CRL:

```
MQEnvironment.sslCertStores = crls;
```

Il certificato presentato dal gestore code durante l'impostazione di una connessione viene convalidato nel modo seguente:

1. Il primo oggetto CertStore nella raccolta identificata da sslCertStores viene utilizzato per identificare un server CRL.
2. È stato effettuato un tentativo di contattare il server CRL.
3. Se il tentativo ha esito positivo, il server viene ricercato per una corrispondenza per il certificato.
   a. Se viene rilevato che il certificato è stato revocato, il processo di ricerca è stato completato e la richiesta di connessione ha esito negativo con codice motivo MQRC_SSL_CERTIFICATE_REVOKED.
   b. Se il certificato non viene trovato, il processo di ricerca è stato completato e la connessione può continuare.
4. Se il tentativo di contattare il server non ha esito positivo, il successivo oggetto CertStore viene utilizzato per identificare un server CRL e il processo si ripete dal passo 2.

   Se si tratta dell'ultima CertStore nella raccolta o se la raccolta non contiene oggetti CertStore , il processo di ricerca non è riuscito e la richiesta di connessione ha esito negativo con codice motivo MQRC_SSL_CERT_STORE_ERROR.

L'oggetto Raccolta determina l'ordine in cui vengono utilizzati i CertStores .

La raccolta di CertStores può essere impostata anche utilizzando CMQC.SSL_CERT_STORE_PROPERTY. Per comodità, questa proprietà consente anche di specificare un singolo CertStore senza essere membro di una raccolta.

Se sslCertStores è impostato su null, non viene eseguito alcun controllo CRL. Questa proprietà viene ignorata se la suite sslCiphernon è impostata.

Rinegoziazione della chiave segreta in IBM MQ classes for Java
Un'applicazione client di IBM MQ classes for Java può controllare quando la chiave segreta utilizzata per la codifica su una connessione client viene rinegoziata, in termini di numero totale di byte inviati e ricevuti.

L'applicazione può eseguire questa operazione in uno dei modi seguenti: se l'applicazione utilizza più di uno di questi modi, si applicano le solite regole di precedenza.

- Impostando il campo sslResetCount nella classe MQEnvironment.
- Impostando la proprietà di ambiente MQC.SSL_RESET_COUNT_PROPERTY in un oggetto Hashtable. L'applicazione, quindi, assegna l'hashtable al campo `properties` nella classe MQEnvironment o passa l'hashtable a un oggetto MQQueueManager sul relativo costruttore.

Il valore del campo sslReseto la proprietà dell'ambiente MQC.SSL_RESET_COUNT_PROPERTY rappresenta il numero totale di byte inviati e ricevuti dal codice client IBM MQ classes for Java prima che la chiave segreta venga rinegoziata. Il numero di byte inviati è il numero prima della codifica e il numero di byte ricevuti è il numero dopo la decodifica. Il numero di byte include anche le informazioni di controllo inviate e ricevute dal client IBM MQ classes for Java .

Se il conteggio di reimpostazione è zero, che è il valore predefinito, la chiave segreta non viene mai rinegoziata. Il conteggio di reimpostazioni viene ignorato se non viene specificato alcun CipherSuite .

Fornitura di un SSLSocketFactory personalizzato in IBM MQ classes for Java
Se si utilizza un factory di socket JSSE personalizzato, impostare MQEnvironment.sslSocketFactory sull'oggetto factory personalizzato. I dettagli variano tra le diverse implementazioni JSSE.

Diverse implementazioni JSSE possono fornire funzioni differenti. Ad esempio, un'implementazione JSSE specializzata potrebbe consentire la configurazione di un particolare modello di hardware di codifica. Inoltre, alcuni provider JSSE consentono la personalizzazione dei keystore e dei truststore in base al programma o consentono di modificare la scelta del certificato di identità dal keystore. In JSSE, tutte queste personalizzazioni vengono astratte in una classe factory, javax.net.ssl.SSLSocketFactory.

Consultare la documentazione JSSE per i dettagli su come creare un'implementazione SSLSocketFactory personalizzata. I dettagli variano da fornitore a provider, ma una tipica sequenza di passi potrebbe essere:

1. Creazione di un oggetto SSLContext utilizzando un metodo statico su SSLContext
2. Inizializzare questo SSLContext con le implementazioni KeyManager e TrustManager appropriate (create dalle proprie classi factory)

3. Crea un SSLSocketFactory da SSLContext

Quando si dispone di un oggetto SSLSocketFactory , impostare MQEnvironment.sslSocketFactory sull'oggetto factory personalizzato. Ad esempio:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java utilizzare questo SSLSocketFactory per connettersi al gestore code IBM MQ . Questa proprietà può essere impostata anche utilizzando CMQC.SSL_SOCKET_FACTORY_PROPERTY. Se sslSocketFactory è impostato su null, viene utilizzato il valore predefinito SSLSocketFactory della JVM. Questa proprietà viene ignorata se la suite sslCiphernon è impostata.

Quando si utilizzano SSLSocketFactoriespersonalizzati, considerare l'effetto della condivisione della connessione TCP/IP. Se la condivisione della connessione è possibile, non viene richiesto un nuovo socket del SSLSocketFactory fornito, anche se il socket prodotto sarebbe diverso in qualche modo nel contesto di una successiva richiesta di connessione. Ad esempio, se un certificato client differente deve essere presentato su una connessione successiva, la condivisione della connessione non deve essere consentita.

Esecuzione di modifiche al keystore o al truststore JSSE in IBM MQ classes for Java
Se si modifica il keystore o il truststore JSSE, è necessario eseguire alcune azioni per rendere effettive le modifiche.

Se si modifica il contenuto del keystore o del truststore JSSE o si modifica l'ubicazione del file keystore o truststore, le applicazioni IBM MQ classes for Java in esecuzione al momento non acquisiscono automaticamente le modifiche. Per rendere effettive le modifiche, è necessario eseguire le seguenti operazioni:

- Le applicazioni devono chiudere tutte le relative connessioni ed eliminare tutte le connessioni inutilizzate nei pool di connessioni.
- Se il provider JSSE memorizza nella cache le informazioni dal keystore e dal truststore, tali informazioni devono essere aggiornate.

Una volta eseguite queste azioni, le applicazioni possono ricreare le connessioni.

A seconda della modalità di progettazione delle applicazioni e della funzione fornita dal provider JSSE, potrebbe essere possibile eseguire queste azioni senza arrestare e riavviare le applicazioni. Tuttavia, l'arresto e il riavvio delle applicazioni potrebbe essere la soluzione più semplice.

Gestione degli errori quando si utilizza TLS con IBM MQ classes for Java
IBM MQ classes for Java può emettere un numero di codici di errore durante la connessione a un gestore code utilizzando TLS.

Questi sono spiegati nel seguente elenco:

MQRC_SSL_NOT_ALLOWED
La proprietà sslCipherSuite è stata impostata, ma è stata utilizzata la connessione dei bind. Solo la connessione client supporta TLS.

ERRORE MQRC_JSSE
Il fornitore JSSE ha riportato un errore che non può essere gestito da IBM MQ. Ciò potrebbe essere causato da un problema di configurazione con JSSE o perché non è stato possibile convalidare il certificato presentato dal gestore code. L'eccezione prodotta da JSSE può essere richiamata utilizzando il metodo getCause() su MQException.

ERRORE MQRC_SSL_INITIALIZATION_ERROR
È stata emessa una chiamata MQCONN o MQCONNX con le opzioni di configurazione TLS specificate, ma si è verificato un errore durante l'inizializzazione dell'ambiente TLS.

MQRC_SSL_PEER_NAME_MISMATCH
Il pattern DN specificato nella proprietà Nome sslPeernon corrisponde al DN presentato dal gestore code.

ERRORE MQRC_SSL_PEER_NAME_ERROR
Il modello DN specificato nella proprietà Nome sslPeernon è valido.

MQRC_UNSUPPORT_CIPHER_SUITE
La CipherSuite indicata nella suite sslCiphernon è stata riconosciuta dal fornitore JSSE. Un elenco completo di CipherSuites supportati dal provider JSSE può essere ottenuto da un programma utilizzando il metodo SSLSocketFactory.getSupportedCipherSuites(). Un elenco di CipherSuites che è possibile utilizzare per comunicare con IBM MQ è disponibile in "TLS CipherSpecs e CipherSuites in IBM MQ classes for Java" a pagina 412.

MQRC_SSL_CERTIFICATE_REVOKED
Il certificato presentato dal gestore code è stato trovato in un CRL specificato con la proprietà sslCertStores. Aggiornare il gestore code per utilizzare i certificati attendibili.

ERRORE - CERT_STORE_MQRC_SSL_CERT_
Non è stato possibile ricercare in nessuno dei CertStores forniti il certificato presentato dal gestore code. Il metodo MQException.getCause() restituisce l'errore che si è verificato durante la ricerca del primo CertStore tentato. Se l'eccezione causale è NoSuchElementException, ClassCastException o NullPointerException, verificare che la raccolta specificata nella proprietà sslCertStores contenga almeno un oggetto CertStore valido.

TLS CipherSpecs e CipherSuites in IBM MQ classes for Java
La capacità delle applicazioni IBM MQ classes for Java di stabilire connessioni a un gestore code dipende dalla CipherSpec specificata all'estremità server del canale MQI e dalla CipherSuite specificata all'estremità client.

La seguente tabella elenca i CipherSpecs supportati da IBM MQ e i relativi CipherSuitesequivalenti.

Deprecated È necessario esaminare l'argomento Deprecated CipherSpecs per verificare se uno dei CipherSpecs, elencati nella seguente tabella, è stato dichiarato obsoleto da IBM MQ e, in tal caso, è stato reso obsoleto l'aggiornamento di CipherSpec .

Importante: Le CipherSuites elencate sono quelle supportate da IBM Java Runtime Environment (JRE) fornito con IBM MQ. Le CipherSuites elencate includono quelle supportate da JRE Oracle Java . Per ulteriori informazioni sulla configurazione della propria applicazione per utilizzare un JRE Oracle Java , consultare "Configurazione della tua applicazione per utilizzare le associazioni IBM Java o Oracle Java CipherSuite" a pagina 431.

La tabella indica anche il protocollo utilizzato per le comunicazioni e se CipherSuite è conforme allo standard FIPS 140-2.

Nota: Su AIX, Linux, and Windows, IBM MQ fornisce la conformità FIPS 140-2 tramite il modulo crittografico IBM Crypto for C (ICC) . Il certificato per questo modulo è stato spostato nello stato cronologico. I clienti devono visualizzare il certificatoIBM Crypto for C (ICC) ed essere a conoscenza di eventuali consigli forniti da NIST. Un modulo FIPS 140-3 di sostituzione è attualmente in corso e il relativo stato può essere visualizzato ricercandolo in NIST CMVP modules in process list.

Le suite di cifratura indicate come conformi a FIPS 140-2 possono essere utilizzate se l'applicazione non è stata configurata per applicare la conformità FIPS 140-2, ma se la conformità FIPS 140-2 è stata configurata per l'applicazione (vedere le seguenti note sulla configurazione), è possibile configurare solo le CipherSuites contrassegnate come compatibili con FIPS 140-2; il tentativo di utilizzare altre CipherSuites genera un errore.

Nota: Ogni JRE può avere più provider di sicurezza crittografica, ognuno dei quali può contribuire con un'implementazione della stessa CipherSuite. Tuttavia, non tutti i provider di sicurezza sono certificati FIPS 140-2. Se la conformità FIPS 140-2 non viene applicata per un'applicazione, è possibile che venga utilizzata un'implementazione non certificata di CipherSuite . Le implementazioni non certificate potrebbero non essere conformi a FIPS 140-2, anche se CipherSuite in teoria soddisfa il livello di sicurezza minimo richiesto dallo standard. Consultare le seguenti note per ulteriori informazioni sulla configurazione dell'applicazione FIPS 140-2 nelle applicazioni IBM MQ Java .

Per ulteriori informazioni sulla conformità FIPS 140-2 e Suite - B per CipherSpecs e CipherSuites, consultare Specifica di CipherSpecs. Potrebbe anche essere necessario essere a conoscenza delle informazioni relative a Federal Information Processing Standardsdegli Stati Uniti.

Per utilizzare la serie completa di CipherSuites e per operare con la conformità FIPS 140-2 e / o Suite - B certificata, è richiesto un JRE adatto. IBM Java 7 Service Refresh 4 Fix Pack 2 o un livello superiore di IBM JRE fornisce il supporto appropriato per il TLS 1.2 CipherSuites elencato in Tabella 59 a pagina 413.

Per poter utilizzare le crittografie TLS 1.3 , il JRE che esegue l'applicazione deve supportare TLS 1.3.

Nota: Per utilizzare alcuni CipherSuites, i file della politica 'unrestricted' devono essere configurati in JRE. Per ulteriori dettagli su come vengono configurati i file delle politiche in un SDK o JRE, consultare l'argomento *IBM SDK Policy files* nel manuale *Security Reference for IBM SDK, Java Technology Edition* per la versione che si sta utilizzando.

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sì
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	no

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sì

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sì
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	no

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no
TLS_RSA_WITH_3DES_EDE_CBC_SHA "2" a pagina 431	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	no "4" a pagina 431

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	no "4" a pagina 431
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	no "4" a pagina 431

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	no "4" a pagina 431
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	no "4" a pagina 431

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	no "4" a pagina 431
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	no "4" a pagina 431

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	no
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	no
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	sì
TLS_AES_128_GCM_SHA256 "3" a pagina 431	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS v1.3	no
TLS_AES_256_GCM_SHA384 "3" a pagina 431	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS v1.3	no

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_CHACHA20_POLY1305_SHA256 "3" a pagina 431	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS v1.3	no
TLS_AES_128_CCM_SHA256 "3" a pagina 431	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS v1.3	no

Tabella 59. CipherSpecs supportati da IBM MQ e i relativi CipherSuites equivalenti (Continua)

CipherSpec **"1" a pagina 431**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite CipherSuite (JRE Oracle)**	**Protocollo**	**Compatibile FIPS 140-2**
TLS_AES_128_CCM_8_SHA256 "3" a pagina 431	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS v1.3	no
Qualsiasi"3" a pagina 431	*ANY	*ANY	Multipli/e	no
ANY_TLS13 "3" a pagina 431	*TLS13	*TLS13	TLS V13	no
ANY_TLS12_OR_HIGHER "3" a pagina 431	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 e versioni successive	no
ANY_TLS13_OR_HIGHER "3" a pagina 431	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 e versioni successive	no

Note:

1. Questo è il valore configurato su un canale in IBM MQ, incluso in un CCDT (binario o JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleto. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.
3. Per poter utilizzare le crittografie TLS v1.3 , il JRE (Java runtime environment) che esegue la tua applicazione deve supportare TLS v1.3.
4. V 9.3.0.17 V 9.3.5.1 Da IBM MQ 9.3.5 CSU 1 e IBM MQ 9.3.0 CSU 17, IBM Java 8 JRE rimuove il supporto per lo scambio di chiavi RSA quando si opera in modalità FIPS.

Configurazione di Ciphersuites e FIPS - compliance in un'applicazione IBM MQ classes for Java

- Un'applicazione che utilizza IBM MQ classes for Java può utilizzare uno dei due metodi per impostare CipherSuite per una connessione:
  - Impostare il campo sslCipherSuite nella classe MQEnvironment sul nome CipherSuite .
  - Impostare la proprietà CMQC.SSL_CIPHER_SUITE_PROPERTY nell'hashtable delle proprietà passata al costruttore MQQueueManager al nome CipherSuite .
- Un'applicazione che utilizza IBM MQ classes for Java può utilizzare uno dei due metodi per applicare la conformità FIPS 140-2:
  - Impostare il campo sslFipsRequired su true nella classe MQEnvironment.
  - Impostare la proprietà CMQC.SSL_FIPS_REQUIRED_PROPERTYin della tabella hash delle proprietà passata al costruttore MQQueueManager su true.

Configurazione della tua applicazione per utilizzare le associazioni IBM Java o Oracle Java CipherSuite

Nota:

V 9.3.3 Per Continuous Delivery da IBM MQ 9.3.3, la Java Proprietà di sistema `com.ibm.mq.cfg.useIBMCipherMappings`, che controlla le associazioni utilizzate, viene rimossa dal prodotto. Da IBM MQ 9.3.3, una cifratura può essere definita come il nome CipherSpec o CipherSuite ed è gestita correttamente da IBM MQ. I seguenti tre file JAR Jackson sono obbligatori se l'applicazione IBM MQ classes for Java crea connessioni TLS sicure a un gestore code:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

Importante: Le seguenti informazioni su `com.ibm.mq.cfg.useIBMCipherMappings` si applicano solo a Long Term Support e Continuous Delivery prima di IBM MQ 9.3.3 .

È possibile configurare se la propria applicazione utilizza le associazioni IBM Java CipherSuite a IBM MQ CipherSpec predefinite oppure le associazioni Oracle CipherSuite a IBM MQ CipherSpec . Pertanto, puoi utilizzare TLS CipherSuites se la tua applicazione utilizza un JRE IBM o un JRE Oracle . La Java Proprietà di sistema `com.ibm.mq.cfg.useIBMCipherMappings` controlla quali associazioni vengono utilizzate. La proprietà può essere uno dei seguenti valori:

vero, true
Utilizzare le associazioni IBM Java CipherSuite a IBM MQ CipherSpec .

Questo è il valore predefinito.

No
Utilizzare le associazioni Oracle CipherSuite a IBM MQ CipherSpec .

Per ulteriori informazioni sull'utilizzo di IBM MQ Java e delle cifrature TLS, consultare il post del blog MQdev MQ Java, Cifrature TLS, JRE e APAR nonIBM IT06775, IV66840, IT09423, IT10837.

Limitazioni di interoperabilità

Alcune CipherSuites potrebbero essere compatibili con più di una IBM MQ CipherSpec, in base al protocollo in uso. Tuttavia, è supportata solo la combinazione CipherSuite/CipherSpec che utilizza la versione di TLS specificata nella tabella 1. Il tentativo di utilizzare le combinazioni non supportate di CipherSuites e CipherSpecs avrà esito negativo con un'eccezione appropriata. Le installazioni che utilizzano una di queste combinazioni CipherSuite/CipherSpec devono essere spostate in una combinazione supportata.

La seguente tabella mostra CipherSuites a cui si applica questa limitazione.

Tabella 60. CipherSuites e i CipherSpecs supportati e non supportati

CipherSuite	CipherSpec TLS supportato	CipherSpec SSL non supportato
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" a pagina 432	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Nota:

1. Deprecated Questa CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleta. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.

Esecuzione di applicazioni IBM MQ classes for Java

Se si scrive un'applicazione (una classe che contiene un metodo main ()), utilizzando la modalità client o bind, eseguire il programma utilizzando l'interprete Java .

Utilizzare il comando:

```
java -Djava.library.path= library_path MyClass
```

dove *library_path* è il percorso delle librerie IBM MQ classes for Java . Per ulteriori informazioni, consultare "IBM MQ classes for Java librerie" a pagina 360.

Attività correlate

Traccia delle applicazioni IBM MQ classes for Java

Traccia dell'adattatore di risorse IBM MQ

Comportamento dipendente dall'ambiente IBM MQ classes for Java

IBM MQ classes for Java consente di creare applicazioni che possono essere eseguite su versioni differenti di IBM MQ. Questa raccolta di argomenti descrive il comportamento delle classi Java dipendenti da tali versioni differenti.

IBM MQ classes for Java fornisce un nucleo di classi, che forniscono funzioni e comportamenti coerenti in tutti gli ambienti. Le funzioni esterne a questo core dipendono dalla capacità del gestore code a cui è connessa l'applicazione.

Tranne dove indicato qui, il comportamento visualizzato è quello descritto nel Riferimento dell'applicazione MQI appropriato per il gestore code.

Classi principali in IBM MQ classes for Java

IBM MQ classes for Java contiene una serie principale di classi, che possono essere utilizzate in tutti gli ambienti.

La seguente serie di classi è considerata classi principali e può essere utilizzata in tutti gli ambienti con solo le variazioni minori elencate in "Limitazioni e variazioni per le classi principali di IBM MQ classes for Java" a pagina 434.

- Ambiente MQ
- Eccezione MQException
- Opzioni MQGetMessage

  Escluso:

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentazione
- MQManagedObject

  Escluso:

  - inquire ()
  - set ()
- Messaggio MQT

  Escluso:

  - groupId
  - messageFlags
  - Numero messageSequence
  - scostamento
  - originalLength
- MQPoolServices
- Eventi MQPoolServices
- MQPoolServicesEventListener
- MQPoolToken
- Opzioni MQPutMessage

  Escluso:

  - KnownDestCount
  - UnknownDestCount
  - InvalidDestCount
  - recordFields
- Processo MQ
- MQQUEUE
- MQQueueManager

  Escluso:

  - begin ()
  - Elenco accessDistribution()
- Gestore MQSimpleConnection
- MQArgomento

- MQC

Nota:

1. Alcune costanti non sono incluse nel core (per i dettagli, consultare “Limitazioni e variazioni per le classi principali di IBM MQ classes for Java” a pagina 434); non utilizzarle in programmi completamente portatili.
2. Alcune piattaforme non supportano tutte le modalità di connessione. Su queste piatteforme, è possibile utilizzare solo le classi principali e le opzioni relative alle modalità supportate.

Limitazioni e variazioni per le classi principali di IBM MQ classes for Java

Le classi principali generalmente si comportano in modo congruente in tutti gli ambienti, anche se le chiamate MQI equivalenti normalmente presentano differenze di ambiente. Il comportamento è come se fosse utilizzato un gestore code AIX, Linuxo Windows , ad eccezione delle seguenti limitazioni e variazioni minori.

*Limitazioni per i valori MQGMO_ * in IBM MQ classes for Java*
Alcuni valori MQGMO_ * non sono supportati da tutti i gestori code.

L'utilizzo dei seguenti valori MQGMO_ * potrebbe causare una MQException generata da MQQueue.get():

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
LOCK_MQGMO
MQGMO_UNLOCK
ORDER LOGICAL_MQGMO_
MESSAGGIO_COMPLETAMENTO_MQGM
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

Inoltre, MQGMO_SET_SIGNAL non è supportato se utilizzato da Java.

*Limitazioni per i valori di MQPMRF_ * in IBM MQ classes for Java*
Questi sono utilizzati solo quando si inseriscono i messaggi in un elenco di distribuzione e sono supportati solo dai gestori code che supportano gli elenchi di distribuzione. Ad esempio, i gestori code z/OS non supportano gli elenchi di distribuzione.

*Restrizioni per i valori MQPMO_ * in IBM MQ classes for Java*
Alcuni valori MQPMO_ * non sono supportati da tutti i gestori code

L'utilizzo dei seguenti valori MQPMO_ * potrebbe causare un'eccezione MQException generata da MQQueue.put() o da MQQueueManager.put ():

ORDER MQPMO_LOGICAL_
ID_CORREL_NEW_MQPMO_
ID MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

*Limitazioni e variazioni per i valori MQCNO_ * in IBM MQ classes for Java*
Alcuni valori `MQCNO_*` non sono supportati.

- La riconnessione automatica del client non è supportata da IBM MQ classes for Java. Indipendentemente dal valore `MQCNO_RECONNECT_*` impostato, la connessione continua a funzionare come se si impostasse `MQCNO_RECONNECT_DISABLED`.

- MQCNO_FASTPATH viene ignorato sui gestori code che non supportano MQCNO_FASTPATH. Viene anche ignorato dalle connessioni client.

*Limitazioni per i valori MQRO_ * in IBM MQ classes for Java*
È possibile impostare le seguenti opzioni del report.

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
DATI_COA_WITH_MQRO_FULL_DATA
DAD_COD MQRO_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_E_SCADENZA

Per ulteriori informazioni, consultare Report.

Differenze varie tra IBM MQ classes for Java su z/OS e altre piattaforme
IBM MQ for z/OS si comporta in modo diverso da IBM MQ su altre piattaforme in alcune aree.

BackoutCount
Un gestore code z/OS restituisce un massimo di BackoutCount di 255, anche se è stato eseguito il backout del messaggio più di 255 volte.

Prefisso coda dinamica predefinito
Quando ci si connette a un gestore code z/OS utilizzando una connessione di bind, il prefisso della coda dinamica predefinito è CSQ. *. Altrimenti, il prefisso della coda dinamica predefinito è AMQ. *.

Costruttore MQQueueManager
La connessione client non è supportata su z/OS. Il tentativo di connessione con opzioni client provoca un'eccezione MQException con MQCC_FAILED e MQRC_ENVIRONMENT_ERROR.

Il costruttore MQQueueManager potrebbe avere esito negativo anche con MQRC_CHAR_CONVERSION_ERROR (se non riesce ad inizializzare la conversione tra le codepage IBM-1047 e ISO8859-1) o MQRC_UCS2_CONVERSION_ERROR (se non riesce ad inizializzare la conversione tra la codepage del gestore code e Unicode). Se l'applicazione ha esito negativo con uno di questi codici di errore, assicurarsi che sia installato il componente National Language Resources di Language Environment e che siano disponibili le tabelle di conversione corrette.

Le tabelle di conversione per Unicode sono installate come parte della funzione facoltativa C/C++ di z/OS . Consultare il manuale *z/OS C/C++ Programming Guide*, SC09-4765, per ulteriori informazioni sull'abilitazione delle conversioni UCS-2 .

Funzioni esterne alle classi principali di IBM MQ classes for Java
IBM MQ classes for Java contiene alcune funzioni specificamente progettate per utilizzare le estensioni API non supportate da tutti i gestori code. Questa raccolta di argomenti descrive come si comportano quando si utilizza un gestore code che non li supporta.

Variazioni nell'opzione del costruttore MQQueueManager
Alcuni dei costruttori MQQueueManager includono un argomento numero intero facoltativo. Alcuni valori di questo argomento non sono accettati su tutte le piattaforme.

Quando un costruttore MQQueueManager include un argomento integer facoltativo, viene associato al campo delle opzioni MQCNO dell'MQI e viene utilizzato per passare da una connessione normale a una connessione fast path. Questo formato esteso del costruttore viene accettato in tutti gli ambienti, se le uniche opzioni utilizzate sono MQCNO_STANDARD_BINDING o MQCNO_FASTPATH_BINDING. Qualsiasi altra opzione causa l'errore del costruttore con MQRC_OPTIONS_ERROR. L'opzione di percorso rapido CMQC.MQCNO_FASTPATH_BINDING viene rispettato solo con una connessione di bind a un gestore code che lo supporta. In altri ambienti, viene ignorato.

Restrizioni sul metodo MQQueueManager.begin ()
Questo metodo può essere utilizzato solo per un gestore code IBM MQ su sistemi AIX, Linux, and Windows in modalità bind. Altrimenti, ha esito negativo con MQRC_ENVIRONMENT_ERROR.

Per ulteriori dettagli, vedere “Coordinamento JTA/JDBC utilizzando IBM MQ classes for Java” a pagina 403.

Variazioni nei campi Opzioni di MQGetMessage
Alcuni gestori code non supportano la struttura MQGMO Versione 2, pertanto è necessario impostare alcuni campi sui valori predefiniti.

Quando si utilizza un gestore code che non supporta la struttura MQGMO versione 2, lasciare impostati i seguenti campi sui valori predefiniti:

GroupStatus
SegmentStatus
Segmentazione

Inoltre, il campo MatchOptions supporta solo MQMO_MATCH_MSG_ID e MQMO_MATCH_CORREL_ID. Se si inseriscono valori non supportati in questi campi, il successivo MQDestination.get() ha esito negativo con MQRC_GMO_ERROR. Se il gestore code non supporta la struttura MQGMO Versione 2, questi campi non vengono aggiornati dopo un esito positivo di MQDestination.get().

Restrizioni negli elenchi di distribuzione in IBM MQ classes for Java
Non tutti i gestori code consentono di aprire un MQDistributionList.

Le seguenti classi vengono utilizzate per creare elenchi di distribuzione:

MQDistributionList
MQDistributionListElemento
MQMessageTracker

È possibile creare e popolare gli elementi MQDistributionLists e MQDistributionListin qualsiasi ambiente, ma non tutti i gestori code consentono di aprire un MQDistributionList. In particolare, i gestori code z/OS non supportano gli elenchi di distribuzione. Il tentativo di apertura di un MQDistributionList quando si utilizza un gestore code di questo tipo risulta in MQRC_OD_ERROR.

Variazioni nei campi Opzioni di MQPutMessage
Se un gestore code non supporta gli elenchi di distribuzione, alcuni campi MQPMO vengono trattati in modo diverso.

Quattro campi in MQPMO vengono rappresentati come le seguenti variabili membro nella classe di opzioni MQPutMessage:

KnownDestCount
UnknownDestCount
InvalidDestCount
recordFields

Questi campi sono principalmente destinati ad essere utilizzati con gli elenchi di distribuzione. Tuttavia, un gestore code che supporta gli elenchi di distribuzione compila i campi DestCount dopo un MQPUT in una singola coda. Ad esempio, se la coda si risolve in una coda locale, knownDestCount è impostato su 1 e gli altri due campi di conteggio sono impostati su 0.

Se il gestore code non supporta gli elenchi di distribuzione, questi valori vengono simulati nel modo seguente:

- Se l'operazione put () ha esito positivo, unknownDestCount è impostato su 1 e gli altri sono impostati su 0.
- Se l'operazione put () ha esito negativo, invalidDestCount è impostato su 1 e gli altri sono impostati su 0.

la variabile recordFields viene utilizzata con gli elenchi di distribuzione. Un valore può essere scritto in recordFields in qualsiasi momento, indipendentemente dall'ambiente. Viene ignorato se l'oggetto MQPutMessageOptions viene utilizzato su un successivo MQDestination.put() o MQQueueManager.put (), piuttosto che MQDistributionList.put ().

Restrizioni nei campi MQMD con IBM MQ classes for Java
Alcuni campi MQMD relativi alla segmentazione dei messaggi devono essere lasciati al loro valore predefinito quando si utilizza un gestore code che non supporta la segmentazione.

I seguenti campi MQMD sono principalmente interessati alla segmentazione dei messaggi:

GroupId
MsgSeqNumber
Offset
MsgFlags
OriginalLength

Se un'applicazione imposta uno qualsiasi di questi campi MQMD su valori diversi da quelli predefiniti, e quindi esegue un comando put () o get () su un gestore code che non li supporta, il comando put () o get () genera un'eccezione MQException con MQRC_MD_ERROR. Un put () o get () con un gestore code di questo tipo lascia sempre i campi MQMD impostati sui valori predefiniti. Non inviare un messaggio raggruppato o segmentato a un'applicazione Java che viene eseguita su un gestore code che non supporta il raggruppamento e la segmentazione dei messaggi.

Se un'applicazione Java tenta di richiamare () un messaggio da un gestore code che non supporta questi campi e il messaggio fisico da recuperare fa parte di un gruppo di messaggi segmentati (ossia, ha valori non predefiniti per i campi MQMD), viene richiamato senza errori. Tuttavia, i campi MQMD in MQMessage non vengono aggiornati, la proprietà del formato MQMessage è impostata su MQFMT_MD_EXTENSION e i dati del messaggio true sono preceduti da una struttura MQMDE che contiene i valori per i nuovi campi.

Limitazioni per IBM MQ classes for Java in CICS Transaction Server
Nell'ambiente CICS Transaction Server per z/OS , solo il thread principale (primo) può emettere chiamate CICS o IBM MQ .

Si noti che le classi IBM MQ JMS non sono supportate per l'utilizzo in un'applicazione CICS Java .

Pertanto, non è possibile condividere gli oggetti MQQueueManager o MQQueue tra i thread in questo ambiente o creare un nuovo MQQueueManager su un thread secondario.

z/OS “Differenze varie tra IBM MQ classes for Java su z/OS e altre piattaforme” a pagina 435 identifica alcune limitazioni e variazioni che si applicano al IBM MQ classes for Java durante l'esecuzione rispetto a un gestore code z/OS . Inoltre, quando si esegue in CICS, i metodi di controllo delle transazioni su MQQueueManager non sono supportati. Invece di emettere MQQueueManager.commit () o MQQueueManager.backout (), le applicazioni utilizzano i metodi di sincronizzazione dell'attività JCICS , Task.commit() e Task.rollback(). La classe Task viene fornita da JCICS nel pacchetto com.ibm.cics.server .

Utilizzo dell'adattatore di risorse IBM MQ
L'adattatore di risorse consente alle applicazioni in esecuzione in un application server di accedere alle risorse IBM MQ . Supporta la comunicazione in entrata e in uscita.

Contenuto dell'adattatore di risorse
V 9.3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per le applicazioni esistenti. Oltre all'adattatore di risorse che supporta Java EE e JMS 2.0, IBM MQ 9.3.0 fornisce un adattatore di risorse che supporta Jakarta Messaging.

JM 3.0 **IBM MQ adattatore di risorse per Jakarta Messaging**
Jakarta Connectors Architecture fornisce una modalità standard di connessione delle applicazioni in esecuzione in un ambiente Jakarta EE a un EIS (Enterprise Information System) come IBM MQ o Db2. L'adattatore di risorse IBM MQ per Jakarta Messaging implementa le interfacce Jakarta Connectors 2.0.0 e contiene IBM MQ classes for Jakarta Messaging. Consente alle applicazioni Jakarta Messaging e agli MDB (message driven bean) in esecuzione su un server delle applicazioni di accedere alle

risorse di un gestore code IBM MQ . L'adattatore risorse supporta sia il dominio point-to-point che il dominio di pubblicazione / sottoscrizione.

JMS 2.0 **IBM MQ adattatore di risorse per JMS 2.0**
Java Platform, Enterprise Edition Connector Architecture (JCA) fornisce un modo standard per connettere le applicazioni in esecuzione in un ambiente Java EE a un EIS (Enterprise Information System) come IBM MQ o Db2. L'adattatore di risorse IBM MQ per JMS 2.0 implementa le interfacce JCA 1.7 e contiene IBM MQ classes for JMS. Consente alle applicazioni JMS e agli MDB (message driven bean), in esecuzione in un server delle applicazioni, di accedere alle risorse di un gestore code IBM MQ . L'adattatore risorse supporta sia il dominio point-to-point che il dominio di pubblicazione / sottoscrizione.

L'adattatore di risorse IBM MQ supporta due tipi di comunicazione tra un'applicazione e un gestore code:

Comunicazione in uscita
Un'applicazione avvia una connessione a un gestore code, quindi invia i messaggi JMS alle destinazioni JMS e riceve i messaggi JMS dalle destinazioni JMS in modo sincrono.

Comunicazione in ingresso
Un messaggio JMS che arriva a una destinazione JMS viene consegnato a MDB, che elabora il messaggio in modo asincrono.

L'adattatore di risorse contiene anche IBM MQ classes for Java. Le classi sono automaticamente disponibili per le applicazioni in esecuzione in un server delle applicazioni in cui è stato distribuito l'adattatore risorse e consentono alle applicazioni in esecuzione in tale server delle applicazioni di utilizzare l'API IBM MQ classes for Java quando accedono alle risorse di un gestore code IBM MQ .

L'utilizzo di IBM MQ classes for Java in un ambiente Java EE è supportato con limitazioni. Per informazioni su queste restrizioni, consultare “Esecuzione di applicazioni IBM MQ classes for Java in Java EE” a pagina 352.

Quale versione dell'adattatore di risorse utilizzare

V 9.3.0 V 9.3.0 La versione dell'adattatore di risorse utilizzato dipende dal fatto che venga distribuito in un server delle applicazioni che supporta Jakarta EE o Java EE:

V 9.3.0 JM 3.0 V 9.3.0 **Jakarta EE**
Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato. L'adattatore di risorse IBM MQ per Jakarta Messaging deve essere distribuito in un application server che supporta Jakarta EE.

Java EE 7
L'adattatore di risorse IBM MQ 8.0 e successive supporta JCA v1.7 e fornisce il supporto JMS 2.0 . Questo adattatore di risorse deve essere distribuito all'interno di un server delle applicazioni Java EE 7 e successivo (consultare “Istruzione di supporto dell'adattatore di risorse IBM MQ” a pagina 439).

È possibile installare l'adattatore risorse IBM MQ 8.0 o successivo su qualsiasi server delle applicazioni certificato come conforme alla specifica Java Platform, Enterprise Edition 7 . Utilizzando l'adattatore di risorse IBM MQ 8.0 o successivo, un'applicazione può connettersi a un gestore code utilizzando il trasporto BINDINGS o CLIENT.

Importante: L'adattatore di risorse IBM MQ 8.0 o successivo può essere distribuito solo in un server delle applicazioni che supporta JMS 2.0.

Utilizzo dell'adattatore di risorse con WebSphere Application Server traditional

Da IBM MQ 9.0, l'adattatore di risorse IBM MQ è preinstallato in WebSphere Application Server traditional 9.0 o versioni successive. Pertanto, non è necessario installare un nuovo adattatore di risorse.

JM 3.0 WebSphere Application Server traditional attualmente non supporta Jakarta EE. Consultare IBM MQ resource adapter statement of support.

Nota: Un IBM MQ 9.0 o un adattatore di risorse successivo può connettersi in modalità di trasporto CLIENT o BINDINGS a qualsiasi gestore code IBM MQ in servizio.

Utilizzo dell'adattatore di risorse con WebSphere Liberty

Per collegarsi a IBM MQ da WebSphere Liberty, è necessario utilizzare l'adattatore di risorse IBM MQ . Poiché Liberty non contiene l'adattatore risorse IBM MQ , è necessario ottenerlo separatamente da Fix Central.

V 9.3.0 V 9.3.0 La versione dell'adattatore di risorse utilizzato dipende dal fatto che venga distribuito in una versione di Liberty che supporta Jakarta EE o Java EE.

Per ulteriori informazioni su come scaricare e installare l'adattatore di risorse, consultare “Installazione dell'adattatore di risorse in Liberty” a pagina 447.

Concetti correlati

“Configurazione dell'adattatore di risorse per la comunicazione in entrata” a pagina 454
Per configurare la comunicazione in entrata, definire le proprietà di uno o più oggetti ActivationSpec .

“Configurazione dell'adattatore di risorse per la comunicazione in uscita” a pagina 472
Per configurare la comunicazione in uscita, definire le proprietà di un oggetto ConnectionFactory e di un oggetto di destinazione gestito.

“Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono i provider di messaggistica Java forniti con IBM MQ. Oltre ad implementare le interfacce definite nelle specifiche JMS e Jakarta Messaging , questi provider di messaggistica aggiungono due serie di estensioni all'API di messaggistica Java .

“Utilizzo di IBM MQ classes for Java” a pagina 349
Utilizzare IBM MQ in un ambiente Java . IBM MQ classes for Java consenti a un'applicazione Java di connettersi a IBM MQ come client IBM MQ o di connettersi direttamente a un gestore code IBM MQ .

Riferimenti correlati

Configurazione del server delle applicazioni per utilizzare il livello di manutenzione dell'adattatore di risorse più recente

Determinazione dei problemi per l'adattatore di risorse IBM MQ

WebSphere Application Server argomenti

Manutenzione dell'adattatore di risorse IBM MQ

Distribuzione delle applicazioni JMS a Liberty per utilizzare il provider di messaggistica IBM MQ

Istruzione di supporto dell'adattatore di risorse IBM MQ

L'adattatore risorse IBM MQ che è necessario utilizzare per la comunicazione tra un'applicazione e un gestore code dipende dal fatto che si stia utilizzando l'API Jakarta Messaging 3.0 o l'API JMS 2.0 .

JMS 2.0 IBM MQ 8.0 o successivo viene fornito con un adattatore di risorse che implementa la specifica JMS 2.0 . Può essere distribuito solo in un server di applicazioni che è compatibile con Java Platform, Enterprise Edition 7 (Java EE 7) e quindi supporta JMS 2.0. Un elenco di server delle applicazioni certificati per Java Platform, Enterprise Edition viene conservato nel sito Web diOracle.

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per le applicazioni esistenti. Oltre all'adattatore di risorse che supporta Java EE e JMS 2.0, IBM MQ 9.3.0 fornisce un adattatore di risorse che supporta Jakarta Messaging. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, consultare Utilizzo delle classi IBM MQ per JMS.

Distribuzione in WebSphere Liberty

WebSphere Liberty 8.5.5 Fix Pack 6 e versioni successive e WebSphere Application Server Liberty 9.0 e versioni successive sono server delle applicazioni certificati Java EE 7 in modo che l'adattatore di risorse IBM MQ 9.0 possa essere distribuito in essi.

V 9.3.0 V 9.3.0 Per utilizzare l'adattatore di risorse IBM MQ per Jakarta Messaging con Liberty, è necessario utilizzare una versione di Liberty che supporta Jakarta EE.

WebSphere Liberty dispone delle seguenti funzioni disponibili per gestire gli adattatori di risorse:

- V 9.3.0 JM 3.0 V 9.3.0 La funzione messaging-3.0 per consentire l'utilizzo degli adattatori di risorse Jakarta Messaging 3.0 .
- La funzione wmqJmsClient-2.0 per consentire l'utilizzo degli adattatori di risorse JMS 2.0 .
- La funzione wmqJmsClient-1.1 per consentire l'utilizzo degli adattatori di risorse JMS 1.1 .

Importante:

- V 9.3.0 JM 3.0 V 9.3.0 L'adattatore di risorse IBM MQ per Jakarta Messaging deve essere distribuito in una versione di Liberty che supporta Jakarta EE. Questo adattatore di risorse non può essere utilizzato con versioni di Liberty che supportano la precedente Java EE specifica non Jakarta EE.
- JMS 2.0 L'adattatore di risorse IBM MQ 8.0 o successivo che supporta JMS 2.0 deve essere distribuito con la funzione wmqJmsClient-2.0 .

Distribuzione in WebSphere Application Server traditional

WebSphere Application Server traditional 9.0 viene fornito con un adattatore risorse IBM MQ 9.0 già installato. Pertanto, non è necessario installare un nuovo adattatore di risorse. L'adattatore di risorse installato può connettersi in modalità di trasporto CLIENT o BINDINGS a tutti i gestori code in esecuzione su una versione supportata di IBM MQ. Per ulteriori informazioni, consultare “Connettività ai gestori code IBM MQ 8.0 o successivi” a pagina 441.

Importante:

- L'adattatore di risorse IBM MQ 9.0 non può essere distribuito nelle versioni di WebSphere Application Server traditional precedenti a IBM MQ 9.0, perché queste versioni non sono Java EE 7 certificate.
- JM 3.0 WebSphere Application Server traditional attualmente non supporta Jakarta EE.

Per ulteriori informazioni sulle versioni dell'adattatore di risorse fornite con WebSphere Application Server, consultare la technote Quale versione di WebSphere MQ Resource Adapter (RA) viene fornita con WebSphere Application Server?

Utilizzo dell'adattatore di risorse con altri server delle applicazioni

Per tutti gli altri server delle applicazioni conformi a Java EE 7 o Jakarta EE , i problemi che si verificano dopo il corretto completamento dell'IVT (Installation Verification Test) dell' IBM MQ adattatore risorse possono essere riportati a IBM per l'analisi della traccia del prodotto IBM MQ e altre informazioni diagnostiche IBM MQ . Se l'adattatore di risorse IBM MQ IVT non può essere eseguito correttamente, è probabile che i problemi riscontrati siano causati da una distribuzione non corretta o da definizioni di risorse non corrette specifiche del server delle applicazioni e che i problemi debbano essere esaminati utilizzando la documentazione del server delle applicazioni e l'organizzazione di supporto per tale server delle applicazioni.

Java Runtime

Il JRE (Java Runtime) utilizzato per eseguire il server delle applicazioni deve essere supportato con il client IBM MQ 9.0 o successivo. Per ulteriori informazioni, consultare Requisiti di sistema per IBM MQ. (Selezionare la versione e il sistema operativo o il report del componente che si desidera visualizzare, quindi seguire il collegamento **Java** elencato nella scheda **Software supportato** .)

Connettività ai gestori code IBM MQ 8.0 o successivi

La gamma completa di funzionalità di JMS 2.0 è disponibile quando ci si connette a un gestore code IBM MQ 8.0 o successivo utilizzando l'adattatore di risorse che è stato distribuito in un server delle applicazioni certificato Java EE 7 . Per utilizzare la funzionalità JMS 2.0 , l'adattatore di risorse deve connettersi al gestore code utilizzando la modalità normale del provider di messaggistica IBM MQ . Per ulteriori informazioni, consultare Configurazione della proprietà JMS **PROVIDERVERSION**.

La gamma completa di funzionalità Jakarta Messaging 3.0 è disponibile quando ci si connette a un gestore code IBM MQ 9.3 utilizzando l'adattatore di risorse distribuito in un server delle applicazioni certificato Jakarta EE .

Estensioni MQ

La specifica JMS 2.0 introduce modifiche al funzionamento di determinati comportamenti. Poiché IBM MQ 8.0 o versioni successive implementa questa specifica, vi sono delle modifiche nel comportamento tra IBM MQ 8.0 e versioni successive e versioni precedenti del prodotto. In IBM MQ 8.0 o versioni successive, IBM MQ classes for JMS include il supporto per la proprietà di sistema Java `com.ibm.mq.jms.SupportMQExtensions` che, se impostata su TRUE, fa sì che queste versioni di IBM MQ ripristinino questi comportamenti a quelli di IBM WebSphere MQ 7.5 o versioni precedenti. Il valore predefinito della proprietà è FALSE.

L'adattatore di risorse IBM MQ 9.0 o successivo include anche una proprietà dell'adattatore di risorse denominata `supportMQExtensions` che ha lo stesso effetto e valore predefinito della proprietà di sistema `com.ibm.mq.jms.SupportMQExtensions` Java . Questa proprietà dell'adattatore di risorse è impostata su false in `ra.xml` per impostazione predefinita.

Se sono impostate sia la proprietà dell'adattatore di risorse che la proprietà di sistema Java , la proprietà di sistema ha la precedenza.

Notare che all'interno dell'adattatore di risorse già distribuito in WebSphere Application Server traditional 9.0, questa proprietà viene impostata automaticamente su TRUE per facilitare la migrazione.

Per ulteriori informazioni, consultare “proprietà SupportMQExtensions” a pagina 330.

Problemi generali

L'interfolizzazione della sessione non è supportata

Alcuni server delle applicazioni forniscono una funzione denominata interfoliazione della sessione, in cui la stessa sessione JMS può essere utilizzata in più transazioni, sebbene sia elencata solo in una alla volta. L'adattatore di risorse IBM MQ non supporta questa funzionalità, il che può portare ai seguenti problemi:

```
Un tentativo di inserire un messaggio in una coda MQ ha esito negativo con codice di errore
2072 (MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Le chiamate a xa_close () non riescono con codice motivo -3 (XAER_PROTO) e un FDC con ID probe AT040010 viene generato sul gestore code IBM MQ a cui si accede dal server delle applicazioni. Per informazioni su come disabilitare questa funzione, consultare la documentazione del server delle applicazioni.

Specifica JTA (Java Transaction API) di come vengono ripristinate le risorse XA per il recupero delle transazioni XA

La sezione 3.4.8 della specifica JTA non definisce un meccanismo specifico mediante il quale le risorse XA vengono ricreate per eseguire il ripristino transazionale XA. Come tale, è compito di ogni singolo gestore transazioni (e, quindi, del server delle applicazioni) come vengono recuperate le risorse XA coinvolte in una transazione XA. È possibile che, per alcuni server delle applicazioni, l'adattatore di risorse IBM MQ 9.0 non implementi i meccanismi specifici del server delle applicazioni utilizzati per eseguire il ripristino transazionale XA.

Connessioni corrispondenti in un factory ManagedConnection

Un server delle applicazioni può richiamare il metodo matchManagedConnections su un'istanza factory ManagedConnectionfornita dall'adattatore di risorse IBM MQ . Una ManagedConnection

viene restituita solo se il metodo ne trova uno che corrisponde sia agli argomenti **javax.security.auth.Subject** che **javax.resource.spi.ConnectionRequestInfo** che sono stati passati al metodo dal server delle applicazioni.

Limitazioni dell'adattatore di risorse IBM MQ

L'adattatore di risorse IBM MQ è supportato su tutte le piattaforme IBM MQ . Tuttavia, quando si utilizza l'adattatore di risorse IBM MQ , alcune funzioni di IBM MQ non sono disponibili o sono limitate.

L'adattatore di risorse IBM MQ ha i seguenti limiti:

- Poiché IBM MQ 8.0, l'adattatore risorse è un adattatore risorse Java Platform, Enterprise Edition 7 (Java EE 7) che fornisce la funzione JMS 2.0 . Di conseguenza, l'adattatore di risorse IBM MQ 8.0 o successivo deve essere installato in un server delle applicazioni certificato Java EE 7 o successivo. Può connettersi in modalità di trasporto bind o client a qualsiasi gestore code in servizio.
- Durante l'esecuzione all'interno del server delle applicazioni WebSphere Liberty , i IBM MQ classes for Java stabilizzati non sono supportati. In altri server delle applicazioni, l'utilizzo di IBM MQ classes for Java non è consigliato. Consultare la IBM technote Utilizzo di WebSphere MQ Java Interfaces in J2EE/JEE Environments per dettagli sulle considerazioni IBM MQ classes for Java in Java EE.
- Quando si esegue all'interno di WebSphere Liberty Application Server su z/OS, è necessario utilizzare la funzione wmqJmsClient-2.0 . Il supporto JCA generico non è possibile per z/OS.
- L'adattatore di risorse IBM MQ non supporta i programmi di uscita del canale scritti in lingue diverse da Java.
- Mentre un server delle applicazioni è in esecuzione, il valore della proprietà sslFipsRequired deve essere true per tutte le risorse JCA o false per tutte le risorse JCA . Si tratta di un requisito anche se le risorse JCA non vengono utilizzate contemporaneamente. Se la proprietà sslFipsRequired ha valori differenti per diverse risorse JCA , IBM MQ emette il codice motivo MQRC_UNSUPPORTED_CIPHER_SUITE, anche se non viene utilizzata una connessione TLS.
- Non è possibile specificare più di un keystore per un server delle applicazioni. Se le connessioni vengono effettuate a più di un gestore code, tutte le connessioni devono utilizzare lo stesso keystore. Questa limitazione non si applica a WebSphere Application Server.
- Se si utilizza una tabella di definizione del canale client (CCDT) con più di una definizione di canale di connessione client adatta, in caso di errore l'adattatore di risorse potrebbe selezionare una definizione di canale differente e quindi un gestore code differente dalla CCDT, il che causerebbe problemi per il ripristino della transazione. L'adattatore di risorse non intraprende alcuna azione per impedire l'utilizzo di tale configurazione ed è responsabilità dell'utente evitare configurazioni che potrebbero causare problemi per il ripristino della transazione.
- La funzionalità dei tentativi di connessione non è supportata per le connessioni in uscita durante l'esecuzione in un contenitore Java EE (EJB/Servlet). Il nuovo tentativo di connessione non è supportato per JMS in uscita quando l'adattatore viene utilizzato in un contesto contenitore JEE , indipendentemente dalla configurazione della transazione o per l'utilizzo non transazionale.
- La riautenticazione, come definita in Sezione 9.1.9 della specifica Java EE Connector Architecture versione 1.7 , delle connessioni JMS non è supportata. Il file `ra.xml` all'interno dell'utilità di gestione risorse IBM MQ deve avere la proprietà denominata **reauthentication-support** impostata sul valore `false`. Un tentativo da parte del server delle applicazioni di autenticare nuovamente una connessione JMS ha come risultato che l'adattatore di risorse IBM MQ genera una javax.resource.spi.SecurityException con il codice messaggio MQJCA1028 .

Attività correlate

Specifica che solo i CipherSpecs certificati FIPS vengono utilizzati al runtime sul client MQI

Riferimenti correlati

FIPS (Federal Information Processing Standards) per AIX, Linux, and Windows

WebSphere Application Server e l'adattatore risorse IBM MQ

L'adattatore di risorse IBM MQ viene utilizzato dalle applicazioni che eseguono la messaggistica JMS con il fornitore di messaggistica IBM MQ in WebSphere Application Server.

Importante: Non utilizzare l'adattatore risorse IBM MQ con WebSphere Application Server 6.0 o WebSphere Application Server 6.1.

WebSphere Application Server traditional 9.0 include una versione dell'adattatore risorse IBM MQ 9.0 . L'adattatore di risorse IBM MQ 9.0 o successivo non può essere distribuito nelle versioni precedenti di WebSphere Application Server, poiché queste versioni non hanno la certificazione Java EE 7 .

WebSphere Application Server traditional attualmente non supporta Jakarta EE. Consultare IBM MQ resource adapter statement of support.

Se si desidera utilizzare un'applicazione JMS per accedere alle risorse di un gestore code IBM MQ da WebSphere Application Server, utilizzare il provider di messaggistica IBM MQ in WebSphere Application Server. Il provider di messaggistica IBM MQ contiene una versione di IBM MQ classes for JMS. Per ulteriori informazioni, consultare la technote Quale versione di WebSphere MQ Resource Adapter (RA) viene fornita con WebSphere Application Server?.

Importante: Non includere alcun file JAR IBM MQ classes for JMS o IBM MQ classes for Java all'interno dell'applicazione. Questa operazione può generare eccezioni ClassCaste può essere difficile da gestire.

Liberty e l'adattatore risorse IBM MQ

L'adattatore di risorse IBM MQ può essere installato in WebSphere Liberty utilizzando una funzione Liberty . La funzione utilizzata dipende dalla versione dell'adattatore risorse che si sta installando. In alternativa, in base ad alcune limitazioni, è possibile installare l'adattatore risorse utilizzando il supporto generico Java Platform, Enterprise Edition Connector Architecture (Java EE JCA).

Limitazioni generali durante l'installazione dell'adattatore di risorse in Liberty

Le seguenti limitazioni si applicano all'adattatore di risorse quando si utilizza la funzione wmqJmsClient-1.1 o wmqJmsClient-2.0 e anche quando si utilizza il supporto JCA generico:

- Le IBM MQ classes for Java non sono supportate in Liberty. Non devono essere utilizzati con la funzione di messaggistica IBM MQ Liberty o con il supporto JCA generico. Per ulteriori informazioni, consultare Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.
- L'adattatore di risorse IBM MQ ha un tipo di trasporto BINDINGS_THEN_CLIENT. Questo tipo di trasporto non è supportato nella funzionalità di messaggistica IBM MQ Liberty .
- Prima di IBM MQ 9.0, la funzione Advanced Message Security (AMS) non era inclusa nella funzione di messaggistica IBM MQ Liberty . Tuttavia, AMS è supportato con un adattatore di risorse IBM MQ 9.0 o successivo.

Nota: Sulle versioni IBM MQ successive a IBM MQ 9.0.0.6 e IBM MQ 9.1.0.1 , è necessario utilizzare la funzione transportSecurity-1.0 invece della funzione ssl-1.0 .

Per ulteriori informazioni, consultare:

Abilitazione delle comunicazioni SSL in Liberty
Valori predefiniti SSL in Liberty
Sicurezza dei trasporti 1.0

Limitazioni quando si utilizzano funzioni Liberty

Con WebSphere Liberty 8.5.5 Fix Pack 2 in WebSphere Liberty 8.5.5 Fix Pack 5 inclusi, era disponibile solo la funzione wmqJmsClient-1.1 e poteva essere utilizzato solo JMS 1.1 . WebSphere Liberty 8.5.5 Fix Pack 6 ha aggiunto la funzione wmqJmsClient-2.0 in modo da poter utilizzare JMS 2.0 .

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato. Per utilizzare l'adattatore di risorse IBM MQ per Jakarta Messaging con Liberty, è necessario utilizzare una versione di Liberty che supporta Jakarta EE. È necessario utilizzare l'adattatore risorse per Jakarta Messaging con la funzione Liberty generic messaging-3.0 .

La funzione che è necessario utilizzare dipende dalla versione dell'adattatore risorse che si sta utilizzando:

- L'adattatore di risorse IBM MQ 8.0.0 Fix Pack 3 e successive IBM MQ 8.0 può essere utilizzato solo con la funzione wmqJmsClient-2.0 .
- L'adattatore di risorse IBM MQ 9.0 può essere utilizzato solo con funzione wmqJmsClient-2.0 .
- V 9.3.0 JM 3.0 V 9.3.0 La funzione messaging-3.0 consente di utilizzare gli adattatori di risorse Jakarta Messaging 3.0 .

Limitazioni quando si utilizza il supporto JCA generico

Se si utilizza il supporto JCA generico, si applicano le seguenti limitazioni:

- È possibile specificare il livello di JMS quando si utilizza il supporto JCA generico. JMS 2.0 e JCA 1.7 devono essere utilizzati solo con adattatori di risorse IBM MQ 8.0.0 Fix Pack 3 e successivi IBM MQ 8.0 .
- Non è possibile eseguire l'adattatore di risorse IBM MQ su z/OS utilizzando il supporto JCA generico. Per poter eseguire l'adattatore di risorse IBM MQ su z/OS, è necessario eseguirlo con la funzione wmqJmsClient-1.1 o wmqJmsClient-2.0 .
- L'ubicazione dell'adattatore di risorse viene specificata utilizzando il seguente elemento xml:

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Importante: Il valore della tag ID può essere EXCEPT per `wmqJms`. Se si utilizza `wmqJms` come ID, Liberty non è in grado di caricare correttamente l'adattatore di risorse. Questo perché `wmqJms` è l'ID utilizzato internamente per fare riferimento alla funzione specifica per IBM MQ. In realtà, crea un'eccezione NullPointer.

I seguenti esempi mostrano alcuni frammenti da un file `server.xml` durante l'esecuzione di JMS 2.0:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

Suggerimento: Si noti l'utilizzo delle funzioni jca-1.7 e jms-2.0 e la mancanza della funzione wmqJmsClient-2.0 .

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Suggerimento: Notare l'utilizzo di `mqJms` per l'ID, che è preferito. Non utilizzare `wmqJms`.

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
```

```
classProviderRef="mqJms"/>
</application>
```

Suggerimento: Notare il riferimento classloaderProviderall'adattatore di risorse mediante l'ID `mqJms`; ciò consente il caricamento di classi specifiche di IBM MQ.

Limitazioni quando si esegue la traccia utilizzando il supporto JCA generico

La traccia e la registrazione non sono integrate nel sistema di traccia Liberty . Invece, la traccia dell'adattatore di risorse IBM MQ deve essere abilitata utilizzando le proprietà di sistema Java o un file di configurazione IBM MQ classes for JMS , come descritto in Traccia delle applicazioni IBM MQ classes for JMS. Per i dettagli su come impostare le proprietà di sistema Java in Liberty, consultare la documentazione diWebSphere Liberty.

Ad esempio, per abilitare la traccia dell'adattatore di risorse IBM MQ in Liberty 19.0.0.9, aggiungere una voce al file Liberty `jvm.options`:

1. Creare un file di testo denominato `jvm.options`.
2. Inserire le opzioni JVM seguenti per abilitare la traccia, una per riga, in questo file:

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. Per applicare queste impostazioni ad un singolo server, salvare `jvm.options` all'indirizzo:

   ```
   ${server.config.dir}/jvm.options
   ```

   Per applicare queste modifiche a tutti Liberty, salvare `jvm.options` in:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   Ciò avrà effetto per tutte le JVM che non dispongono di un file `jvm.options` definito localmente.
4. Riavviare il server per abilitare le modifiche.

Ciò comporta la scrittura della traccia in un file di traccia denominato `MQRA-WLP_<process identifier>.trc` nella directory `<path_to_trace_to>`.

Supporto Liberty XA completo con le tabelle di definizione del canale client

Quando si utilizza WebSphere Liberty 18.0.0.2 in avanti, con IBM MQ 9.2.0 o successive, è possibile utilizzare i gruppi di gestori code all'interno della CCDT (client channel definition table) insieme alle transazioni XA. Ciò significa che ora è possibile utilizzare la distribuzione e la disponibilità del workload, fornite dai gruppi di gestori code, mantenendo l'integrità della transazione.

In caso di errori di connettività a un gestore code, il gestore code deve diventare nuovamente disponibile in modo che la transazione possa essere risolta. Il ripristino della transazione è gestito da Libertye potrebbe essere necessario configurare il gestore transazioni, in modo che sia consentito un periodo di tempo appropriato affinché i gestori code diventino nuovamente disponibili. Per ulteriori informazioni, consultare Transaction manager (transaction) nella documentazione del prodotto WebSphere Liberty .

Si tratta di una funzione lato client, ossia, è necessario un adattatore di risorse IBM MQ 9.2.0 o successivo, non un gestore code IBM MQ 9.2.0 o successivo.

Installazione dell'adattatore di risorse IBM MQ

L'adattatore di risorse IBM MQ viene fornito come file RAR (Resource Archive). Installare il file RAR nel server delle applicazioni. Potrebbe essere necessario aggiungere directory al percorso di sistema.

Informazioni su questa attività

L'adattatore di risorse IBM MQ viene fornito come file RAR (Resource Archive):

- V 9.3.0 JM 3.0 V 9.3.0 Per Jakarta Messaging 3.0, questo file è denominato `wmq.jakarta.jmsra.rar`. Il file RAR contiene IBM MQ classes for Jakarta Messaging e l'implementazione IBM MQ delle interfacce Jakarta Connectors Architecture (JCA).
- JMS 2.0 Per JMS 2.0, questo file è denominato `wmq.jmsra.rar`. Il file RAR contiene IBM MQ classes for JMS e l'implementazione IBM MQ delle interfacce Java EE Connector Architecture (JCA).

Quando si installa l'adattatore di risorse come parte dell'installazione del prodotto IBM MQ , il file RAR viene installato con IBM MQ classes for JMS nella directory visualizzata in Tabella 61 a pagina 446.

Tabella 61. Directory IBM MQ contenente il file RAR per ogni piattaforma

Piattaforma	**Directory**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

È necessario utilizzare l'adattatore di risorse IBM MQ per collegarsi a IBM MQ da un application server. A seconda del server delle applicazioni che si sta utilizzando, l'adattatore risorse potrebbe essere preinstallato o potrebbe essere necessario installarlo da soli.

Tabella 62. Installazione dell'adattatore di risorse in un application server

Server delle applicazioni	**Preinstallato o deve essere installato?**
WebSphere Application Server traditional 9.0	L'adattatore di risorse IBM MQ 9.0 è preinstallato in WebSphere Application Server traditional 9.0. Pertanto, non è necessario installare un nuovo adattatore risorse in WebSphere Application Server traditional 9.0.
WebSphere Liberty	WebSphere Liberty non contiene l'adattatore di risorse IBM MQ , quindi è necessario ottenerlo separatamente da Fix Central.
Altro server delle applicazioni Java EE o Jakarta EE	Ottenere l'adattatore di risorse separatamente da Fix Central, come per WebSphere Liberty.

Procedura

- Se ci si sta collegando a IBM MQ da WebSphere Libertyo da un altro server delle applicazioni Java EE o Jakarta EE , scaricare e installare l'adattatore di risorse IBM MQ come descritto in “Installazione dell'adattatore di risorse in Liberty” a pagina 447.
- Linux AIX
  Per i collegamenti sui sistemi AIX and Linux , accertarsi che la directory contenente le librerie JNI (Java Native Interface) si trovi nel percorso di sistema.

  Per l'ubicazione di questa directory, contenente anche le librerie IBM MQ classes for JMS , consultare “Configurazione delle librerie JNI (Java Native Interface)” a pagina 98.

  Windows Su Windows, questa directory viene aggiunta automaticamente al percorso di sistema durante l'installazione di IBM MQ classes for JMS.

  Suggerimento: Come alternativa all'impostazione del percorso di sistema, l'adattatore di risorse IBM MQ dispone di una proprietà denominata nativeLibraryche può essere utilizzata per specificare

l'ubicazione della libreria JNI. Ad esempio, in WebSphere Liberty , viene configurato come mostrato nel seguente esempio:

```
<wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
```

Le transazioni sono supportate sia in modalità client che bind.

Installazione dell'adattatore di risorse in Liberty

Per connettersi a IBM MQ da WebSphere Libertyo altri server delle applicazioni Java EE o Jakarta EE , è necessario utilizzare l'adattatore di risorse IBM MQ . Poiché Liberty non contiene l'adattatore risorse IBM MQ , è necessario ottenerlo separatamente da Fix Central.

Prima di iniziare

Nota: Le informazioni in questo argomento non si applicano a WebSphere Application Server traditional 9.0. L'adattatore di risorse IBM MQ 9.0 è preinstallato in WebSphere Application Server traditional 9.0. Pertanto, non è necessario installare un nuovo adattatore risorse in questo caso.

Prima di avviare questa attività, assicurarsi di avere un JRE (Java runtime environment) installato sulla macchina e che il JRE sia stato aggiunto al percorso di sistema.

Il programma di installazione Java utilizzato in questo processo di installazione non richiede l'esecuzione come root o come utente specifico. L'unico requisito è che l'utente con cui viene eseguito abbia accesso in scrittura alla directory in cui si desidera inserire i file.

Per le versioni Liberty fino a WebSphere Liberty 8.5.5 Fix Pack 1, se un EJB viene distribuito utilizzando solo la configurazione all'interno di `ejb-jar.xml`, la versione di WebSphere Application Server che il profilo Liberty sta utilizzando deve avere l'APAR PM89890 applicato. Questo metodo di configurazione viene utilizzato per l'IVT (installation verification program) dell'adattatore di risorse, quindi questo APAR è richiesto per l'esecuzione dell'IVT.

V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato. Per utilizzare l'adattatore di risorse IBM MQ per Jakarta Messaging con Liberty, è necessario utilizzare una versione di Liberty che supporta Jakarta EE. Ad esempio, puoi utilizzare la funzione Liberty generic messaging-3.0 .

Informazioni su questa attività

Il file JAR per l'adattatore di risorse che è possibile scaricare da Fix Central è eseguibile. Quando si esegue questo file eseguibile, viene visualizzato l'accordo di licenza IBM MQ , che deve essere accettato. Richiede una directory in cui installare l'adattatore di risorse IBM MQ . Il file RAR dell'adattatore di risorse e il programma IVT (installation verification test) vengono quindi installati in tale directory. È possibile accettare il valore predefinito o specificare un'altra directory, che potrebbe essere la directory degli adattatori di risorse di un server delle applicazioni o qualsiasi altra directory sul sistema. La directory viene creata come parte dell'installazione, se non esiste.

Prima di IBM MQ 9.0, il nome del file da scaricare era nel formato *V.R.M.F*`-WS-MQ-Java-InstallRA.jar`, ad esempio `8.0.0.6-WS-MQ-Java-InstallRA.jar`. Da IBM MQ 9.0, il formato del nome file è *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar`, ad esempio `9.0.0.0-IBM-MQ-Java-InstallRA.jar`.

Dopo aver scaricato e installato l'adattatore di risorse, è possibile configurarlo in WebSphere Liberty.

Procedura

1. Scaricare l'adattatore di risorse IBM MQ da Fix Central.
   a) Fare clic sul link: IBM MQ Adattatore di risorse.
   b) Individuare l'adattatore di risorse per la propria versione di IBM MQ nell'elenco visualizzato di fix disponibili.

Ad esempio:

```
release level: 9.1.4.0-IBM-MQ-Java-InstallRA
Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
Servers
```

Quindi, fare clic sul nome file dell'adattatore di risorse e seguire il processo di download.

2. Avviare l'installazione immettendo il seguente comando dalla directory in cui è stato scaricato il file.

   Da IBM MQ 9.0, il formato del comando è il seguente:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   dove *V.R.M.F* è il numero di versione, release, modifica e fix pack e *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar` è il nome del file scaricato da Fix Central.

   Ad esempio, per installare l'adattatore di risorse IBM MQ per la release IBM MQ 9.1.4 , utilizzare il seguente comando:

   ```
   java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
   ```

   Nota: Per eseguire questa installazione, è necessario avere un JRE installato sulla macchina e aggiunto al percorso di sistema.

   Quando si immettono i comandi, vengono visualizzate le seguenti informazioni:

   ```
   Prima di poter utilizzare, estrarre o installare IBM MQ 9.1, è necessario accettare
   i termini di 1. IBM Accordo di licenza internazionale per la valutazione di
   Programmi 2. IBM International Program License Agreement e ulteriori
   informazioni sulla licenza. Leggere attentamente i seguenti accordi di licenza.

   L'accordo di licenza è visualizzabile separatamente utilizzando il
   Opzione --viewLicenseAgreement.
   Premere Invio per visualizzare i termini della licenza o 'x' per ignorare.
   ```

3. Rivedere e accettare i seguenti termini di licenza:

   a) Per visualizzare la licenza, premere Invio.

      In alternativa, premendo x si salta la visualizzazione della licenza.

      Dopo la visualizzazione della licenza o immediatamente dopo la selezione di x, viene visualizzato il seguente messaggio per indicare che è possibile scegliere di visualizzare ulteriori termini di licenza:

      ```
      Ulteriori informazioni sulla licenza sono visualizzabili separatamente utilizzando il
      Opzione --viewLicenseInfo.
      Premere Invio per visualizzare ora ulteriori informazioni sulla licenza oppure 'x' per
      ignorare.
      ```

   b) Per visualizzare i termini di licenza aggiuntivi, premere Invio.

      In alternativa, premendo x si salta la visualizzazione dei termini di licenza aggiuntivi.

      Dopo la visualizzazione dei termini di licenza aggiuntivi o immediatamente dopo la selezione di x, viene visualizzato il seguente messaggio che richiede di accettare l'accordo di licenza:

      ```
      Scegliendo l'opzione "Accetto" di seguito, si accettano i termini del
      accordo di licenza e clausole nonIBM , se applicabili. Se non si
      accettare, selezionare "Non accetto".

      Seleziona [ 1] Accetto, o [ 2] Non accetto:
      ```

   c) Per accettare l'accordo di licenza e continuare con la selezione della directory di installazione, selezionare 1.

      In alternativa, se si seleziona 2, l'installazione termina immediatamente.

      Se è stato selezionato 1, viene visualizzato il seguente messaggio, che richiede di selezionare una directory di installazione di destinazione:

      ```
      Immettere la directory per i file del prodotto oppure lasciare vuoto per accettare il
      valore predefinito.
      La directory di destinazione predefinita è H: \Liberty\WMQ
      Directory di destinazione per i file del prodotto?
      ```

4. Specificare la directory di installazione per l'adattatore di risorse:

- Se si desidera installare l'adattatore di risorse nell'ubicazione predefinita, premere Invio senza specificare un valore.
- Se si desidera installare l'adattatore di risorse in un percorso diverso da quello predefinito, specificare il nome della directory in cui si desidera installare l'adattatore di risorse e premere Invio.

Una volta installati i file nell'ubicazione selezionata, viene visualizzato un messaggio di conferma come mostrato nel seguente esempio:

```
Estrazione dei file in H: \Liberty\WMQ \wmq
Tutti i file del prodotto sono stati estratti correttamente.
```

Durante l'installazione, viene creata una nuova directory denominata `wmq` all'interno della directory di installazione selezionata e i seguenti file vengono quindi installati nella directory `wmq` :

- Il programma di verifica dell'installazione, `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) o `wmq.jmsra.ivt` (JMS 2.0).
- Il file RAR IBM MQ ,`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 o `wmq.jmsra.rar` (JMS 2.0).

5. JMS 2.0

Opzionale: Configurare l'adattatore di risorse Java EE 7 (JMS 2.0) in WebSphere Liberty Profile.

I passi da eseguire per configurare l'adattatore di risorse in Liberty sono i seguenti. Per ulteriori informazioni, consultare la documentazione del prodottoWebSphere Application Server.

a) Aggiungere la funzione wmqJmsClient-2.0 al file `server.xml` per consentire l'utilizzo dell'adattatore di risorse IBM MQ .

Per ulteriori informazioni, consultare “Quale versione dell'adattatore di risorse utilizzare” a pagina 438.

b) Aggiungere un riferimento al file `wmq.jmsra.rar` (JMS 2.0) installato.

Una configurazione di esempio per supportare i servlet e gli MDB, con JNDI potrebbe essere simile alla seguente:

```
 <featureManager>
    <feature>wmqJmsClient-2.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

6. JM 3.0

Opzionale: Configurare l'adattatore di risorse Jakarta EE 9 (Jakarta Messaging 3.0) in WebSphere Liberty Profile.

I passi da eseguire per configurare l'adattatore di risorse in Liberty sono i seguenti. Per ulteriori informazioni, consultare la documentazione del prodottoWebSphere Application Server.

a) Aggiungere la funzione wmqJmsClient-3.0 al file `server.xml` per utilizzare l'adattatore di risorse IBM MQ .

Per ulteriori informazioni, consultare “Quale versione dell'adattatore di risorse utilizzare” a pagina 438.

b) Aggiungere un riferimento al file `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) installato.

Una configurazione di esempio per supportare i servlet e gli MDB, con JNDI potrebbe essere simile alla seguente:

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
```

```
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

Nota: Se si utilizza Open Liberty, piuttosto che WebSphere Liberty Profile, sarà necessario utilizzare la funzione di supporto dell'adattatore di risorse generico "messagingClient-3.0" al posto di "wmqJmsClient-3.0" e altri aspetti della configurazione saranno diversi. Per ulteriori dettagli, fare riferimento alla documentazione Open Liberty .

Configurazione dell'adattatore di risorse IBM MQ

Per configurare l'adattatore di risorse IBM MQ , definire varie risorse Java Platform, Enterprise Edition Connector Architecture (JCA) e, facoltativamente, le proprietà di sistema. È inoltre necessario configurare l'adattatore risorse per eseguire il programma IVT (Installation Verification Test). Ciò è importante perché il servizio IBM potrebbe richiedere l'esecuzione di questo programma per indicare che qualsiasi server delle applicazioni nonIBM è stato correttamente configurato.

Prima di iniziare

Questa attività presuppone che l'utente abbia già familiarità con JMS e IBM MQ classes for JMS. Molte delle proprietà utilizzate per configurare l'adattatore di risorse IBM MQ sono equivalenti alle proprietà degli oggetti IBM MQ classes for JMS e hanno la stessa funzione.

Informazioni su questa attività

Ogni server delle applicazioni fornisce la propria serie di interfacce di amministrazione. Alcuni server delle applicazioni forniscono interfacce utente grafiche per definire le risorse JCA , ma altri richiedono che l'amministratore scriva piani di distribuzione XML. È quindi al di là dell'ambito di questa documentazione per fornire informazioni su come configurare l'adattatore di risorse IBM MQ per ciascun server delle applicazioni.

I seguenti passi si concentrano quindi solo su ciò che è necessario configurare. Fare riferimento alla documentazione fornita con il server delle applicazioni per informazioni su come configurare un adattatore di risorse JCA .

Procedura

Definire le risorse JCA nelle categorie seguenti:

- Definire le proprietà dell'oggetto ResourceAdapter .

  Queste proprietà, che rappresentano le proprietà globali dell'adattatore di risorse, come il livello di traccia diagnostica, sono descritte in “Configurazione per le proprietà oggetto ResourceAdapter” a pagina 451.
- Definire le proprietà di un oggetto ActivationSpec .

  Queste proprietà determinano come viene attivato un MDB per le comunicazioni in entrata. Per ulteriori informazioni, consultare “Configurazione dell'adattatore di risorse per la comunicazione in entrata” a pagina 454.
- Definire le proprietà di un oggetto ConnectionFactory .

  Il server delle applicazioni utilizza tali proprietà per creare un oggetto JMS ConnectionFactory per la comunicazione in uscita. Per ulteriori informazioni, consultare “Configurazione dell'adattatore di risorse per la comunicazione in uscita” a pagina 472.
- Definire le proprietà di un oggetto di destinazione gestito.

  Il server delle applicazioni utilizza queste proprietà per creare un oggetto JMS Queue o JMS Topic per la comunicazione in uscita. Per ulteriori informazioni, consultare “Configurazione dell'adattatore di risorse per la comunicazione in uscita” a pagina 472.
- Opzionale: Definire un piano di distribuzione per l'adattatore di risorse.

Il file RAR dell'adattatore risorse IBM MQ contiene un file denominato `META-INF/ra.xml`, che contiene un descrittore di distribuzione per l'adattatore risorse. Questo descrittore di distribuzione è definito dallo schema XML in https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd e contiene informazioni sull'adattatore di risorse e sui servizi che fornisce. Un server delle applicazioni potrebbe anche richiedere un piano di distribuzione per l'adattatore di risorse. Questo piano di distribuzione è specifico del server delle applicazioni.

Specificare le proprietà di sistema JVM come richiesto:

- Se si utilizza TLS (Transport Layer Security), specificare le ubicazioni del file keystore e del file truststore come proprietà di sistema JVM, come nel seguente esempio:

```
java ... -Djavax.net.ssl.keyStore=
key_store_location
         -Djavax.net.ssl.trustStore=trust_store_location
         -Djavax.net.ssl.keyStorePassword=key_store_password
```

  Queste proprietà non possono essere proprietà di un oggetto ActivationSpec o ConnectionFactory e non è possibile specificare più di un keystore per un server delle applicazioni. Le proprietà si applicano all'intera JVM e potrebbero quindi influire sul server delle applicazioni se altre applicazioni, in esecuzione sul server delle applicazioni, utilizzano connessioni TLS. Il server delle applicazioni potrebbe anche reimpostare queste proprietà su valori differenti. Per ulteriori informazioni sull'utilizzo di TLS con IBM MQ classes for JMS, consultare “Utilizzo di TLS con IBM MQ classes for JMS” a pagina 257.

- Opzionale: Se necessario, configurare l'adattatore di risorse per registrare i messaggi di avvertenza nel log di output standard del proprio server delle applicazioni.

  I log dell'adattatore risorse, i messaggi di avvertenza e di errore utilizzano lo stesso meccanismo di IBM MQ classes for JMS. Per ulteriori informazioni, vedere Registrazione degli errori per IBM MQ classes for JMS. Ciò significa che, per impostazione predefinita, i messaggi vengono inviati a un file denominato `mqjms.log`. Per configurare l'adattatore di risorse per registrare ulteriormente i messaggi di avvertenza nel log di output standard del server delle applicazioni, impostare la seguente proprietà di sistema JVM per il server delle applicazioni:

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

  Questa è la stessa proprietà di quella utilizzata per controllare la traccia per IBM MQ classes for JMS. Come con IBM MQ classes for JMS, è possibile utilizzare una proprietà di sistema che punta al file `jms.config` (consultare “Il file di configurazione IBM MQ classes for JMS/Jakarta Messaging” a pagina 100). Per informazioni su come impostare una proprietà di sistema JVM, consultare la documentazione del server delle applicazioni.

Configurare l'adattatore di risorse per eseguire il test di verifica dell'installazione

- Configurare l'adattatore di risorse per eseguire il programma IVT (Installation Verification Test) fornito con l'adattatore di risorse IBM MQ .

  Per informazioni su cosa è necessario configurare per eseguire il programma IVT, consultare “Verifica dell'installazione dell'adattatore di risorse” a pagina 492.

  Ciò è importante poiché il servizio IBM potrebbe richiedere l'esecuzione di questo programma per indicare che qualsiasi server delle applicazioni nonIBM è stato configurato correttamente.

  Importante: È necessario configurare l'adattatore risorse prima di poter eseguire il programma.

Configurazione per le proprietà oggetto ResourceAdapter

L'oggetto ResourceAdapter incapsula le proprietà globali dell'adattatore di risorse IBM MQ , come ad esempio il livello di traccia diagnostica. Per definire queste proprietà, utilizzare le funzioni dell'adattatore di risorse, come descritto nella documentazione fornita con il server delle applicazioni.

L'oggetto ResourceAdapter ha due serie di proprietà:

- Proprietà associate alla traccia diagnostica
- Proprietà associate al pool di connessione gestito dall'adattatore di risorse

Il modo in cui si definiscono queste proprietà dipende dalle interfacce di gestione fornite dal server delle applicazioni. Se si utilizza WebSphere Application Server traditional, consultare “WebSphere Application Server traditional configurazione” a pagina 453 o se si utilizza WebSphere Liberty, consultare “WebSphere Liberty configurazione” a pagina 453. Per altri server delle applicazioni, consultare la relativa documentazione del prodotto.

Per ulteriori informazioni relative alla definizione delle proprietà associate alla traccia diagnostica, consultare Traccia dell'adattatore di risorse IBM MQ

L'adattatore di risorse gestisce un pool di connessioni interno di connessioni JMS utilizzate per consegnare i messaggi agli MDB. Tabella 63 a pagina 452 elenca le proprietà dell'oggetto ResourceAdapter associate al pool di connessione.

Tabella 63. Proprietà dell'oggetto ResourceAdapter associate al lotto connessioni

Nome della proprietà	Tipo	Valore predefinito	Descrizione
maxConnections	Stringa	50	Il numero massimo di connessioni a un gestore code IBM MQ e il numero massimo di MDB distribuiti.
connectionConcurrency	Stringa	1	Il numero massimo di MDB per condividere una connessione JMS . La condivisione delle connessioni non è possibile e questa proprietà ha sempre il valore 1.
Conteggio reconnectionRetry	Stringa	5	Il numero massimo di tentativi effettuati dall'adattatore di risorse per riconnettersi a un gestore code IBM MQ se una connessione non riesce.
Intervallo reconnectionRetry	Stringa	300 000	Il tempo, in millisecondi, che l'adattatore di risorse attende prima di tentare la riconnessione a un gestore code IBM MQ .
Conteggio startupRetry	Stringa	0	Il numero predefinito di tentativi di connessione di un MDB all'avvio, se il gestore code non è in esecuzione quando viene avviato il server delle applicazioni.
Intervallo startupRetry	Stringa	30 000	Il tempo di sospensione predefinito tra i tentativi di avvio della connessione (in millisecondi).
supportMQExtensions	Stringa	No	Ripristina il comportamento di IBM MQ JMS a un comportamento precedente aJMS 2.0 . Per ulteriori informazioni, consultare “proprietà SupportMQExtensions” a pagina 330.
nativeLibrary	Stringa	<empty>	Il percorso da utilizzare per caricare la libreria JNI IBM MQ per consentire le connessioni in modalità bind. Windows Su Windows , il percorso di sistema deve contenere anche l'ubicazione dell'installazione IBM MQ corrispondente.

Quando un MDB viene distribuito nel server delle applicazioni, viene creata una nuova connessione JMS e viene avviata una conversazione con il gestore code, purché non venga superato il numero massimo di connessioni specificato dalla proprietà maxConnection . Il numero massimo di MDB è quindi uguale

al numero massimo di connessioni. Se il numero di MDB distribuiti raggiunge questo valore massimo, qualsiasi tentativo di distribuire un altro MDB ha esito negativo. Se un MDB viene arrestato, la sua connessione può essere utilizzata da un altro MDB.

In genere, se devono essere distribuiti molti MDB, è necessario aumentare il valore della proprietà maxConnections .

Le proprietà reconnectionRetryCount e reconnectionRetryInterval gestiscono il comportamento dell'adattatore di risorse quando le connessioni a un gestore code IBM MQ hanno esito negativo, ad esempio a causa di un malfunzionamento della rete. Quando una connessione non riesce, l'adattatore di risorse sospende la consegna dei messaggi a tutti gli MDB forniti da tale connessione per un intervallo specificato dalla proprietà reconnectionRetryInterval. L'adattatore risorse tenta quindi di riconnettersi al gestore code. Se il tentativo ha esito negativo, l'adattatore di risorse effettua ulteriori tentativi di riconnessione a intervalli specificati dalla proprietà reconnectionRetryfino a quando non viene raggiunto il limite imposto dalla proprietà Conteggio reconnectionRetry. Se tutti i tentativi non riescono, la consegna viene arrestata in modo permanente fino a quando gli MDB non vengono riavviati manualmente.

In generale, l'oggetto ResourceAdapter non richiede alcuna gestione. Tuttavia, ad esempio, per abilitare la traccia diagnostica sui sistemi AIX and Linux , è possibile impostare le seguenti proprietà:

```
traceEnabled:    true
traceLevel:      10
```

Queste proprietà non hanno alcun effetto se l'adattatore di risorse non è stato avviato, il che è il caso, ad esempio, quando le applicazioni che utilizzano le risorse IBM MQ sono in esecuzione solo nel contenitore client. In questa situazione, è possibile impostare le proprietà per la traccia diagnostica come proprietà di sistema JVM (Java Virtual Machine). È possibile impostare le proprietà utilizzando l'indicatore -D sul comando **java** , come nel seguente esempio:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

Non è necessario definire tutte le proprietà dell'oggetto ResourceAdapter . Tutte le proprietà non specificate assumono i valori predefiniti. In un ambiente gestito, è preferibile non combinare i due modi di specificare le proprietà. Se si combinano, le proprietà di sistema JVM hanno la precedenza sulle proprietà dell'oggetto ResourceAdapter .

WebSphere Application Server traditional configurazione

Le stesse proprietà sono disponibili per l'adattatore di risorse in WebSphere Application Server traditional, ma devono essere impostate all'interno del pannello delle proprietà dell'adattatore di risorse (consultare Impostazioni del provider JMS nella documentazione del prodotto WebSphere Application Server traditional . La traccia è controllata dalla sezione di diagnostica della configurazione WebSphere Application Server traditional . Per ulteriori informazioni, consultare Utilizzo dei provider per la diagnostica nella documentazione del prodotto WebSphere Application Server traditional .

WebSphere Liberty configurazione

L'adattatore risorse viene configurato utilizzando elementi XML nel file `server.xml` , come mostrato nel seguente esempio:

JM 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

La traccia è abilitata aggiungendo questo elemento XML:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

Configurazione dell'adattatore di risorse per la comunicazione in entrata

Per configurare la comunicazione in entrata, definire le proprietà di uno o più oggetti ActivationSpec .

Le proprietà di un oggetto ActivationSpec determinano il modo in cui un MDB (message driven bean) riceve JMS messaggi da una coda IBM MQ . Il comportamento transazionale di MDB è definito nel relativo descrittore di distribuzione.

Un oggetto ActivationSpec ha due serie di proprietà:

- Proprietà utilizzate per creare una connessione JMS a un gestore code IBM MQ
- Proprietà utilizzate per creare un consumer di connessione JMS che distribuisce i messaggi in modo asincrono man mano che arrivano su una coda specificata

Il modo in cui si definiscono le propriet ... di un oggetto ActivationSpec dipende dalle interfacce di amministrazione fornite dal server delle applicazioni.

Proprietà utilizzate per creare una connessione JMS a un gestore code IBM MQ

Tutte le proprietà in Tabella 64 a pagina 454 sono facoltative.

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
applicationName	String a	• Il nome della classe di richiamo, se disponibile, è regolato in modo da non superare i 28 caratteri. Se non è disponibile, viene utilizzata la stringa `WebSphere MQ Client per Java`.	Il nome con cui un'applicazione è registrata con il gestore code. Questo nome applicazione viene visualizzato dal comando **DISPLAY CONN MQSC/PCF** (dove il campo è denominato **APPLTAG**) o nella visualizzazione di IBM MQ Explorer **Application Connections** (dove il campo è denominato **App name**).
brokerCCDurSubQueue [1]	String a	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un utente della connessione riceve i messaggi di sottoscrizione durevoli

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
brokerCCSubcoda [1]	String a	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un utente della connessione riceve messaggi di sottoscrizione non durevoli
brokerControlCoda [1]	String a	• **SYSTEM.BROKER.CONTROL.QUEUE** • Un nome coda	Il nome della coda di controllo del broker
brokerQueueGestore [1]	String a	• **"" (stringa vuota)** • Un nome gestore code	Il nome del gestore code su cui è in esecuzione il broker
brokerSubCoda [1]	String a	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un utente di messaggi non durevoli riceve i messaggi
brokerVersion [1]	String a	• **non specificato** - Dopo la migrazione del Broker da V6 a V7, impostare questa proprietà in modo che le intestazioni RFH2 non vengano più utilizzate. Dopo la migrazione, questa proprietà non è più rilevante. • **V1** - Per utilizzare un broker IBM MQ di pubblicazione / sottoscrizione broker.This è il valore predefinito se TRANSPORT è impostato su BIND o CLIENT. • **V2** - Per utilizzare un broker di IBM Integration Bus in modalità nativa. Questo valore è il valore predefinito se TRANSPORT è impostato su DIRECT o DIRECTHTTP.	La versione del broker utilizzato
ccdtURL	String a	• **vuoto** • Un URL (Uniform Resource Locator) (URL)	Un URL che identifica il nome e l'ubicazione del file contenente la CCDT (client channel definition table) e specifica il modo in cui è possibile accedere al file
CCSID	String a	• **819** • Un CCSID (coded character set identifier) supportato dalla JVM (Java virtual machine)	Il CCSID (coded character set identifier) per una connessione
canale	String a	• **SYSTEM.DEF.SVRCONN** • Il nome di un canale MQI	Il nome del canale MQi da utilizzare

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
cleanupInterval [2]	int	• **3 600 000** • Un numero intero positivo	L'intervallo, in millisecondi, tra le esecuzioni in background del programma di utilità di ripulitura di pubblicazione / sottoscrizione
cleanupLevel [1]	String a	• **SICURA** • Nessuna • forte • Forza • NONDUR	Il livello di ripulitura per un archivio di sottoscrizioni basato sul broker
clientID	String a	• **vuoto** • Un identificativo client	L'identificativo client per una connessione
cloneSupport	String a	• **DISABLED** - Solo un'istanza di un sottoscrittore di argomenti durevoli può essere eseguita alla volta. • ENABLED - Due o più istanze dello stesso sottoscrittore di argomenti durevoli possono essere eseguite simultaneamente, ma ciascuna istanza deve essere eseguita in una JVM (Java virtual machine) separata.	Se due o più istanze dello stesso sottoscrittore di argomenti durevoli possono essere eseguite contemporaneamente
Ricerca connectionFactory	String a	• **vuoto** • Il nome JNDI per un oggetto ConnectionFactory	Se questa proprietà è impostata, ActivationSpec ricerca un oggetto JMS ConnectionFactory con il nome JNDI specificato nello spazio dei nomi JNDI del server delle applicazioni e utilizza le proprietà di tale oggetto per creare una connessione JMS a un gestore code IBM MQ , con una sola eccezione. L'unica proprietà di ActivationSpec che verrà utilizzata quando si crea la connessione JMS è clientID. Per ulteriori informazioni, consultare “Proprietà ActivationSpec connectionFactoryLookup e destinationLookup” a pagina 468.

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
ConnectionNameList	Stringa	• **host locale (1414)** • Una stringa composta da elementi separati da virgole in cui ogni elemento assume il formato: `HOSTNAME(PORT)` dove *HOSTNAME* è un nome DNS o un indirizzo IP.	Un elenco di nomi di connessione TCP/IP utilizzati per le comunicazioni in entrata. Quando specificato, **connectionNameList** sostituisce le proprietà **hostname** e **port** . Questa proprietà viene utilizzata per riconnettersi ai gestori code a più istanze. **connectionNameList** è simile nel formato a **localAddress**, ma non deve essere confuso con esso. **localAddress** specifica le caratteristiche delle comunicazioni locali, mentre **connectionNameList** specifica come raggiungere un gestore code remoto.
V 9.3.0 V 9.3.0 dynamicallyBalanced [4]	Booleano	• **No** • vero, true	Indica se a questo MDB può essere richiesto di ricevere messaggi da un gestore code differente come parte del bilanciamento dell'applicazione in un cluster uniforme.
FAILIFQUIESCE	Booleano	• **true** • No	Indica se le chiamate a determinati metodi non riescono se il gestore code è in uno stato di sospensione
headerCompression	Stringa	• **Nessuno** • Viene eseguita la compressione dell'intestazione del messaggio SYSTEM - RLE	Un elenco delle tecniche che può essere utilizzato per comprimere i dati di intestazione su una connessione
hostName	Stringa	• **host locale** • Un nome host • Un indirizzo IP	Il nome host o l'indirizzo IP del sistema su cui si trova il gestore code. Le proprietà **hostname** e **port** vengono sostituite dalla proprietà **connectionNameList** quando viene specificata.

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
localAddress	Stringa	• **vuoto** • Una stringa nel formato: `[ host_name ][(low_port [, high_port ])]` dove *host_name* è un nome host o un indirizzo IP, *low_port* e *high_port* sono numeri di porta TCP e le parentesi indicano un componente facoltativo	Per una connessione a un gestore code, questa proprietà specifica uno o entrambi i seguenti elementi: • L'interfaccia di rete locale da utilizzare • La porta locale o l'intervallo di porte locali da utilizzare **localAddress** è simile nel formato a **connectionNameList**, ma non deve essere confuso con esso. **localAddress** specifica le caratteristiche delle comunicazioni locali, mentre **connectionNameList** specifica come raggiungere un gestore code remoto.
messageCompression	Stringa	• **Nessuno** • Un elenco di uno o più dei seguenti valori separati da caratteri vuoti: RLE ZLIBFAST ZLIBHIGH	Un elenco delle tecniche che è possibile utilizzare per comprimere i dati del messaggio su una connessione
messageRetention [1]	Booleano	• **true** - I messaggi indesiderati rimangono nella coda di input • false - I messaggi indesiderati vengono gestiti in base alle opzioni di disposizione	Se il consumer della connessione conserva i messaggi indesiderati nella coda di input
messageSelection [1]	Stringa	• **client** • BROKER	Determina se la selezione del messaggio viene effettuata da IBM MQ classes for JMS o dal broker. La selezione dei messaggi da parte del broker non è supportata quando brokerVersion ha il valore 1.
password	Stringa	• **vuoto** • Una password	La password predefinita da utilizzare quando si crea una connessione al gestore code

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
pollingInterval [1]	int	• **5000** • Qualsiasi numero intero positivo	Se ciascun listener messaggi in una sessione non dispone di alcun messaggio adatto nella propria coda, questo valore è l'intervallo massimo, in millisecondi, che trascorre prima che ciascun listener messaggi provi di nuovo ad ottenere un messaggio dalla propria coda. Se si verifica spesso che non è disponibile alcun messaggio adatto per nessuno dei listener di messaggi in una sessione, aumentare il valore di questa proprietà. Questa proprietà è rilevante solo se TRANSPORT ha il valore BIND o CLIENT.
porta	int	• **1414** • Un numero di porta TCP	La porta su cui è in ascolto il gestore code. Le proprietà **hostname** e **port** vengono sostituite dalla proprietà **connectionNameList** quando viene specificata.
providerVersion	string a	• **non specificato** • Una stringa in uno dei seguenti formati – V.R.M.F – V.R.M – V.R – V dove V, R, M e F sono valori interi maggiori o uguali a zero.	La versione, la release, il livello di modifica e il fix pack del gestore code a cui MDB intende connettersi.
queueManager	String a	• **"" (stringa vuota)** • Un nome gestore code	Il nome del gestore code a cui connettersi
receiveExit [3]	String a	• **vuoto** • Una stringa che comprende uno o più elementi separati da virgole, dove ogni elemento è il nome completo di una classe che implementa l'interfaccia IBM MQ classes for Java , MQReceiveExit	Identifica un programma di uscita di ricezione del canale o una sequenza di programmi di uscita di ricezione da eseguire in successione

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
Inizializzazione di receiveExit	String a	• **vuoto** • Una stringa che comprende uno o più elementi di dati utente separati da virgole	I dati utente passati ai programmi di uscita di ricezione del canale quando vengono richiamati
rescanInterval [1]	int	• **5000** • Qualsiasi numero intero positivo	Quando un utente di messaggi nel dominio point-to-point utilizza un selettore di messaggi per selezionare quali messaggi desidera ricevere, IBM MQ classes for JMS ricerca nella coda IBM MQ i messaggi appropriati nella sequenza determinata dall'attributo **MsgDeliverySequence** della coda. Quando IBM MQ classes for JMS trova un messaggio adatto e lo consegna all'utente, IBM MQ classes for JMS riprende la ricerca del successivo messaggio adatto dalla posizione corrente nella coda. IBM MQ classes for JMS continua a ricercare la coda in questo modo fino a quando non raggiunge la fine della coda o fino a quando l'intervallo di tempo in millisecondi, come determinato dal valore di questa proprietà, non è scaduto. In ogni caso IBM MQ classes for JMS ritorna all'inizio della coda per continuare la ricerca e inizia un nuovo intervallo di tempo.
securityExit [3]	String a	• **vuoto** • Il nome completo di una classe che implementa l'interfaccia IBM MQ classes for Java , MQSecurityExit	Identifica un programma di uscita di sicurezza del canale
Inizializzazione di securityExit	String a	• **vuoto** • Una stringa di dati utente	I dati utente che vengono passati a un programma di uscita di sicurezza del canale quando viene richiamato

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
sendExit [3]	Stringa	• **vuoto** • Una stringa che comprende uno o più elementi separati da virgole, dove ogni elemento è il nome completo di una classe che implementa l'interfaccia IBM MQ classes for Java , MQSendExit	Identifica un programma di uscita di invio del canale o una sequenza di programmi di uscita di invio da eseguire in successione
SENDEXITINIT	Stringa	• **vuoto** • Una stringa che comprende uno o più elementi di dati utente separati da virgole	I dati utente passati ai programmi di uscita di invio del canale quando vengono richiamati
SHARECONVALLOWED	Booleano	• NO - Una connessione client non può condividere il proprio socket. • **YES** - Una connessione client può condividere il proprio socket.	Se una connessione client può condividere il proprio socket con altre connessioni JMS di livello superiore dallo stesso processo allo stesso gestore code, se le definizioni di canale corrispondono
sparseSubscriptions [1]	Booleano	• **false** - Le sottoscrizioni ricevono messaggi corrispondenti frequenti. • true - Le sottoscrizioni ricevono messaggi di corrispondenza non frequenti. Questo valore richiede che la coda di sottoscrizione possa essere aperta per la ricerca.	Controlla la politica di recupero messaggi di un oggetto TopicSubscriber
Archivi sslCert	Stringa	• **vuoto** • Una stringa di uno o più URL LDAP separati da spazi. Ogni URL LDAP ha il formato: `ldap://host_name [: port ]` dove *host_name* è un nome host o un indirizzo IP, *port* è un numero di porta TCP e le parentesi indicano un componente facoltativo.	I server LDAP (Lightweight Directory Access Protocol) che contengono CRL (Certificate Revocation List) da utilizzare su una connessione TLS
SSLCipherSuite	Stringa	• **vuoto** • Il nome di una CipherSuite	La CipherSuite da utilizzare per una connessione TLS
sslFipsRichiesto [2]	Booleano	• **No** • vero, true	Se una connessione TLS deve utilizzare una CipherSuite supportata dal provider FIPS JSSE IBM Java (IBMJSSEFIPS)

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
SSLPeerName	String a	• **vuoto** • Un modello per i DN (distinguished name)	Per una connessione TLS, un modello utilizzato per controllare il DN (distinguished name) nel certificato digitale fornito dal gestore code
SSLResetCount	int	• **0** • Un numero intero compreso tra 0 e 999 999 999	Il numero totale di byte inviati e ricevuti da una connessione TLS prima che le chiavi segrete utilizzate da TLS vengano rinegoziate
Factory sslSocket	String a	Una stringa che rappresenta il nome classe completo di una classe che fornisce un'implementazione dell'interfaccia javax.net.ssl.SSLSocketFactory . Facoltativamente, includere un argomento da passare al metodo constructor, racchiuso tra parentesi.	Tutte le connessioni stabilite nell'ambito dell'oggetto gestito utilizzano i socket ottenuti da questa implementazione dell'interfaccia SSLSocketFactory .
statusRefreshIntervallo [1]	int	• **60000** • Qualsiasi numero intero positivo	L'intervallo, in millesimi di secondo, tra gli aggiornamenti della transazione di lunga durata che rileva quando un sottoscrittore perde la propria connessione al gestore code. Questa proprietà è rilevante solo se **subscriptionStore** ha il valore QUEUE.
subscriptionStore [1]	String a	• **BROKER** • MIGRAZIONE • CODA	Determina dove IBM MQ classes for JMS memorizza i dati persistenti relativi alle sottoscrizioni attive

Tabella 64. Proprietà di un oggetto ActivationSpec utilizzate per la creazione di una connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
transportType	String a	• **client** • BIND • BINDING_THEN_CLIENT	Indica se una connessione a un gestore code utilizza la modalità client o la modalità bind. Se viene specificato il valore `BINDINGS_THEN_CLIENT`, l'adattatore di risorse tenta prima di stabilire una connessione in modalità bind. Se questo tentativo di connessione ha esito negativo, l'adattatore risorse tenta di stabilire una connessione in modalità client. z/OS Se una specifica di attivazione in esecuzione su un sistema WebSphere Application Server for z/OS è stata configurata per utilizzare la modalità di trasporto `BINDINGS_THEN_CLIENT` e una connessione precedentemente stabilita viene interrotta, qualsiasi tentativo di riconnessione da parte della specifica di attivazione tenta prima di utilizzare la modalità di trasporto `BINDINGS`. Se il tentativo di connessione in modalità di trasporto `BINDINGS` ha esito negativo, la specifica di attivazione tenta successivamente una connessione in modalità di trasporto `CLIENT`.
nome utente	String a	• **vuoto** • Un nome utente	Il nome dell'utente predefinito da utilizzare durante la creazione di una connessione a un gestore code
wildcardFormat	String a	• CHAR - Riconosce solo i caratteri jolly, come utilizzato nel broker versione 1 • **TOPIC** - Riconosce solo i caratteri jolly a livello di argomento, come utilizzato nel broker versione 2	La versione della sintassi dei caratteri jolly da utilizzare

Note:

1. Questa proprietà può essere utilizzata con la versione 70 di IBM MQ classes for JMS.
2. Per informazioni importanti sull'utilizzo della proprietà sslFipsRequired, consultare "Limitazioni dell'adattatore di risorse IBM MQ" a pagina 442.
3. Per informazioni su come configurare l'adattatore risorse in modo che possa individuare un'uscita, consultare "Configurazione di IBM MQ classes for JMS per l'utilizzo delle uscite canale" a pagina 284.
4. V 9.3.0 La proprietà dynamicallyBalanced non è supportata insieme al supporto della transazione XA. Se dynamicallyBalanced è "true", l'MDB deve essere configurato per disabilitare le transazioni XA.

Proprietà utilizzate per creare un consumer di connessione JMS

Nota: `destination` e **`destinationType`** devono essere definiti esplicitamente. Tutte le altre proprietà in Tabella 65 a pagina 464 sono facoltative.

Tabella 65. Proprietà di un oggetto ActivationSpec utilizzate per creare un consumer di connessione JMS

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
destinazione	Stringa	Un nome destinazione	La destinazione da cui ricevere i messaggi. La proprietà **`useJNDI`** determina come viene interpretato il valore di questa proprietà.
destinationLookup	Stringa	• **vuoto** • Il nome JNDI per un oggetto di destinazione	Se questa proprietà è impostata, ActivationSpec ricerca un oggetto di destinazione JMS con il nome JNDI specificato nello spazio dei nomi JNDI del server delle applicazioni, quindi utilizza le proprietà di tale oggetto per creare un consumer di connessione JMS , al posto delle altre proprietà specificate in ActivationSpec. Per ulteriori informazioni, consultare "Proprietà ActivationSpec connectionFactoryLookup e destinationLookup" a pagina 468.
destinationType	Stringa	• V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) • V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	Il tipo di destinazione, di coda o di argomento

Tabella 65. Proprietà di un oggetto ActivationSpec utilizzate per creare un consumer di connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
maxMessages	int	• **1** • Un numero intero positivo	Il numero massimo di messaggi che è possibile assegnare ad una sessione server contemporaneamente. Se la specifica di attivazione sta consegnando messaggi a un MDB in una transazione XA, viene utilizzato il valore 1 indipendentemente dall'impostazione di questa proprietà.
Profondità maxPool	int	• **10** • Un numero intero positivo	Il numero massimo di sessioni server nel pool di sessioni server utilizzato dal consumer della connessione
messageSelector	String a	• **vuoto** • Un'espressione del selettore messaggi SQL92	Un'espressione del selettore di messaggi che specifica quali messaggi devono essere consegnati
nonASFTimeout	int	• **0** • Un numero intero positivo	Un valore positivo indica che viene utilizzata la distribuzione non ASF. Il valore è il tempo, in millesimi di secondo, che una richiesta get attende per i messaggi che potrebbero non essere ancora arrivati (una chiamata get with wait). Il valore predefinito, 0, indica che viene utilizzata la consegna ASF. Questo parametro è valido se: • L'applicazione è in esecuzione su WebSphere Application Server 7.0 o versioni successive. • L'applicazione è in esecuzione in WebSphere Liberty utilizzando il livello appropriato della funzione client wmqJms. Per ulteriori informazioni, consultare “Liberty e l'adattatore risorse IBM MQ” a pagina 443.
nonASFRollbackabilitato	Boole ano	• **false** - Il messaggio viene utilizzato anche se l'MDB ha esito negativo • true - L'errore nell'MDB causa il rollback del messaggio nella coda.	Indica se la consegna del messaggio si trova all'interno di un punto di sincronizzazione IBM MQ se l'MDB non è transattato. Ignorato se l'MDB è transattato o se **nonASFTimeout** è impostato su 0.
poolTimeout	int	• **300000** • Un numero intero positivo	Il tempo, in millisecondi, in cui una sessione server inutilizzata viene tenuta aperta nel pool di sessioni server prima di essere chiusa a causa di inattività

Tabella 65. Proprietà di un oggetto ActivationSpec utilizzate per creare un consumer di connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
READAHEADALLOWED	int	• **DESTINATION** - Determina se la lettura anticipata è consentita facendo riferimento alla definizione della coda o dell'argomento. • DISABLED - La lettura anticipata non è consentita. • ENABLED - La lettura anticipata è consentita. • QUEUE - Determinare se la lettura anticipata è consentita facendo riferimento alla definizione della coda. • TOPIC - Determinare se la lettura anticipata è consentita facendo riferimento alla definizione argomento.	Indica se al thread di esplorazione della specifica di attivazione è consentito utilizzare la lettura anticipata per esaminare più messaggi dalla destinazione in un buffer interno, prima di passare alle sessioni del server per un consumo distruttivo. **Nota:** L'abilitazione della lettura anticipata potrebbe risultare in un aumento dei messaggi JMSCC0108 , in una diminuzione delle prestazioni o in entrambi i casi se la velocità di elaborazione MDB non è in grado di tenere il passo con la velocità di visualizzazione dei messaggi dalla destinazione.
readAheadClosePolicy	int	• **ALL** - Tutti i messaggi nel buffer di lettura anticipata interno vengono consegnati all'MDB prima dell'arresto. • CURRENT - Viene completato solo il richiamo MDB corrente, lasciando potenzialmente i messaggi nel buffer di lettura anticipata interno, che vengono quindi eliminati.	Cosa accade ai messaggi nel buffer di lettura anticipata interno quando MDB viene arrestato dall'amministratore.
receiveCCSID	int	• **0** - Utilizza JVM `Charset.defaultCharset` • 1208 - UTF-8 • Un CCSID (coded character set identifier) supportato	Proprietà di destinazione che imposta il CCSID di destinazione per la conversione del messaggio del gestore code. Il valore viene ignorato a meno che **`receiveConversion`** non sia impostato su `QMGR`.
receiveConversion	String a	• **MSG CLI** • QMGR	Proprietà di destinazione che determina se la conversione dati verrà eseguita dal gestore code.
sharedSubscription	Boole ano	• **False** - L'MDB non dovrebbe aprire la sottoscrizione come una sottoscrizione condivisa. • True - L'MDB deve aprire la sottoscrizione come una sottoscrizione condivisa (con le regole che JMS 2.0 implica, consultare la specifica JMS 2.0 all'indirizzo Java.net).	Controlla come un MDB viene guidato da una sottoscrizione condivisa. Per ulteriori informazioni su come utilizzare questa proprietà, consultare “Esempi di come definire la proprietà sharedSubscription” a pagina 470.

Tabella 65. Proprietà di un oggetto ActivationSpec utilizzate per creare un consumer di connessione JMS (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
startTimeout	int	• **10 000** • Un numero intero positivo	Il tempo, in millisecondi, entro il quale deve iniziare la consegna di un messaggio a un MDB dopo che il lavoro per consegnare il messaggio è stato pianificato. Se questo tempo trascorre, viene eseguito il rollback del messaggio sulla coda.
subscriptionDurability	String a	• **NonDurable** - Una sottoscrizione non durevole viene utilizzata per consegnare i messaggi a un MDB che effettua la sottoscrizione all'argomento. • Durevole - Una sottoscrizione durevole viene utilizzata per consegnare i messaggi a un MDB che effettua la sottoscrizione all'argomento.	Se una sottoscrizione durevole o non durevole viene utilizzata per consegnare i messaggi a un MDB che effettua la sottoscrizione all'argomento
subscriptionName	String a	• **"" (stringa vuota)** • Un nome sottoscrizione	Il nome della sottoscrizione durevole
useJNDI	Boolean o	• **false** - La proprietà denominata destination viene interpretata come il nome di una coda IBM MQ o di un argomento. • true - La proprietà denominata destinazione viene interpretata come il nome di uno dei seguenti oggetti nello spazio nomi JNDI del server delle applicazioni: – V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) – V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated Determina come viene interpretato il valore della proprietà denominata destinazione **Nota:** Questa proprietà è obsoleta in IBM MQ 9.0. Utilizzare invece la proprietà destinationLookup .

Dipendenze e conflitti di proprietà

Un oggetto ActivationSpec può avere proprietà in conflitto. Ad esempio, è possibile specificare le proprietà TLS per una connessione in modalità bind. In questo caso, il comportamento è determinato dal tipo di trasporto e dal dominio di messaggistica, che è point-to-point o pubblicazione / sottoscrizione

come determinato dalla proprietà **destinationType** . Tutte le proprietà non applicabili al tipo di trasporto specificato o al dominio di messaggistica vengono ignorate.

Se si definisce una proprietà che richiede la definizione di altre proprietà, ma non si definiscono queste altre proprietà, l'oggetto ActivationSpec genera un'eccezione InvalidPropertyquando il relativo metodo validate () viene richiamato durante la distribuzione di un MDB. L'eccezione viene notificata all'amministratore del server delle applicazioni in un modo che dipende dal server delle applicazioni. Ad esempio, se si imposta la proprietà subscriptionDurability su Durable, che indica che si desidera utilizzare le sottoscrizioni durevoli, è necessario definire anche la proprietà **subscriptionName** .

Se le proprietà denominate **ccdtURL** e **channel** sono entrambe definite, viene generata un'eccezione InvalidProperty. Tuttavia, se si definisce solo la proprietà **ccdtURL** , lasciando la proprietà denominata **channel** con il suo valore predefinito SYSTEM.DEF.SVRCONN, non viene generata alcuna eccezione e la tabella di definizione del canale client identificata dalla proprietà **ccdtURL** viene utilizzata per avviare una connessione JMS .

Proprietà ActivationSpec connectionFactoryLookup e destinationLookup

La specifica JMS 2.0 ha introdotto due nuove proprietà ActivationSpec . Le proprietà connectionFactoryLookup e destinationLookup possono essere fornite con un JNDI nome di oggetto gestito da utilizzare in preferenza rispetto alle altre proprietà ActivationSpec .

Ad esempio, se una factory di connessione è definita in JNDI e il nome JNDI di tale oggetto è specificato nella proprietà connectionFactoryLookup per una specifica di attivazione, tutte le proprietà della factory di connessione definite in JNDI vengono utilizzate al posto delle proprietà in Tabella 64 a pagina 454.

Se una destinazione è definita in JNDI e il nome JNDI è impostato nella proprietà destinationLookup di ActivationSpec, i valori vengono utilizzati di preferenza rispetto ai valori in Tabella 65 a pagina 464. Per ulteriori informazioni su come vengono utilizzate queste due proprietà, consultare “Proprietà ActivationSpec connectionFactoryLookup e destinationLookup” a pagina 468.

Queste proprietà possono essere utilizzate per specificare i nomi JNDI degli oggetti ConnectionFactory e Destination utilizzati in preferenza alle proprietà di ActivationSpec come definito in Tabella 64 a pagina 454 e Tabella 65 a pagina 464.

È importante notare i seguenti punti che descrivono in dettaglio il funzionamento di queste proprietà.

Ricerca connectionFactory
: Il ConnectionFactory ricercato da JNDI viene utilizzato come origine delle proprietà elencate in Tabella 64 a pagina 454. L'oggetto ConnectionFactory non viene utilizzato per creare connessioni JMS , vengono interrogate solo le proprietà dell'oggetto. Queste proprietà di ConnectionFactory sovrascrivono tutte le proprietà definite in ActivationSpec. C'è una sola eccezione a questo. Se per ActivationSpec è impostata la proprietà **ClientID** , il valore di questa proprietà sovrascrive il valore specificato in ConnectionFactory. Questo perché uno scenario comune utilizza un singolo ConnectionFactory con più ActivationSpecs. Ciò semplifica l'amministrazione. Tuttavia, la specifica JMS 2.0 indica che ogni JMS connessione creata da un ConnectionFactory deve avere un **ClientID**univoco. Per questo motivo, ActivationSpecs deve avere la possibilità di sovrascrivere qualsiasi valore impostato su ConnectionFactory. Se non è impostato alcun **ClientID** su ActivationSpec, viene utilizzato qualsiasi valore sul factory di connessione.

destinationLookup
: Una proprietà **Destination** e una proprietà **UseJndi** sono definite in ActivationSpec. Se l'indicatore **UseJndi** è impostato su `true`, il testo specificato nella proprietà di destinazione viene considerato un nome JNDI e un oggetto di destinazione con tale nome JNDI viene ricercato da JNDI.

  La proprietà destinationLookup si comporta esattamente nello stesso modo. Se è stato impostato, un oggetto di destinazione con il nome JNDI specificato dalla proprietà viene ricercato in JNDI. Questa proprietà ha la precedenza sulla proprietà **useJNDI** .

  Deprecated La proprietà useJNDI è obsoleta in IBM MQ 9.0 poiché la proprietà **destinationLookup** è la specifica JMS 2.0 o successiva equivalente all'esecuzione della stessa funzione.

Proprietà ActivationSpec senza equivalenti in IBM MQ classes for JMS

La maggior parte delle proprietà di un oggetto ActivationSpec sono equivalenti alle proprietà di oggetti IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging o ai parametri dei metodi IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging . Tuttavia, tre proprietà di ottimizzazione e una proprietà di utilizzabilità non hanno equivalenti in IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging:

startTimeout
Il tempo, in millesimi di secondo, durante il quale il gestore lavoro del server delle applicazioni attende che le risorse diventino disponibili dopo che l'adattatore risorse pianifica un oggetto Work per consegnare un messaggio a un MDB. Se questo tempo trascorre prima dell'avvio della consegna del messaggio, l'oggetto Work scade, viene eseguito il rollback del messaggio sulla coda e l'adattatore di risorse può quindi tentare nuovamente di consegnare il messaggio. Un'avvertenza viene scritta nella traccia diagnostica, se abilitata, ma non influisce sul processo di consegna dei messaggi. Si potrebbe prevedere che questa condizione si verifichi solo quando il server delle applicazioni sta riscontrando un carico molto elevato. Se la condizione si verifica regolarmente, aumentare il valore di questa proprietà per consentire al gestore lavoro di pianificare la consegna dei messaggi più a lungo.

Profondità maxPool
Il numero massimo di sessioni server nel pool di sessioni server utilizzato da un utente di connessione. Quando viene creata una sessione server, viene avviata una conversazione con un gestore code. Il consumer della connessione utilizza una sessione server per consegnare un messaggio a un MDB. Una maggiore profondità del pool consente la consegna simultanea di più messaggi in situazioni di volumi elevati, ma utilizza più risorse del server delle applicazioni. Se devono essere distribuiti molti MDB, considerare la possibilità di ridurre la profondità del pool per mantenere il carico sul server delle applicazioni ad un livello gestibile. Ogni utente di connessione utilizza il proprio lotto sessioni server, in modo che questa proprietà non definisca il numero totale di sessioni server disponibili per tutti gli utenti di connessione.

poolTimeout
Il tempo, in millisecondi, durante il quale una sessione server inutilizzata viene tenuta aperta nel pool di sessioni server prima di essere chiusa a causa di inattività. Un aumento transitorio del carico di lavoro del messaggio provoca la creazione di sessioni server aggiuntive per distribuire il carico ma, una volta che il carico di lavoro del messaggio è tornato normale, le sessioni server aggiuntive rimangono nel pool e non vengono utilizzate.

Ogni volta che viene utilizzata una sessione server, viene contrassegnata con una data / ora. Periodicamente un thread di scavenger verifica che ogni sessione server sia stata utilizzata entro il periodo specificato da questa proprietà. Se una sessione del server non è stata utilizzata, viene chiusa e rimossa dal pool di sessioni del server. Una sessione server potrebbe non essere chiusa immediatamente dopo la scadenza del periodo specificato, questa proprietà rappresenta il periodo minimo di inattività prima della rimozione.

useJNDI
Per una descrizione di questa proprietà, consultare Tabella 65 a pagina 464.

Distribuzione di un MDB

Per distribuire un MDB, definire prima le proprietà di un oggetto ActivationSpec , specificando le proprietà richieste da MDB. Il seguente esempio è una serie tipica di proprietà che è possibile definire esplicitamente:

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

Il server delle applicazioni utilizza le proprietà per creare un oggetto ActivationSpec , che viene quindi associato a MDB. Le proprietà dell'oggetto ActivationSpec determinano in che modo i messaggi vengono consegnati all'MDB. La distribuzione dell'MDB ha esito negativo se l'MDB richiede transazioni distribuite ma l'adattatore di risorse non supporta le transazioni distribuite. Per informazioni su come installare l'adattatore risorse in modo che le transazioni distribuite siano supportate, consultare “Installazione dell'adattatore di risorse IBM MQ” a pagina 445.

Se più di un MDB sta ricevendo messaggi dalla stessa destinazione, un messaggio inviato nel dominio point-to-point viene ricevuto da un solo MDB, anche se altri MDB sono idonei a ricevere il messaggio. In particolare, se due MDB stanno utilizzando diversi selettori di messaggi e un messaggio in entrata corrisponde a entrambi i selettori di messaggi, solo uno degli MDB riceve il messaggio. L'MDB scelto per ricevere un messaggio non è definito e non è possibile basarsi su un MDB specifico che riceve il messaggio. I messaggi inviati nel dominio di pubblicazione / sottoscrizione vengono ricevuti da tutti gli MDB idonei.

In alcune circostanze, un messaggio consegnato a un MDB potrebbe essere sottoposto a rollback su una coda IBM MQ . Questo rollback può verificarsi, ad esempio, se un messaggio viene consegnato all'interno di un'unità di lavoro di cui viene eseguito il rollback. Un messaggio di cui è stato eseguito il rollback viene recapitato di nuovo, ma un messaggio formattato in modo non corretto potrebbe causare ripetutamente un errore MDB e quindi non può essere consegnato. Tale messaggio è chiamato un messaggio di veleno. È possibile configurare IBM MQ in modo che IBM MQ classes for JMS trasferisca automaticamente un messaggio non elaborabile a un'altra coda per ulteriori indagini o elimini il messaggio.

Per i dettagli su come gestire i messaggi dannosi, consultare “Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236.

Concetti correlati

Specifica che solo i CipherSpecs certificati FIPS vengono utilizzati al runtime sul client MQI

FIPS (Federal Information Processing Standards) per AIX, Linux, and Windows

Attività correlate

Configurazione delle risorse JMS in WebSphere Application Server

Esempi di come definire la proprietà sharedSubscription

È inoltre possibile definire la proprietà sharedSubscription di una specifica di attivazione all'interno di un file WebSphere Liberty `server.xml` . In alternativa, è possibile definire la proprietà all'interno di un MDB (message driven bean) utilizzando le annotazioni.

Esempio: definizione all'interno di un file Liberty `server.xml`

All'interno di un file WebSphere Liberty `server.xml` , definire una specifica di attivazione come mostrato nel seguente esempio. Questo esempio crea una sottoscrizione condivisa durevole a un gestore code su localhost/port 1490.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

Esempio: definizione all'interno di un MDB

È inoltre possibile definire la proprietà sharedSubscription all'interno di MDB utilizzando le annotazioni come mostrato nel seguente esempio:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

Il seguente esempio mostra una parte di codice MDB che usa il metodo delle annotazioni:

V 9.3.0 JM 3.0 V 9.3.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

Concetti correlati

Sottoscrittori e sottoscrizioni

Durata sottoscrizione

“Sottoscrizioni clonate e condivise” a pagina 329

In IBM MQ 8.0 o versioni successive, esistono due metodi per fornire a più consumatori l'accesso alla stessa sottoscrizione. Questi due metodi vengono utilizzati utilizzando sottoscrizioni clonate o sottoscrizioni condivise.

Configurazione dell'adattatore di risorse per la comunicazione in uscita

Per configurare la comunicazione in uscita, definire le proprietà di un oggetto ConnectionFactory e di un oggetto di destinazione gestito.

Esempio di utilizzo della comunicazione in uscita

Quando si utilizza la comunicazione in uscita, un'applicazione in esecuzione in un server delle applicazioni avvia una connessione a un gestore code, quindi invia i messaggi alle code e riceve i messaggi dalle code in modo sincrono. Ad esempio, il seguente metodo servlet, doGet(), utilizza la comunicazione in uscita:

V 9.3.0 JM 3.0 V 9.3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent
```

```
     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

Quando il servlet riceve una richiesta GET HTTP , richiama un oggetto ConnectionFactory e un oggetto Queue dallo spazio dei nomi JNDI e utilizza gli oggetti per inviare un messaggio a una coda IBM MQ . Il servlet riceve quindi il messaggio che ha inviato.

Risorse necessarie per la comunicazione in uscita

Per configurare la comunicazione in uscita, definire le risorse Java EE Connector Architecture (JCA) nelle categorie seguenti:

- Le propriet ... di un oggetto ConnectionFactory, che il server delle applicazioni utilizza per creare un oggetto JMS ConnectionFactory .
- Le proprietà di un oggetto di destinazione gestito, che il server delle applicazioni utilizza per creare un oggetto JMS Queue o JMS Topic.

Il modo in cui si definiscono queste proprietà dipende dalle interfacce di gestione fornite dal server delle applicazioni. Gli oggetti ConnectionFactory, Queue e Topic creati dal server delle applicazioni sono collegati in uno spazio dei nomi JNDI da cui possono essere recuperati da un'applicazione.

Generalmente, si definisce un oggetto ConnectionFactory per ogni gestore code a cui le applicazioni potrebbero aver bisogno di connettersi. Si definisce un oggetto Coda per ogni coda a cui le applicazioni potrebbero dover accedere nel dominio point - to - point. E si definisce un oggetto Argomento per ogni argomento che le applicazioni potrebbero voler pubblicare o sottoscrivere. Un oggetto ConnectionFactory può essere indipendente dal dominio. In alternativa, può essere specifico del dominio, un oggetto factory QueueConnectionper il dominio point-to-point o un oggetto factory TopicConnectionper il dominio di pubblicazione / sottoscrizione.

Suggerimento: Con JMS 2.0, una factory di connessione può essere utilizzata per creare sia connessioni che contesti. Come risultato, è possibile avere un pool di connessioni associato a un factory di connessione che contiene una combinazione di connessioni e contesti. Si consiglia di utilizzare una factory di connessione solo per creare connessioni o contesti. Ciò garantisce che il pool di connessioni per tale factory di connessione contenga solo oggetti di un singolo tipo, rendendo il pool più efficace.

Proprietà di un oggetto ConnectionFactory

Tabella 66 a pagina 473 elenca le proprietà di un oggetto ConnectionFactory . Il server delle applicazioni utilizza queste proprietà per creare un oggetto JMS ConnectionFactory .

Tabella 66. Proprietà di un oggetto ConnectionFactory

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
applicationName	String a	• Il nome della classe di richiamo, se disponibile, è regolato in modo da non superare i 28 caratteri. Se non è disponibile, viene utilizzata la stringa `WebSphere MQ Client per Java` .	Il nome con cui un'applicazione è registrata con il gestore code. Questo nome applicazione viene visualizzato dal comando **`DISPLAY CONN MQSC/PCF`** (dove il campo è denominato **`APPLTAG`**) o nella visualizzazione IBM MQ Esplora **Connessioni applicazione** (dove il campo è denominato **App name**).

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
Coda brokerCCSub	Stringa	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un utente della connessione riceve messaggi di sottoscrizione non durevoli.
Coda brokerControl	Stringa	• **SYSTEM.BROKER.CONTROL.QUEUE** • Un nome coda	Il nome della coda di controllo broker.
Coda brokerPub	Stringa	• **SYSTEM.BROKER.DEFAULT.STREAM** • Un nome coda	Il nome della coda in cui vengono inviati i messaggi pubblicati (la coda di flusso).
Gestore brokerQueue	Stringa	• **"" (stringa vuota)** • Un nome gestore code	Il nome del gestore code su cui è in esecuzione il broker.
Coda brokerSub	Stringa	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un utente di messaggi non durevoli riceve i messaggi. Per ulteriori informazioni, consultare la proprietà BROKERSUBQ .
brokerVersion	Stringa	• **non specificato** - Dopo che il broker è stato migrato da V6 a V7, impostare questa proprietà in modo che le intestazioni RFH2 non vengano più utilizzate. Dopo la migrazione, questa proprietà non è più rilevante. • **V1** - Per utilizzare un broker di pubblicazione / sottoscrizione IBM MQ . Questo valore è il valore predefinito se TRANSPORT è impostato su BIND o CLIENT. • **V2** - Per utilizzare un broker di IBM Integration Bus in modalità nativa. Questo valore è il valore predefinito se TRANSPORT è impostato su DIRECT o DIRECTHTTP.	La versione del broker utilizzato.
ccdtURL	Stringa	• **vuoto** • Un URL (Uniform Resource Locator) (URL)	Un URL che identifica il nome e l'ubicazione del file contenente la CCDT (client channel definition table) e specifica come è possibile accedere al file.
CCSID	Stringa	• **819** • Un CCSID (coded character set identifier) supportato dalla JVM (Java virtual machine)	Il CCSID (coded character set identifier) per una connessione.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
canale	String a	• **SYSTEM.DEF.SVRCONN** • Il nome di un canale MQI	Il nome del canale MQI da utilizzare.
cleanupInterval	int	• **3 600 000** • Un numero intero positivo	L'intervallo, in millisecondi, tra le esecuzioni in background del programma di utilità di ripulitura di pubblicazione / sottoscrizione.
cleanupLevel	String a	• **SICURA** • Nessuna • forte • Forza • NONDUR	Il livello di ripulitura per un archivio sottoscrizioni basato su broker.
clientID	String a	• **vuoto** • Un identificativo client	L'identificativo client per una connessione.
cloneSupport	String a	• **DISABLED** - Solo un'istanza di un sottoscrittore di argomenti durevoli può essere eseguita alla volta. • ENABLED - Due o più istanze dello stesso sottoscrittore di argomenti durevoli possono essere eseguite simultaneamente, ma ciascuna istanza deve essere eseguita in una JVM (Java virtual machine) separata.	Se due o più istanze dello stesso sottoscrittore di argomenti durevoli possono essere eseguite contemporaneamente.
ConnectionNameList	String a	• **host locale (1414)** • Una stringa composta da elementi separati da virgole in cui ogni elemento assume il formato: `HOSTNAME(PORT)` dove *HOSTNAME* è un nome DNS o un indirizzo IP.	Un elenco di nomi di connessione TCP/IP utilizzati per le comunicazioni in uscita. **connectionNameList** sostituisce le proprietà **hostname** e **port** . Questa proprietà viene utilizzata per riconnettersi ai gestori code a più istanze. **connectionNameList** è simile nel formato a **localAddress**, ma non deve essere confuso con esso. **localAddress** specifica le caratteristiche delle comunicazioni locali, mentre **connectionNameList** specifica come raggiungere un gestore code remoto.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
FAILIFQUIESCE	Booleano	• **true** • No	Indica se le chiamate a determinati metodi hanno esito negativo se il gestore code è in uno stato di inattività.
headerCompression	Stringa	• **Nessuno** • Viene eseguita la compressione dell'intestazione del messaggio SYSTEM - RLE.	Un elenco delle tecniche che è possibile utilizzare per comprimere i dati di intestazione su una connessione.
hostName	Stringa	• **host locale** • Un nome host • Un indirizzo IP	Il nome host o l'indirizzo IP del sistema su cui si trova il gestore code. Le proprietà **hostname** e **port** vengono sostituite dalla proprietà **connectionNameList** quando viene specificata.
localAddress	Stringa	• **vuoto** • Una stringa nel formato: `[ host_name ][(low_port [, high_port ])]` dove *host_name* è un nome host o un indirizzo IP, *low_port* e *high_port* sono numeri di porta TCP e le parentesi indicano un componente facoltativo	Per una connessione a un gestore code, questa proprietà specifica uno o entrambi i seguenti valori: • L'interfaccia di rete locale da utilizzare • La porta locale o l'intervallo di porte locali da utilizzare **localAddress** è simile nel formato a **connectionNameList**, ma non deve essere confuso con esso. **localAddress** specifica le caratteristiche delle comunicazioni locali, mentre **connectionNameList** specifica come raggiungere un gestore code remoto.
messageCompression	Stringa	• **Nessuno** • Un elenco di uno o più dei seguenti valori separati da caratteri vuoti: RLE ZLIBFAST ZLIBHIGH	Un elenco delle tecniche che è possibile utilizzare per comprimere i dati del messaggio su una connessione.
messageSelection	Stringa	• **client** • BROKER	Determina se la selezione del messaggio viene effettuata da IBM MQ classes for JMS o dal broker. La selezione del messaggio da parte di un broker non è supportata quando **brokerVersion** ha il valore 1.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
password	String a	• **vuoto** • Una password	La password predefinita da utilizzare quando si crea una connessione al gestore code.
pollingInterval	int	• **5000** • Qualsiasi numero intero positivo	Se ciascun listener messaggi in una sessione non dispone di alcun messaggio adatto nella propria coda, questo valore è l'intervallo massimo, in millisecondi, che trascorre prima che ciascun listener messaggi provi di nuovo ad ottenere un messaggio dalla propria coda. Se si verifica spesso che non è disponibile alcun messaggio adatto per nessuno dei listener di messaggi in una sessione, aumentare il valore di questa proprietà. Questa proprietà è rilevante solo se **TRANSPORT** ha il valore `BIND` o `CLIENT`.
porta	int	• **1414** • Un numero di porta TCP	La porta su cui è in ascolto il gestore code. Le proprietà **hostname** e **port** vengono sostituite dalla proprietà **connectionNameList** quando viene specificata.
providerVersion	string a	• **non specificato** • Una stringa in uno dei seguenti formati – V.R.M.F – V.R.M – V.R – V dove V, R, M e F sono valori interi maggiori o uguali a zero.	La versione, la release, il livello di modifica e il fix pack del gestore code al quale l'applicazione intende connettersi.
Intervallo pubAck	int	• **25** • Un numero intero positivo	Il numero di messaggi pubblicati dal publisher prima che IBM MQ classes for JMS richieda un riconoscimento dal broker.
queueManager	String a	• **"" (stringa vuota)** • Un nome gestore code	Il nome del gestore code a cui connettersi.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)			
Nome della proprietà	**Tipo**	**Valori validi (valore predefinito in grassetto)**	**Descrizione**
receiveExit [3]	String a	• **vuoto** • Una stringa che comprende uno o più elementi separati da virgole, dove ogni elemento è il nome completo di una classe che implementa l'interfaccia IBM MQ classes for Java , MQReceiveExit	Identifica un programma di uscita di ricezione del canale o una sequenza di programmi di uscita di ricezione da eseguire in successione.
Inizializzazione di receiveExit	String a	• **vuoto** • Una stringa che comprende uno o più elementi di dati utente separati da virgole	I dati utente passati ai programmi di uscita di ricezione del canale quando vengono richiamati.
rescanInterval	int	• **5000** • Qualsiasi numero intero positivo	Quando un utente di messaggi nel dominio point-to-point utilizza un selettore di messaggi per selezionare quali messaggi desidera ricevere, IBM MQ classes for JMS ricerca nella coda IBM MQ i messaggi appropriati nella sequenza determinata dall'attributo **MsgDeliverySequence** della coda. Quando IBM MQ classes for JMS trova un messaggio adatto e lo consegna all'utente, IBM MQ classes for JMS riprende la ricerca del successivo messaggio adatto dalla posizione corrente nella coda. IBM MQ classes for JMS continua a ricercare la coda in questo modo fino a quando non raggiunge la fine della coda o fino a quando l'intervallo di tempo in millisecondi, come determinato dal valore di questa proprietà, non è scaduto. In ogni caso IBM MQ classes for JMS ritorna all'inizio della coda per continuare la ricerca e inizia un nuovo intervallo di tempo.
securityExit [3]	String a	• **vuoto** • Il nome completo di una classe che implementa l'interfaccia IBM MQ classes for Java , MQSecurityExit	Identifica un programma di uscita di sicurezza del canale.
Inizializzazione di securityExit	String a	• **vuoto** • Una stringa di dati utente	I dati utente che vengono passati a un programma di uscita di sicurezza del canale quando viene richiamato.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)			
Nome della proprietà	**Tipo**	**Valori validi (valore predefinito in grassetto)**	**Descrizione**
SENDCHECKCOUNT	int	• **0** • Qualsiasi numero intero positivo	Il numero di chiamate di invio da consentire tra il controllo degli errori di inserimento asincrono, all'interno di una singola sessione JMS non transatta.
sendExit [3]	Stringa	• **vuoto** • Una stringa che comprende uno o più elementi separati da virgole, dove ogni elemento è il nome completo di una classe che implementa l'interfaccia IBM MQ classes for Java , MQSendExit	Identifica un programma di uscita di invio del canale o una sequenza di programmi di uscita di invio da eseguire in successione.
SENDEXITINIT	Stringa	• **vuoto** • Una stringa che comprende uno o più elementi di dati utente separati da virgole	I dati utente trasmessi ai programmi di uscita di invio del canale quando vengono richiamati.
SHARECONVALLOWED	Booleano	• NO - Una connessione client non può condividere il proprio socket. • **YES** - Una connessione client può condividere il proprio socket.	Indica se una connessione client può condividere il suo socket con altre connessioni JMS di livello superiore dallo stesso processo allo stesso gestore code, se le definizioni di canale corrispondono.
sparseSubscriptions	Booleano	• **false** - Le sottoscrizioni ricevono messaggi corrispondenti frequenti. • true - Le sottoscrizioni ricevono messaggi di corrispondenza non frequenti. Questo valore richiede che la coda di sottoscrizione possa essere aperta per la ricerca.	Controlla la politica di richiamo messaggi di un oggetto TopicSubscriber .
Archivi sslCert	Stringa	• **vuoto** • Una stringa di uno o più URL LDAP separati da spazi. Ogni URL LDAP ha il formato: `ldap://host_name [: port ]` dove *host_name* è un nome host o un indirizzo IP, *port* è un numero di porta TCP e le parentesi indicano un componente facoltativo.	I server LDAP (Lightweight Directory Access Protocol) che contengono CRL (Certificate Revocation List) da utilizzare su una connessione TLS.
SSLCipherSuite	Stringa	• **vuoto** • Il nome di una CipherSuite	La CipherSuite da utilizzare per una connessione TLS.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
sslFipsRichiesto [2]	Booleano	• **No** • vero, true	Indica se una connessione TLS deve utilizzare una CipherSuite supportata dal provider IBM Java JSSE FIPS (IBMJSSEFIPS).
SSLPeerName	Stringa	• **vuoto** • Un modello per i DN (distinguished name)	Per una connessione TLS, un modello utilizzato per controllare il DN (distinguished name) nel certificato digitale fornito dal gestore code.
SSLResetCount	int	• **0** • Un numero intero compreso tra 0 e 999 999 999	Il numero totale di byte inviati e ricevuti da una connessione TLS prima che le chiavi segrete utilizzate da TLS vengano rinegoziate.
Factory sslSocket	Stringa	Una stringa che rappresenta il nome classe completo di una classe che fornisce un'implementazione dell'interfaccia javax.net.ssl.SSLSocketFactory , includendo facoltativamente un argomento da passare al metodo constructor, racchiuso tra parentesi.	Tutte le connessioni stabilite nell'ambito dell'oggetto di destinazione gestito utilizzano i socket ottenuti da questa implementazione dell'interfaccia SSLSocketFactory .
Intervallo statusRefresh	int	• **60000** • Qualsiasi numero intero positivo	L'intervallo, in millesimi di secondo, tra gli aggiornamenti della transazione di lunga durata che rileva quando un sottoscrittore perde la propria connessione al gestore code. Questa proprietà è rilevante solo se **SUBSTORE** ha il valore QUEUE.
subscriptionStore	Stringa	• **BROKER** • MIGRAZIONE • CODA	Determina dove IBM MQ classes for JMS memorizza i dati persistenti relativi alle sottoscrizioni attive.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
Corrispondenza targetClient	Boolean	• **true** • No	Se un messaggio di risposta, inviato alla coda identificata dal campo di intestazione JMSReplyTo di un messaggio in entrata, dispone di un'intestazione MQRFH2 solo se il messaggio in entrata ha un'intestazione MQRFH2 . È inoltre possibile configurare questa proprietà per una specifica di attivazione. Per ulteriori informazioni, consultare “Configurazione della proprietà di corrispondenza targetClientper una specifica di attivazione” a pagina 490.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
temporaryModel	String a	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • Qualsiasi stringa	Il nome della coda modello da cui vengono create le code temporanee JMS . Utilizzare `SYSTEM.DEFAULT.MODEL.QUEUE` se entrambe le seguenti istruzioni sono vere: • L'applicazione utilizza una coda temporanea che accetterà messaggi non persistenti. • Solo un'applicazione creerà una coda temporanea sul gestore code a cui punta ConnectionFactory alla volta. Notare che `SYSTEM.DEFAULT.MODEL.QUEUE` può essere aperta solo da un'applicazione alla volta. Utilizzare `SYSTEM.JMS.TEMPQ.MODEL` nelle situazioni seguenti: • Quando l'applicazione utilizza una coda temporanea che accetterà messaggi persistenti. • Se più applicazioni possono connettersi al gestore code a cui punta ConnectionFactory e tali applicazioni devono creare code temporanee contemporaneamente. Definire una nuova coda modello con l'attributo **DEFPSIST** impostato su YESe l'attributo **DEFSOPT** impostato su SHARED nella seguente situazione: • Quando l'applicazione utilizza una coda temporanea che accetta messaggi non persistenti e più applicazioni si connetteranno al gestore code a cui fa riferimento ConnectionFactory e tali applicazioni devono creare code temporanee contemporaneamente. Quando viene creata la nuova coda modello, impostare la proprietà **`temporaryModel`** sul nome della nuova coda modello.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
tempQPrefix	String a	• **"" (stringa vuota)** • Un prefisso che può essere utilizzato per formare il nome di una coda dinamica IBM MQ . Le regole per formare il prefisso sono le stesse regole per formare il contenuto del campo **DynamicQName** in un descrittore di oggetti IBM MQ , la struttura MQOD, ma l'ultimo carattere non vuoto deve essere un asterisco (*). Se il valore della proprietà è la stringa vuota, IBM MQ classes for JMS utilizza il valore AMQ.* quando si crea una coda dinamica.	Il prefisso utilizzato per formare il nome di una coda dinamica IBM MQ .
TEMPTOPICPREFIX	String a	Qualsiasi stringa non null costituita solo da caratteri validi per una stringa di argomenti IBM MQ	Quando si creano argomenti temporanei, JMS genera una stringa di argomenti nel formato "TEMP/*TEMPTOPICPREFIX*/*unique_id* " oppure, se questa proprietà viene lasciata con il valore predefinito, solo "TEMP/*unique_id*". La specifica di un **TEMPTOPICPREFIX** non vuoto consente di definire code modello specifiche per la creazione di code gestite per i sottoscrittori di argomenti temporanei creati in questa connessione.

Tabella 66. Proprietà di un oggetto ConnectionFactory (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
transportType	String a	• **client** • BIND • BINDING_THEN_CLIENT	Indica se una connessione a un gestore code utilizza la modalità client o la modalità bind. Se viene specificato il valore `BINDINGS_THEN_CLIENT` , l'adattatore di risorse tenta prima di stabilire una connessione in modalità bind. Se questo tentativo di connessione non riesce, l'adattatore risorse tenta di stabilire una connessione in modalità client. z/OS Se una specifica di attivazione in esecuzione su un sistema WebSphere Application Server for z/OS è stata configurata per utilizzare la modalità di trasporto `BINDINGS_THEN_CLIENT` e una connessione precedentemente stabilita viene interrotta, qualsiasi tentativo di riconnessione da parte della specifica di attivazione tenta prima di utilizzare la modalità di trasporto `BINDINGS` . Se il tentativo di connessione in modalità di trasporto `BINDINGS` ha esito negativo, la specifica di attivazione tenta successivamente una connessione in modalità di trasporto `CLIENT` .
nome utente	String a	• **vuoto** • Un nome utente	Il nome utente predefinito da utilizzare quando si crea una connessione a un gestore code.
wildcardFormat	int	• CHAR - Riconosce solo i caratteri jolly, come utilizzato nel broker versione 1 • TOPIC - Riconosce solo i caratteri jolly a livello di argomento, come utilizzato nel broker versione 2	La versione della sintassi dei caratteri jolly da utilizzare.

Note:

1. Per informazioni importanti sull'utilizzo della proprietà sslFipsRequired, consultare “Limitazioni dell'adattatore di risorse IBM MQ” a pagina 442.
2. Per informazioni su come configurare l'adattatore risorse in modo che possa individuare un'uscita, consultare “Configurazione di IBM MQ classes for JMS per l'utilizzo delle uscite canale” a pagina 284.

Il seguente esempio mostra una serie tipica di proprietà di un oggetto ConnectionFactory :

```
channel:        SYSTEM.DEF.SVRCONN
```

```
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

Proprietà di un oggetto di destinazione gestito

Il server delle applicazioni utilizza le proprietà di un oggetto di destinazione gestito per creare un oggetto JMS Queue o JMS Topic.

Tabella 67 a pagina 485 elenca le proprietà comuni a un oggetto Coda e a un oggetto Argomento.

Tabella 67. Proprietà comuni a un oggetto Coda e a un oggetto Argomento

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
CCSID	Stringa	• **1208** • Un CCSID (coded character set identifier) supportato dalla JVM (Java virtual machine)	Il CCSID (coded character set identifier) per la destinazione.
codifica	Stringa	• **Nativa** • Una stringa di tre caratteri: – Il primo carattere specifica la rappresentazione dei numeri interi binari: - *N* indica la codifica normale. - *R* indica la codifica inversa. – Il secondo carattere specifica la rappresentazione dei numeri interi decimali compressi: - *N* indica la codifica normale. - *R* indica la codifica inversa. – Il terzo carattere specifica la rappresentazione dei numeri a virgola mobile: - *N* indica la codifica IEEE standard. - *R* indica la codifica IEEE inversa. - *3* indica la codifica zSeries . NATIVE è equivalente alla stringa NNN.	La rappresentazione di numeri interi binari, interi decimali compressi e numeri a virgola mobile per la destinazione.
scadenza	Stringa	• **APP** - Il tempo di scadenza di un messaggio è determinato dal produttore del messaggio. • UNLIM - Un messaggio non scade mai. • 0 - Un messaggio non scade mai. • Un numero intero positivo che rappresenta la scadenza di un messaggio in millesimi di secondo.	L'ora di scadenza di un messaggio inviato alla destinazione.
FAILIFQUIESCE	Stringa	• **true** • No	Indica se un tentativo di accesso alla destinazione non riesce se il gestore code è in uno stato di inattività.

Tabella 67. Proprietà comuni a un oggetto Coda e a un oggetto Argomento (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
Stile messageBody	Stringa	• **NON SPECIFICATO** • JMS • MQ	È possibile impostare la proprietà **messageBodyStyle** nelle code e negli argomenti JMS : UNSPECIFIED (predefinito) • Durante l'invio, IBM MQ classes for JMS generare e includere un'intestazione MQRFH2 , in base al valore di WMQ_TARGET_CLIENT. • Quando si riceve, IBM MQ classes for JMS impostare le proprietà del messaggio JMS in base ai valori in MQRFH2, se presenti. MQRFH2 non viene presentato come parte del corpo del messaggio JMS . JMS • Durante l'invio, IBM MQ classes for JMS genera automaticamente un'intestazione MQRFH2 e include l'intestazione nel messaggio IBM MQ . • Quando si riceve, IBM MQ classes for JMS impostare le proprietà del messaggio JMS in base ai valori in MQRFH2, se presenti. MQRFH2 non viene presentato come parte del corpo del messaggio JMS . MQ • Durante l'invio, IBM MQ classes for JMS non genera un MQRFH2. • Quando si riceve, IBM MQ classes for JMS presenta MQRFH2 come parte del corpo del messaggio JMS .

Tabella 67. Proprietà comuni a un oggetto Coda e a un oggetto Argomento (Continua)

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
persistenza	Stringa	• **APP** - La persistenza di un messaggio è determinata dal produttore del messaggio. • QDEF - La persistenza di un messaggio è determinata dall'attributo **DefPersistence** della coda IBM MQ . • PERS - Un messaggio è persistente. • NON - Un messaggio è non persistente. • HIGH - La persistenza di un messaggio è determinata dall'attributo **NonPersistentMessageClass** della coda IBM MQ in base alla spiegazione in "JMS Messaggi persistenti" a pagina 256.	La persistenza di un messaggio inviato alla destinazione.
priorità	Stringa	• **APP** - La priorità di un messaggio è determinata dal produttore del messaggio. • QDEF - La priorità di un messaggio è determinata dall'attributi **DefPriority** della coda IBM MQ . • Un numero intero compreso tra 0, la priorità più bassa, e 9, la priorità più alta.	La priorità di un messaggio inviato alla destinazione.
PUTASYNCALLOWED	Stringa	• QUEUE - Determinare se gli inserimenti asincroni sono consentiti facendo riferimento alla definizione della coda. • TOPIC - Determinare se gli inserimenti asincroni sono consentiti facendo riferimento alla definizione dell'argomento. • DESTINATION - Determinare se gli inserimenti asincroni sono consentiti facendo riferimento alla definizione della coda o dell'argomento. • DISABLED - Gli inserimenti asincroni non sono consentiti. • ENABLED - Le operazioni di inserimento asincrone sono consentite.	Indica se ai produttori di messaggi è consentito utilizzare gli inserimenti asincroni per inviare messaggi a questa destinazione.

Tabella 67. Proprietà comuni a un oggetto Coda e a un oggetto Argomento (Continua)			
Nome della proprietà	**Tipo**	**Valori validi (valore predefinito in grassetto)**	**Descrizione**
READAHEADALLOWED	int	• **DESTINATION** - Determina se la lettura anticipata è consentita facendo riferimento alla definizione della coda o dell'argomento. • DISABLED - La lettura anticipata non è consentita. • ENABLED - La lettura anticipata è consentita. • QUEUE - Determinare se la lettura anticipata è consentita facendo riferimento alla definizione della coda. • TOPIC - Determinare se la lettura anticipata è consentita facendo riferimento alla definizione argomento.	Indica se gli utenti dei messaggi e i browser delle code possono utilizzare la lettura anticipata per ottenere i messaggi non persistenti dalla destinazione in un buffer interno prima di riceverli.
receiveCCSID	int	• **0** - Utilizza JVM `Charset.defaultCharset` • 1208 - UTF-8 • Un CCSID (coded character set identifier) supportato	Proprietà di destinazione che imposta il CCSID di destinazione per la conversione del messaggio del gestore code. Il valore viene ignorato a meno che **`receiveConversion`** non sia impostato su `QMGR`.
receiveConversion	Stringa	• **MSG CLI** • QMGR	Proprietà di destinazione che determina se la conversione dati verrà eseguita dal gestore code.
targetClient	Stringa	• **JMS** - La destinazione di un messaggio è un'applicazione JMS . • MQ - La destinazione di un messaggio è un'applicazione nonJMS IBM MQ .	Se la destinazione di un messaggio inviato alla destinazione è un'applicazione JMS . Un messaggio con una destinazione che è un'applicazione JMS contiene un'intestazione MQRFH2 .

Tabella 68 a pagina 488 elenca le proprietà specifiche di un oggetto Coda.

Tabella 68. Proprietà specifiche di un oggetto Coda			
Nome della proprietà	**Tipo**	**Valori validi (valore predefinito in grassetto)**	**Descrizione**
baseQueueManagerName	Stringa	• **"" (stringa vuota)** • Un nome gestore code	Il nome del gestore code proprietario della coda IBM MQ sottostante.
Nome baseQueue	Stringa	• **"" (stringa vuota)** • Un nome coda	Il nome della coda IBM MQ sottostante.

Tabella 69 a pagina 489 elenca le proprietà specifiche per un oggetto Argomento.

Tabella 69. Proprietà specifiche di un oggetto argomento

Nome della proprietà	Tipo	Valori validi (valore predefinito in grassetto)	Descrizione
Nome baseTopic	String a	• **"" (stringa vuota)** • Un nome argomento	Il nome dell'argomento sottostante.
brokerCCDurSubQueue >	String a	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un utente della connessione riceve messaggi di sottoscrizione durevoli.
brokerDurSubQueue	String a	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • Un nome coda	Il nome della coda da cui un sottoscrittore di argomenti durevoli riceve i messaggi. Consultare la proprietà **BROKEDURRSUBQ** nella documentazione IBM MQ Explorer per ulteriori informazioni.
Coda brokerPub	String a	• **Non impostato** • Un nome coda	Il nome della coda in cui vengono inviati i messaggi pubblicati (la coda di flusso). Il valore di questa proprietà sovrascrive il valore della proprietà **brokerPubQueue** dell'oggetto ConnectionFactory . Tuttavia, se non si imposta il valore di questa proprietà, viene utilizzato il valore della proprietà **brokerPubQueue** dell'oggetto ConnectionFactory .
brokerPubQueueManager	String a	• **"" (stringa vuota)** • Un nome gestore code	Il nome del gestore code proprietario della coda in cui vengono inviati i messaggi pubblicati sull'argomento.
brokerVersion	String a	• **Non impostato** • 1 • 2	La versione del broker utilizzato. Il valore di questa proprietà sovrascrive il valore della proprietà **brokerVersion** dell'oggetto ConnectionFactory . Tuttavia, se non si imposta il valore di questa proprietà, viene utilizzato il valore della proprietà **brokerVersion** dell'oggetto ConnectionFactory .

Il seguente esempio mostra una serie di proprietà di un oggetto Coda:

```
expiry:              UNLIM
persistence:         QDEF
```

```
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

Il seguente esempio mostra una serie di proprietà di un oggetto Argomento:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

Attività correlate

Specifica che solo i CipherSpecs certificati FIPS vengono utilizzati al runtime sul client MQI

Configurazione delle risorse JMS in WebSphere Application Server

Riferimenti correlati

FIPS (Federal Information Processing Standards) per AIX, Linux, and Windows

Configurazione della proprietà di corrispondenza targetClientper una specifica di attivazione

È possibile configurare la propriet. **targetClientMatching** per una specifica di attivazione in modo che un'intestazione MQRFH2 sia inclusa nei messaggi di risposta quando i messaggi di richiesta non contengono un'intestazione MQRFH2 . Ciò significa che tutte le proprietà del messaggio che un'applicazione definisce su un messaggio di risposta vengono incluse quando il messaggio viene inviato.

Informazioni su questa attività

Se un'applicazione MDB (Message - Driven Bean) utilizza messaggi che non contengono un'intestazione MQRFH2 , tramite una specifica di attivazione dell'adattatore di risorse IBM MQ JCA, e successivamente invia messaggi di risposta alla destinazione JMS creata dal campo JMSReplyTo del messaggio di richiesta, i messaggi di risposta devono includere un'intestazione MQRFH2 , anche se i messaggi di richiesta non lo sono, altrimenti le proprietà del messaggio che l'applicazione ha definito in un messaggio di risposta vengono perse.

La proprietà **targetClientMatching** definisce se un messaggio di risposta, inviato alla coda identificata dal campo di intestazione JMSReplyTo di un messaggio in entrata, ha un'intestazione MQRFH2 solo se il messaggio in entrata ha un'intestazione MQRFH2 . È possibile configurare questa proprietà per una specifica di attivazione, sia in WebSphere Application Server traditional che in WebSphere Liberty.

Se si imposta il valore della proprietà **targetClientMatching** su `false`, è possibile includere un'intestazione MQRFH2 in un messaggio di risposta inviato a una destinazione JMS creata dall'intestazione JMSReplyTo di un messaggio di richiesta in entrata che non contenga un MQRFH2. Ciò è dovuto al fatto che la proprietà **targetClient** sulla destinazione JMS viene impostata sul valore `0`, che significa che i messaggi contengono un'intestazione MQRFH2 . La presenza dell'intestazione MQRFH2 nel messaggio in uscita consente la memoria delle proprietà del messaggio definite dall'utente sul messaggio quando viene inviato alla coda IBM MQ .

Se la proprietà **targetClientMatching** è impostata su `true` e un messaggio di richiesta non include un'intestazione MQRFH2 , un'intestazione MQRFH2 non viene inclusa nel messaggio di risposta.

Procedura

- In WebSphere Application Server traditional, utilizzare la console di gestione per definire la proprietà **targetClientMatching** come proprietà personalizzata nella specifica di attivazione IBM MQ :
  a) Nel pannello di navigazione, fare clic su **Risorse -> JMS -> Specifiche attivazione**.
  b) Selezionare il nome della specifica di attivazione che si desidera visualizzare o modificare.
  c) Fare clic su **Proprietà personalizzate -> Nuovo** e immettere i dettagli della nuova proprietà personalizzata.

  Impostare il nome della proprietà su `targetClientMatching`, il tipo su `java.lang.Boolean` e il valore su `false`.

- In WebSphere Liberty, specificare la proprietà **targetClientMatching** nella definizione di una specifica di attivazione all'interno di `server.xml`.

  Ad esempio:

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

Concetti correlati

“Creazione di destinazioni in una applicazione JMS” a pagina 223
Invece di richiamare le destinazioni come oggetti gestiti da un namespace JNDI (Java Naming and Directory Interface), un'applicazione JMS può utilizzare una sessione per creare le destinazioni in modo dinamico al runtime. Un'applicazione può utilizzare un URI (uniform resource identifier) per identificare una coda IBM MQ o un argomento e, facoltativamente, per specificare una o più proprietà di un oggetto Coda o Argomento.

“Configurazione dell'adattatore di risorse per la comunicazione in uscita” a pagina 472
Per configurare la comunicazione in uscita, definire le proprietà di un oggetto ConnectionFactory e di un oggetto di destinazione gestito.

IBM MQ pausa bean basato sui messaggi in WebSphere Liberty

La proprietà **maxSequentialDeliveryFailures** per una specifica di attivazione definisce il numero massimo di errori di consegna di messaggi sequenziali a un'istanza MDB (message - driven bean) che l'adattatore di risorse tollera prima di sospendere MDB.

Prima di iniziare

Devi essere consapevole della serie di eventi che potrebbero causare la sospensione di un MDB in WebSphere Liberty. L'adattatore di risorse considera uno dei seguenti come un errore di distribuzione del messaggio:

- Un'eccezione non verificata generata dal metodo **onMessage** di MDB.
- Un `JMSException` che si verifica nell'elaborazione dell'adattatore di risorse, prima di consegnare il messaggio a MDB.
- Un `JMSException` che si verifica nell'elaborazione dell'adattatore di risorse, dopo la consegna del messaggio all'MDB.
- La transazione XA, o transazione locale, utilizzata per utilizzare il messaggio sottoposto a rollback.
- Nessun thread disponibile nel server delle applicazioni per consegnare il messaggio a MDB.

Informazioni su questa attività

Il valore predefinito della proprietà **maxSequentialDeliveryFailures** è *-1*, il che significa che l'MDB non è mai in pausa. Qualsiasi altro valore negativo viene trattato come *-1*. Un valore di:

- *0* indica che MDB si interrompe al primo errore
- *1* indica che l'MDB si interrompe in caso di due errori sequenziali
- *2* indica che l'MDB si interrompe in seguito a tre errori sequenziali e così via

È possibile configurare questa proprietà per una specifica di attivazione, solo in WebSphere Libertye quando il livello di Liberty è 18.0.0.4o superiore.

Attenzione: Se si imposta l'attributo su un valore non predefinito in qualsiasi ambiente del server delle applicazioni diverso da Liberty, il valore verrà ignorato e verrà scritto un messaggio di avvertenza nel log.

Inoltre, è possibile installare l'adattatore di risorse IBM MQ in WebSphere Liberty come adattatore di risorse generico. Questa operazione disabilita tutte le funzionalità di integrazione di IBM MQ e WebSphere Application Server e impedisce all'adattatore di risorse di rilevare che è in

esecuzione in Liberty. Pertanto, l'impostazione di **maxSequentialDeliveryFailures** su un valore maggiore o uguale a *0* non è supportata e genera un messaggio di avviso nel log.

Procedura

- In WebSphere Liberty, specificare la proprietà **maxSequentialDeliveryFailures** nella definizione di una specifica di attivazione all'interno di `server.xml`.

  Ad esempio:

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

Concetti correlati

“Configurazione dell'adattatore di risorse per la comunicazione in uscita” a pagina 472
Per configurare la comunicazione in uscita, definire le proprietà di un oggetto ConnectionFactory e di un oggetto di destinazione gestito.

Verifica dell'installazione dell'adattatore di risorse

Il programma IVT (Installation Verification Test) per l'adattatore di risorse IBM MQ viene fornito come file EAR. Per utilizzare il programma, è necessario distribuirlo e definire alcuni oggetti come risorse JCA .

Informazioni su questa attività

Il programma IVT (Installation Verification Test) viene fornito come un file EAR (Enterprise Archive) denominato `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) o `wmq.jmsra.ivt.ear` (JMS 2.0). Questo file viene installato con IBM MQ classes for JMS nella stessa directory del file RAR dell'adattatore di risorse IBM MQ , `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) o `wmq.jmsra.rar` (JMS 2.0). Per informazioni su dove sono installati questi file, consultare “Installazione dell'adattatore di risorse IBM MQ” a pagina 445.

È necessario distribuire il programma IVT sul server delle applicazioni. Il programma IVT include un servlet e un MDB che verifica che un messaggio possa essere inviato e ricevuto da una coda IBM MQ . È possibile utilizzare il programma IVT per verificare che l'adattatore di risorse IBM MQ sia stato configurato correttamente per supportare le transazioni distribuite. Se si sta distribuendo l'adattatore di risorse IBM MQ in un server delle applicazioni nonIBM , il servizio IBM potrebbe richiedere di dimostrare che l'IVT funziona per convalidare che l'application server è configurato correttamente.

Prima di poter eseguire il programma IVT, è necessario definire un oggetto ConnectionFactory , un oggetto Queue ed eventualmente un oggetto Activation Specification come risorse JCA e assicurarsi che il server delle applicazioni crei oggetti JMS da queste definizioni e li collega in uno spazio dei nomi JNDI . È possibile scegliere le proprietà degli oggetti in modo che corrispondano alle impostazioni host e porta del proprio QueueManager, ma la seguente serie di proprietà è un semplice esempio:

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1550
queueManager:         QM1
transportType:        CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

Il meccanismo utilizzato per definire gli oggetti ConnectionFactory, Queue e Activation Specification varia a seconda del server delle applicazioni. Ad esempio, per impostare queste proprietà in WebSphere Liberty, aggiungere le voci seguenti al file `server.xml` del server delle applicazioni:

```
JM 3.0<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

```
JMS 2.0<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

Per impostazione predefinita, il programma IVT prevede che un oggetto ConnectionFactory sia collegato nello spazio dei nomi JNDI con il nome jms / ivt/IVTCF e un oggetto Queue con il nome jms / ivt/IVTQueue. È possibile utilizzare nomi diversi, ma in tal caso, è necessario immettere i nomi degli oggetti nella pagina iniziale del programma IVT e modificare il file EAR in modo appropriato.

Dopo aver distribuito il programma IVT e dopo che il server delle applicazioni ha creato gli oggetti JMS e li ha collegati nello spazio nomi JNDI , è possibile avviare il programma IVT completando la seguente procedura.

Procedura

1. Avviare il programma IVT immettendo un URL nel seguente formato nel browser Web:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   dove *app_server_host* è l'indirizzo IP o il nome del sistema su cui è in esecuzione il server delle applicazioni e *port* è il numero della porta TCP su cui è in ascolto il server delle applicazioni. Di seguito è riportato un esempio:

   ```
   http://localhost:9080/WMQ_IVT/
   ```

Ecco un esempio della pagina iniziale visualizzata dal programma IVT.

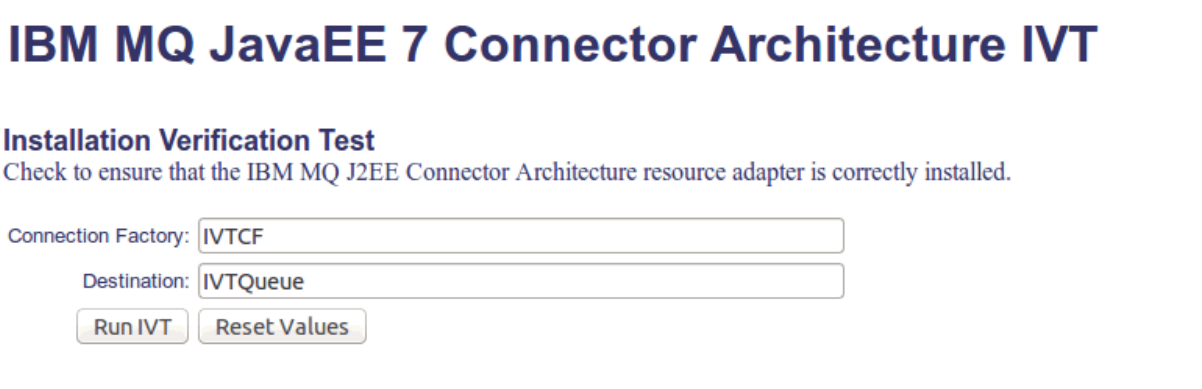


Figura 46. La pagina iniziale del programma IVT

2. Per eseguire il test, fare clic su **Esegui IVT**.

   Di seguito è riportato un esempio della pagina visualizzata se l'IVT ha esito positivo.

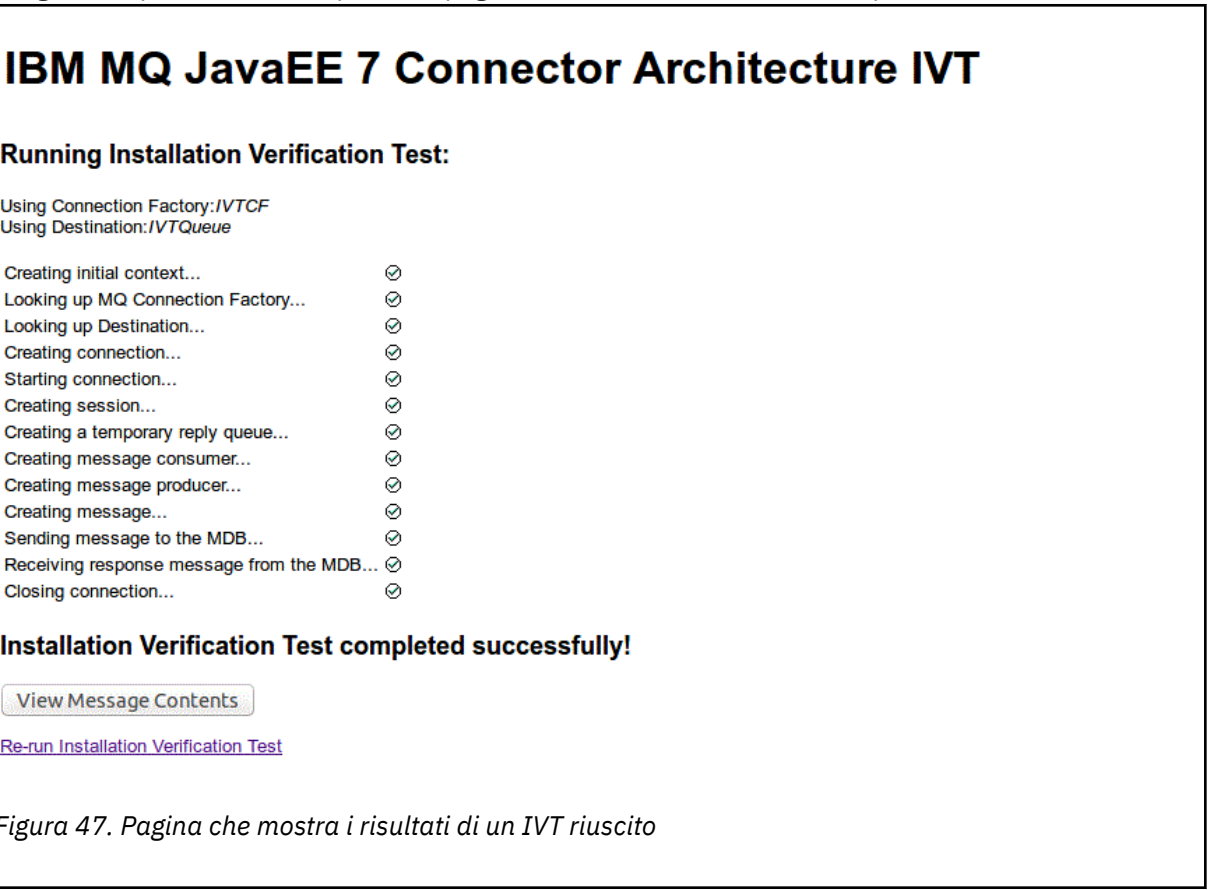


Figura 47. Pagina che mostra i risultati di un IVT riuscito

Di seguito è riportato un esempio della pagina che viene visualizzata se l'IVT ha esito negativo. Per ulteriori informazioni sulla causa dell'errore, fare clic su **Visualizza traccia di stack**.

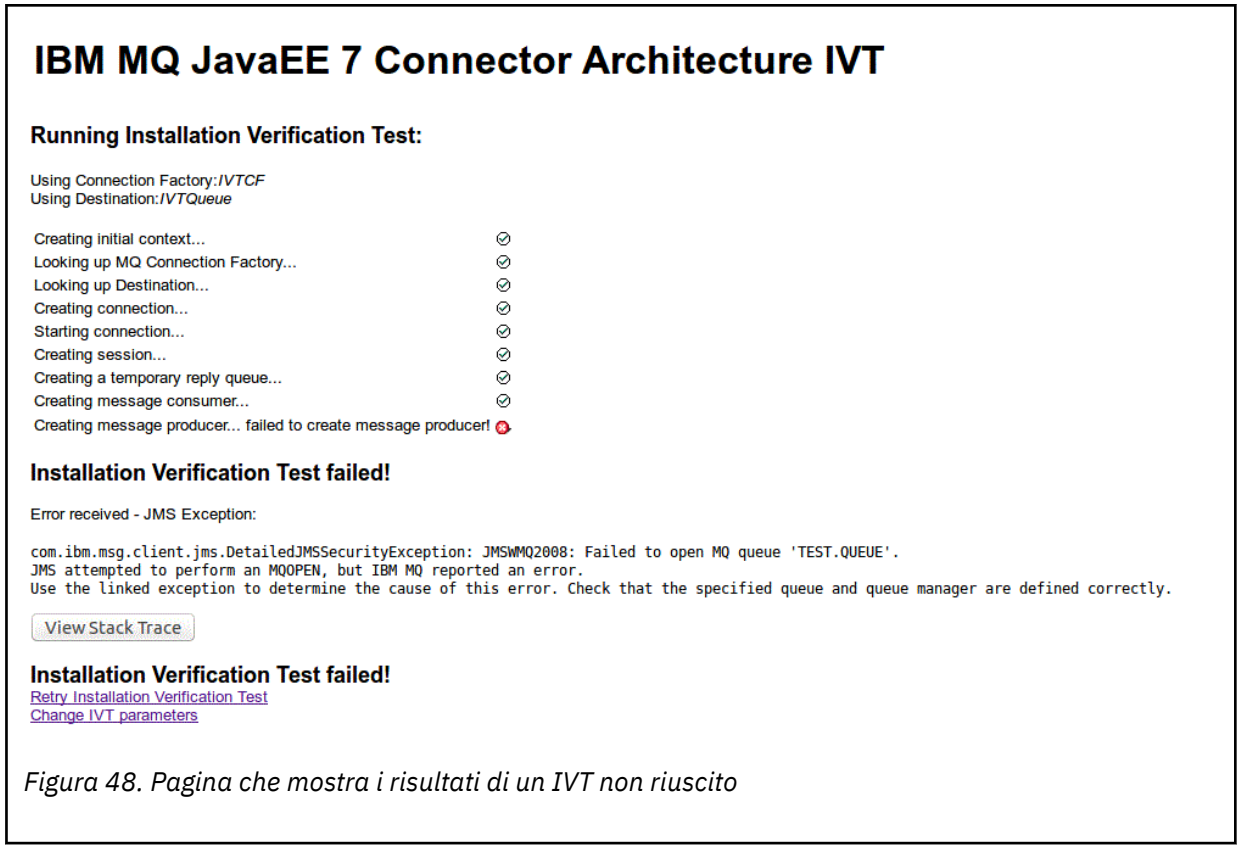


Figura 48. Pagina che mostra i risultati di un IVT non riuscito

Windows Installazione e verifica dell'adattatore di risorse nel server GlassFish

Per installare l'adattatore di risorse IBM MQ nel server GlassFish su un sistema operativo Windows , è necessario creare e avviare prima un dominio. È quindi possibile distribuire e configurare l'adattatore di risorse e distribuire ed eseguire l'applicazione IVT (Installation Verification Test).

Prima di iniziare

- Queste istruzioni sono per GlassFish Server versione 4.
- Questa versione di GlassFish Server non supporta Jakarta EE.

Informazioni su questa attività

Questa attivit ... presuppone che sia in esecuzione un server delle applicazioni GlassFish Server e che si abbia familiarit ... con le attivit ... di gestione standard. Questa attività presuppone inoltre che si disponga di un'installazione IBM MQ sul sistema locale e che si abbia familiarità con le attività di gestione standard.

Nota: Per completare la seguente procedura di attività, è necessario disporre di un'installazione IBM MQ funzionante, con i seguenti oggetti configurati:

- Un gestore code denominato QM, avviato sulla porta 1414, che utilizza il canale SYSTEM.DEF.SVRCONNe che si connette utilizzando il trasporto client.
- Una coda denominata Q1.

Procedura

1. Avviare il programma shell GlassFish server **asadmin** .
   a) Aprire la riga comandi di Windows e passare alla directory *GlassFish*/bin , dove *GlassFish* è la directory in cui è installato GlassFish Server versione 4.

b) Immettere il comando **asadmin** nella riga comandi.

Il comando **asadmin** apre un programma shell nella riga comandi che consente di creare un nuovo dominio.

GlassFish Il server versione 4 viene avviato sul sistema.

2. Creare e quindi avviare un dominio.

a) Utilizzare il comando **create-domain** , specificando la porta e il nome dominio, per creare un nuovo dominio. Immettere il seguente comando dalla riga comandi:

```
create-domain --adminport port domain_name
```

dove *port* è il numero di porta e *nome_dominio* è il nome che si desidera venga utilizzato dal dominio.

Nota: Al comando **create-domain** sono associati molti parametri facoltativi. Tuttavia, per questa attività è necessario solo il parametro `-- adminport` . Per ulteriori informazioni, consultare la documentazione del prodotto per GlassFish Server versione 4.

Se la porta specificata è in uso, viene visualizzato il seguente messaggio:

`La porta per nome_dominio porta è in uso`

Se il nome dominio specificato è in uso, si riceve un messaggio che indica che il nome specificato è già in uso e un elenco di tutti i nomi dominio attualmente non disponibili.

b) Quando viene richiesto di immettere un nome utente e una password, immettere le credenziali da utilizzare per accedere al server delle applicazioni tramite browser Web.

Se il comando viene completato correttamente, sulla riga comandi viene visualizzato un messaggio che riepiloga la creazione del dominio, incluso il messaggio `Command create-domain executed successfully.`

È stato correttamente creato un dominio.

c) Avviare il proprio dominio immettendo il comando seguente nella riga comandi:

```
start-domain domain_name
```

dove *nome_dominio* è il nome dominio precedentemente specificato.

3. Utilizzare un browser Web per accedere al server delle applicazioni GlassFish .

a) Nella barra degli indirizzi di un browser Web, immettere il comando seguente:

```
localhost:port
```

dove *port* è la porta specificata in precedenza durante la creazione del dominio.

Viene visualizzata la console GlassFish .

b) Quando la console GlassFish è stata caricata e ti vengono richiesti un nome utente e una password, immetti le credenziali che hai specificato nel passo 2b.

4. Caricare l'adattatore di risorse su GlassFish Server 4.

a) Nella barra degli strumenti **Attività comuni** selezionare la voce di menu **Applicazioni** per visualizzare la pagina **Applicazioni** .

b) Fare clic sul pulsante **Distribuisci** per aprire la pagina **Distribuisci applicazioni o moduli** .

c) Fare clic su **Sfoglia** , quindi passare all'ubicazione del file `wmq.jmsra.rar` . Selezionare il file e fare clic su **OK**.

5. Creare un pool di connessioni.

a) Nella barra degli strumenti, in **Risorse**, selezionare la voce di menu **Connettori** .

b) Quindi selezionare la voce di menu **Pool di connessioni connettore** per aprire la pagina **Pool di connessioni connettore** .

c) Fare clic su **Nuovo** per aprire la pagina **Nuovo pool di connessioni del connettore (passo 1 di 2)** .

d) Nella pagina **Nuovo pool di connessioni del connettore (Passo 1 di 2)** , immettere il nome del pool come jms / ivt/IVTCF - Connection - Pool nel campo **Nome pool** .

e) Nel campo **Adattatore di risorse** selezionare `wmq.jmsra`.

f) Nel campo **Definizione connessione** , immettere `javax.jms.ConnectionFactory`.

g) Selezionare **Avanti**, quindi selezionare **Fine**.

6. Creare le risorse del connettore.

a) Nella barra degli strumenti, nel menu **Connettori** , selezionare l'opzione **Risorsa connettore** per aprire la pagina **Risorse connettore** .

b) Selezionare **Nuovo**per aprire la pagina **Nuova risorsa connettore** .

c) Nel campo **Nome JNDI** , immettere `IVTCF`.

d) Nel campo **Nome pool** , immettere `jms/ivt/IVTCF-Connection-Pool`.

e) Lasciare vuoti tutti gli altri campi.

f) Per ogni coppia proprietà / valore, fare clic su **Aggiungi proprietà**e immettere il nome della proprietà e il valore come mostrato nel seguente esempio:

- nome: host; valore: localhost
- nome: porta; valore 1414
- nome: canale; valore: SYSTEM.DEF.SVRCONN
- nome: queueManager; valore: QM
- nome: transportType; valore: CLIENT

Nota: Assicurarsi di utilizzare i valori corretti per le proprie impostazioni di configurazione, che potrebbero essere diverse da quelle mostrate in questo esempio.

g) Nella barra degli strumenti, in **Connettori**, selezionare la voce di menu **Admin Object Resources** per aprire la pagina **Admin Object Resources** .

h) Nella pagina **Risorse oggetto admin** , fare clic su **Nuovo** per aprire la pagina **Nuova risorsa oggetto admin** .

i) Nel campo **Nome JNDI** , immettere `IVTQueue`.

j) Nel campo **Adattatore risorse** , immettere `wmq.jmsra`.

k) Nel campo **Tipo di risorsa** , immettere `javax.jms.Queue`.

l) Lasciare il campo **Nome classe** così com' è.

m) Per ogni coppia proprietà / valore, fare clic su **Aggiungi proprietà**e immettere il nome della proprietà e il valore come mostrato nel seguente esempio:

- nome: nome; valore: IVTQueue
- nome: baseQueueManagerName; valore QM
- nome: baseQueueNome; valore: Q1

Nota: Assicurarsi di utilizzare i valori corretti per le proprie impostazioni di configurazione, che potrebbero essere diverse da quelle mostrate in questo esempio.

n) Fare clic su **OK**.

o) Selezionare la casella di spunta **Abilitato** , quindi fare clic su **Abilita**.

7. Distribuire il file EAR `wmq.jmsra.ivt.ear` nel server GlassFish .

a) Fare clic sull'opzione **Applicazioni** nella barra degli strumenti per visualizzare la pagina **Applicazione** .

b) Fare clic su **Distribuisci** per aggiungere l'applicazione IVT.

c) Nel campo **Ubicazione** , individuare e selezionare `wmq.jmsra.ivt.ear`.

d) Nel campo **Virtual Servers** , selezionare **server**, quindi fare clic su **OK**.

8. Avviare il programma IVT.
   a) Fare clic sull'opzione **Applicazioni** nella barra degli strumenti per visualizzare la pagina **Applicazione** .
   b) Fare clic su `wmq.jmsra.ivt` nella tabella Applicazioni distribuite.
   c) Fare clic sul pulsante **Avvia** nella tabella Moduli e Componenti.
   d) Selezionare il link http:.
   e) Fare clic su **Esegui IVT**.

   Il programma IVT è stato avviato e, se l'operazione ha esito positivo, viene visualizzato il seguente output:

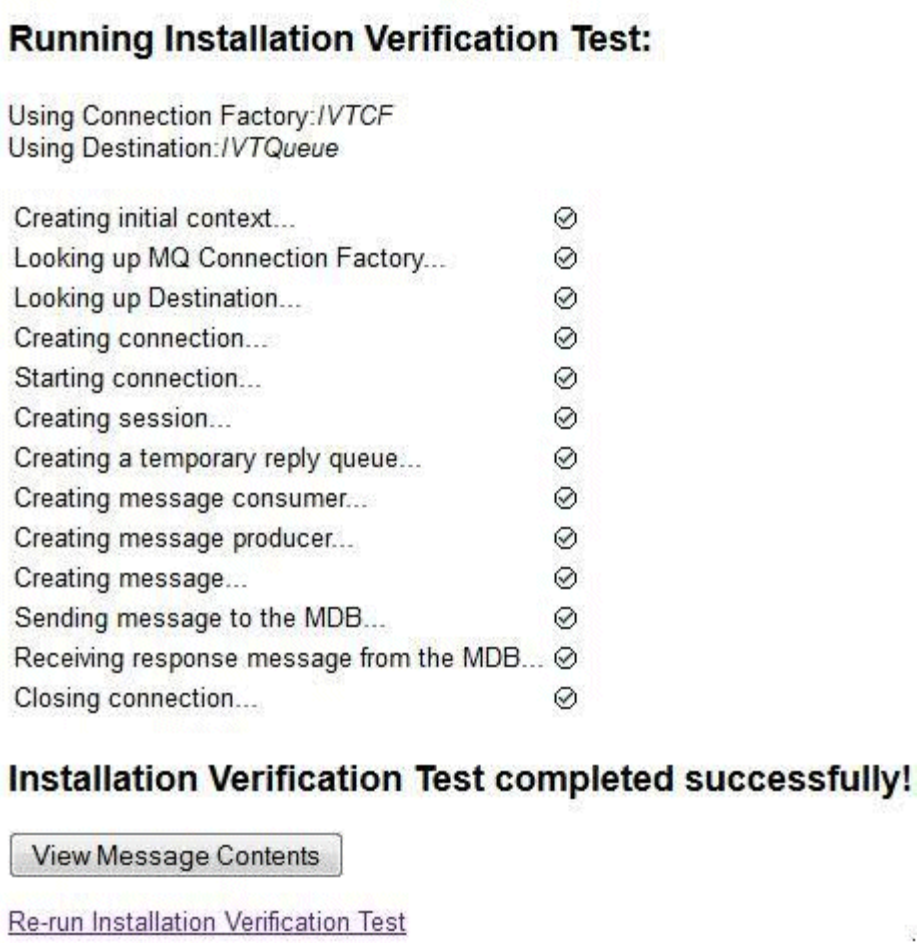


Figura 49. Output IVT riuscito

Installazione e verifica dell'adattatore di risorse in WildFly

Se si sta installando l'adattatore di risorse IBM MQ in WildFly V10, è necessario prima apportare alcune modifiche al file di configurazione per aggiungere una definizione di sottosistema per l'adattatore di risorse IBM MQ . È possibile quindi distribuire l'adattatore risorse e verificarlo installando ed eseguendo l'applicazione IVT (Installation Verification Test).

Prima di iniziare

- Queste istruzioni sono per WildFly V10.
- Questa versione di WildFly non supporta Jakarta EE.

Informazioni su questa attività

Questa attività presuppone che si disponga di un server di applicazioni WildFly in esecuzione e che si abbia familiarità con le attività di gestione standard. Questa attività presuppone inoltre che si disponga di un'installazione IBM MQ e che si abbia familiarità con le attività di gestione standard.

Procedura

1. Creare un gestore code IBM MQ denominato ExampleQMe configurarlo come descritto in "Configurazione di un gestore code per accettare connessioni client su Multiplatforms" a pagina 1072.

   Quando si configura il gestore code, tenere presente quanto segue:

   - Il listener deve essere avviato sulla porta 1414.
   - Il canale da utilizzare è denominato SYSTEM.DEF.SVRCONN.
   - La coda utilizzata dall'applicazione IVT è denominata TEST.QUEUE.

   La coda modello SYSTEM.DEFAULT.MODEL.QUEUE deve essere concessa l'autorizzazione DSP e PUT in modo che questa applicazione possa creare una coda di risposta temporanea.
2. Modificare il file di configurazione *WildFly_Home*/standalone/configuration/standalone-full.xml e aggiungere il seguente sottosistema:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
```

```
                            jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                    <config-property name="baseQueueName">
                        TEST.QUEUE
                    </config-property>
                    </admin-object>
                </admin-objects>
            </resource-adapter>
        </resource-adapters>
    </subsystem>
```

3. Distribuire l'adattatore risorse al server copiando il file `wmq.jmsra.rar` nella directory *WildFly_Home*`/standalone/deployments`.
4. Distribuire l'applicazione IVT copiando il file `wmq.jmsra.ivt.ear` nella directory *WildFly_Home*`/standalone/deployments`.
5. Avviare il server delle applicazioni, visualizzando un prompt dei comandi, passando alla directory *WildFly_Home*`/bin` ed eseguendo il comando:

   ```
   standalone.bat -c standalone-full.xml
   ```

6. Eseguire l'applicazione IVT.

   Per ulteriori informazioni, consultare “Verifica dell'installazione dell'adattatore di risorse” a pagina 492. Per WildFly, l' URL predefinito è `http://localhost:8080/WMQ_IVT/`.

Utilizzo congiunto di IBM MQ e WebSphere Application Server

Tramite il provider di messaggistica IBM MQ in WebSphere Application Server, le applicazioni di messaggistica Java Message Service (JMS) possono utilizzare il proprio sistema IBM MQ come provider esterno di risorse di messaggistica JMS .

Informazioni su questa attività

Le applicazioni scritte in Java e in esecuzione in WebSphere Application Server possono utilizzare la specifica Java Message Service (JMS) per eseguire la messaggistica. La messaggistica in questo ambiente può essere fornita da un gestore code IBM MQ .

Un vantaggio dell'utilizzo di un gestore code IBM MQ è che la connessione delle applicazioni JMS può partecipare pienamente alla funzione di una rete IBM MQ , che consente alle applicazioni di scambiare messaggi con i gestori code in esecuzione su una moltitudine di piattaforme.

Le applicazioni possono utilizzare il *trasporto client* o il *trasporto bind* per l'oggetto factory di connessione code. Per il trasporto dei collegamenti, il gestore code deve esistere localmente per l'applicazione che richiede una connessione.

Per impostazione predefinita, i messaggi JMS contenuti nelle code IBM MQ utilizzano un'intestazione MQRFH2 per contenere alcune delle informazioni di intestazione del messaggio JMS . Molte applicazioni IBM MQ legacy non possono elaborare messaggi con queste intestazioni e richiedono intestazioni proprie, ad esempio MQCIH per CICS Bridge o MQWIH per IBM MQ Workflow. Per ulteriori informazioni su queste considerazioni speciali, consultare Mappatura dei messaggi JMS sui messaggi IBM MQ.

Attività correlate

Configurazione di risorse JMS in WebSphere Application Server

Configurazione del server delle applicazioni per utilizzare il livello di manutenzione dell'adattatore di risorse più recente

Utilizzo di WebSphere Application Server con IBM MQ

IBM MQ e IBM MQ for z/OS possono essere utilizzati con o come alternativa al provider di messaggistica predefinito incluso con WebSphere Application Server.

Il provider di messaggistica IBM MQ è installato come parte di WebSphere Application Server. Ciò include una versione dell'adattatore di risorse IBM MQ e la funzionalità IBM MQ Extended Transactional Client, che consente al gestore code di partecipare alle transazioni XA gestite dal server delle applicazioni.

Utilizzando l'adattatore risorse, i bean basati sui messaggi possono essere configurati per utilizzare le specifiche di attivazione o le porte del listener.

Affinché il server delle applicazioni sia supportato, il programma di test di verifica dell'installazione dell'adattatore di risorse IBMIBM MQ deve essere distribuito nel server delle applicazioni ed eseguito correttamente. Dopo che il programma di verifica dell'installazione dell'adattatore di risorse IBM MQ è stato eseguito correttamente, l'adattatore di risorse IBM MQ può connettersi a qualsiasi gestore code IBM MQ supportato.

JMS connessioni da WebSphere Application Server a IBM MQ

Prima di considerare i livelli di IBM MQ che possono essere utilizzati con WebSphere Application Server, è importante capire in che modo le applicazioni Java Message Service (JMS) in esecuzione nel server delle applicazioni possono connettersi ai gestori code IBM MQ .

Le applicazioni JMS che devono accedere alle risorse di un gestore code IBM MQ possono farlo utilizzando uno dei seguenti tipi di trasporto:

BIND
Questo trasporto può essere utilizzato quando il server delle applicazioni e il gestore code sono installati sulla stessa macchina e sulla stessa immagine del sistema operativo. Quando si utilizza la modalità BINDINGS, tutte le comunicazioni tra i due prodotti vengono eseguite utilizzando IPC (Inter - Process Communication).

Il provider di messaggistica IBM MQ non include le librerie native richieste per la connessione a un gestore code IBM MQ in modalità BINDINGS. Per utilizzare una connessione in modalità BINDINGS, IBM MQ deve essere installato sulla stessa macchina del server delle applicazioni e il percorso della libreria nativa dell'adattatore di risorse deve essere configurato per puntare alla directory IBM MQ in cui si trovano queste librerie. Per ulteriori informazioni, consultare la documentazione del prodotto WebSphere Application Server :

- Per WebSphere Application Server traditional, vedere Configurazione del provider di messaggistica IBM MQ con librerie native.
- Per WebSphere Liberty, consultare Distribuzione delle applicazioni JMS in Liberty per utilizzare il provider di messaggistica IBM MQ.

z/OS Su z/OS, se si desidera connettere un factory di connessione WebSphere Application Server ad un gestore code IBM MQ in modalità bind, è necessario specificare le librerie IBM MQ corrette nella concatenazione STEPLIB WebSphere Application Server . Per ulteriori informazioni, consultare le librerie IBM MQ e WebSphere Application Server per z/OS STEPLIB nella documentazione del prodotto WebSphere Application Server .

CLIENTE
Il trasporto client utilizza TCP/IP per comunicare tra WebSphere Application Server e IBM MQ. Oltre ad essere utilizzato quando il server di applicazioni e il gestore code si trovano su macchine differenti, la modalità CLIENT può essere utilizzata anche quando i due prodotti sono installati nella stessa macchina e immagine del sistema operativo.

Le applicazioni JMS possono specificare anche un tipo di trasporto BINDINGS_THEN_CLIENT. Quando viene utilizzato questo tipo di trasporto, l'applicazione inizialmente tenterà di connettersi al gestore code utilizzando la modalità BINDINGS - se non è in grado di farlo, tenterà il trasporto CLIENT.

Come individuare la versione dell'adattatore di risorse IBM MQ installato all'interno di WebSphere Application Server

Per informazioni su quale versione dell'adattatore di risorse IBM MQ è installata all'interno di WebSphere Application Server, consultare la nota tecnica Quale versione di WebSphere MQ Resource Adapter (RA) viene fornita con WebSphere Application Server?.

È possibile utilizzare i seguenti comandi Jython e JACL per determinare il livello dell'adattatore di risorse attualmente utilizzato da WebSphere Application Server :

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

Nota: È necessario fare doppio clic su **Ritorna** dopo aver immesso questo comando per eseguirlo.

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

Aggiornamento dell'adattatore di risorse

Gli aggiornamenti all'adattatore di risorse IBM MQ installato con il server delle applicazioni sono inclusi nei Fix Pack WebSphere Application Server . Aggiornamento dell'adattatore di risorse IBM MQ utilizzando **Aggiorna adattatore di risorse ...** nella console di gestione WebSphere Application Server non è consigliata, poiché ciò significa che gli aggiornamenti forniti in WebSphere Application Server Fix Pack non avranno alcun effetto.

variabile MQ_INSTALL_ROOT

Da WebSphere Application Server 7.0, MQ_INSTALL_ROOT viene utilizzato solo per individuare le librerie native e viene sovrascritto da qualsiasi percorso di libreria nativa configurato sull'adattatore di risorse.

Connessione da WebSphere Application Server a IBM MQ

Attenzione:

1. Qualsiasi versione supportata di WebSphere Application Server può utilizzare l'adattatore di risorse IBM MQ fornito con esso, per connettersi a qualsiasi versione supportata di IBM MQ.
2. Se viene utilizzata la modalità di bind, alcune librerie in WebSphere Application Server devono corrispondere alla versione del gestore code a cui si sta connettendo:
   - WebSphere Application Server deve essere configurato per caricare le librerie native fornite con IBM MQ 9.3. Per ulteriori informazioni, consulta “Configurazione delle librerie JNI (Java Native Interface)” a pagina 98.
   - z/OS Su z/OS, è necessario specificare le librerie IBM MQ corrette nella concatenazione STEPLIB WebSphere Application Server .

     Consultare IBM MQ libraries e WebSphere Application Server for z/OS STEPLIB per informazioni dettagliate sulle librerie IBM MQ necessarie.

     Se si dispone di librerie per una versione di IBM MQ in LINKLIST (LINKLST), è possibile collegarsi a una versione differente di IBM MQ sovrascrivendo le librerie con STEPLIB.
3. La versione di IBM MQ Resource Adapter è indipendente dalle versioni della libreria nativa (condivisa) fornite dall'installazione del gestore code.

   Ad esempio, WebSphere Application Server 8.5, con un adattatore di risorse IBM MQ 8.0 , può ancora gestire una connessione di bind a un gestore code IBM MQ 9.0 utilizzando le librerie native IBM MQ 9.0 .

Per ulteriori informazioni, consultare “Istruzione di supporto dell'adattatore di risorse IBM MQ” a pagina 439.

I tipi di trasporto BINDINGS e CLIENT possono essere utilizzati per connettersi a IBM MQ da qualsiasi versione di WebSphere Application Server. Per il tipo di trasporto BINDINGS, si applicano le seguenti limitazioni:

- IBM MQ deve essere installato sulla stessa macchina del server di applicazioni.
- WebSphere Application Server deve essere configurato per caricare le librerie native fornite con IBM MQ.
- z/OS Su z/OS, se si desidera connettere una factory di connessione WebSphere Application Server a un gestore code IBM MQ in modalità bind, è necessario specificare le librerie IBM MQ corrette nella concatenazione WebSphere Application Server STEPLIB.

La seguente tabella mostra le versioni di WebSphere Application Server in cui è supportata l'esecuzione di ciascuna versione dell'adattatore di risorse IBM MQ .

Tabella 70. Associazione delle versioni WebSphere Application Server alle versioni dell'adattatore di risorse IBM MQ .

Versione dell'adattatore di risorse IBM MQ	**In quale versione di WebSphere Application Server può essere eseguita questa versione dell'adattatore risorse?**
IBM MQ 9.0 e versioni successive	L'adattatore risorse può essere eseguito in: • Qualsiasi versione conforme a Java EE 7 di WebSphere Liberty. • WebSphere Application Server traditional 9.0
IBM MQ 8.0	L'adattatore di risorse può essere eseguito in qualsiasi versione compatibile con Java EE 7 di WebSphere Liberty L'adattatore di risorse IBM MQ 8.0 non è supportato per l'esecuzione in WebSphere Application Server traditional. L'adattatore di risorse già installato in WebSphere Application Server traditional deve essere utilizzato per connettersi ai gestori code IBM MQ 8.0 .

Concetti correlati

“Istruzione di supporto dell'adattatore di risorse IBM MQ” a pagina 439
L'adattatore risorse IBM MQ che è necessario utilizzare per la comunicazione tra un'applicazione e un gestore code dipende dal fatto che si stia utilizzando l'API Jakarta Messaging 3.0 o l'API JMS 2.0 .

Informazioni correlate

Requisiti di sistema per IBM MQ

Determinazione del numero di connessioni TCP/IP create da WebSphere Application Server a IBM MQ

Utilizzando la funzione di condivisione delle conversazioni, più conversazioni possono condividere istanze di canale MQI, questa è anche nota come connessione TCP/IP.

Informazioni su questa attività

Le applicazioni in esecuzione in WebSphere Application Server 7 e WebSphere Application Server 8, che utilizzano la modalità normale del provider di messaggistica IBM MQ , utilizzeranno automaticamente questa funzione. Ciò significa che più applicazioni in esecuzione all'interno della stessa istanza del server delle applicazioni, che si connettono allo stesso gestore code IBM MQ , possono condividere la stessa istanza del canale.

Il numero di conversazioni che possono essere condivise tra una singola istanza del canale è determinato dalla IBM MQ proprietà del canale **SHARECNV**. Il valore predefinito di questa proprietà per i canali di connessione server è 10.

Esaminando il numero di conversazioni create da WebSphere Application Server 7 e WebSphere Application Server 8, puoi stabilire il numero delle istanze del canale che vengono create.

Per ulteriori informazioni sulla modalità del provider di messaggistica IBM MQ , consultare Modalità normale PROVIDERVERSION.

Concetti correlati

Utilizzo delle conversazioni di condivisione
In un ambiente in cui la condivisione delle conversazioni è consentita, le conversazioni possono condividere un'istanza del canale MQI.

“Condivisione di una connessione TCP/IP in IBM MQ classes for JMS” a pagina 318
È possibile creare più istanze di un canale MQI per condividere una connessione TCP/IP singola.

Factory di connessione JMS

Le applicazioni in esecuzione all'interno di WebSphere Application Server, che utilizzano una factory di connessione del fornitore di messaggistica IBM MQ per creare connessioni e sessioni, hanno conversazioni attive per ogni connessione JMS creata dalla factory di connessione e per ogni sessione JMS creata dalla connessione JMS .

Una conversazione per ogni connessione JMS creata dalla factory di connessione

Ogni factory di connessioni di JMS ha un pool di connessioni associato, diviso in due sezioni, il lotto libero e il lotto attivo. Entrambi i lotti sono inizialmente vuoti.

Quando un'applicazione crea una connessione JMS da una factory di connessione, WebSphere Application Server verifica se esiste una connessione JMS nel lotto libero. In caso affermativo, viene spostato nel pool attivo e assegnato all'applicazione. Altrimenti, viene creata una nuova connessione JMS , inserita nel pool attivo e restituita all'applicazione. Il numero massimo di connessioni che è possibile creare da un factory di connessioni è specificato dalla proprietà del pool di connessioni del factory di connessione **Maximum connections**. Il valore predefinito per questa proprietà è 10.

Dopo che un'applicazione ha terminato una connessione JMS e l'ha chiusa, la connessione viene spostata dal pool attivo al pool libero, dove è disponibile per il riutilizzo. La proprietà del pool di connessioni **Unused timeout** definisce per quanto tempo una connessione JMS può rimanere nel pool libero prima di essere disconnessa. Il valore predefinito per questa proprietà è 1800 secondi (30 minuti).

Quando viene creata una prima connessione JMS , inizia una conversazione tra WebSphere Application Server e IBM MQ . La conversazione rimane attiva fino a quando la connessione non viene chiusa quando viene superato il valore della proprietà **Unused timeout** per il lotto libero.

Una conversazione per ogni sessione JMS creata da una connessione JMS

Ogni connessione JMS creata da una factory di connessione del provider dei messaggi IBM MQ dispone di un pool di sessioni JMS associato. I pool di sessioni funzionano come i pool di connessioni. Il numero massimo di JMS sessioni che è possibile creare da una singola connessione JMS è determinato dalla proprietà del pool di sessioni factory di connessioni **Maximum connections**. Il valore predefinito di questa proprietà è 10.

Una conversazione inizia quando una sessione JMS viene creata per la prima volta. La conversazione rimane attiva finché la sessione JMS non viene chiusa perché è rimasta nel lotto libero per un periodo di tempo superiore al valore della proprietà **Unused timeout** per il lotto sessioni.

Calcolo di un valore per la proprietà SHARECNV

Puoi calcolare il numero massimo di conversazioni da una singola factory di connessione a IBM MQ utilizzando la formula seguente:

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Il numero di istanze del canale che verranno create per consentire questo numero di conversazioni può essere calcolato utilizzando il seguente calcolo:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Tutto il resto di questo calcolo può essere arrotondato per eccesso.

Per un factory di connessione semplice che utilizza il valore predefinito per le proprietà del pool di connessioni **Maximum connections** e del pool di sessioni **Maximum connections** , il numero massimo di conversazioni che possono esistere tra WebSphere Application Server e IBM MQ per questo factory di connessione è:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Ad esempio:

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

Se questo factory di connessione si connette a IBM MQ utilizzando un canale che ha la proprietà **SHARECNV** impostata su 10, il numero massimo di istanze del canale che verranno create per questo factory di connessione è:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

Ad esempio:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

Specifiche di attivazione

Le applicazioni MDB (Message - Driven Bean), che sono configurate per utilizzare una specifica di attivazione, hanno conversazioni attive per la specifica di attivazione per monitorare una destinazione JMS e per ogni sessione server utilizzata per eseguire un'istanza MDB (Message - Driven Bean) per elaborare i messaggi.

Le seguenti conversazioni sono attive per le applicazioni bean basate sui messaggi configurate per utilizzare una specifica di attivazione:

- Una conversazione per la specifica di attivazione per monitorare una destinazione JMS per messaggi adatti. Questa conversazione viene avviata appena viene avviata la specifica di attivazione e rimane attiva fino all'arresto della specifica di attivazione.
- Una conversazione per ogni sessione server utilizzata per eseguire un'istanza del bean basato sui messaggi per elaborare i messaggi.

La proprietà avanzata della specifica di attivazione **Maximum server sessions** specifica il numero massimo di sessioni server che possono essere attive contemporaneamente per una determinata specifica di attivazione. Questa proprietà ha il valore predefinito 10. Le sessioni server vengono create quando sono necessarie e vengono chiuse se sono state inattive per il periodo di tempo specificato dalla proprietà avanzata della specifica di attivazione **Server session pool timeout**. Il valore predefinito per questa proprietà è 300000 millisecondi (5 minuti).

Le conversazioni iniziano quando viene creata una sessione server e vengono arrestate quando la specifica di attivazione viene arrestata o quando una sessione server scade.

Ciò significa che il numero massimo di conversazioni da una singola specifica di attivazione a IBM MQ può essere calcolato utilizzando la formula seguente:

```
Maximum number of conversations = Maximum server sessions + 1
```

Il numero di istanze del canale create per consentire questo numero di conversazioni può essere trovato cantando il seguente calcolo:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Tutto il resto di questo calcolo può essere arrotondato per eccesso.

Per una semplice specifica di attivazione, che utilizza il valore predefinito per la proprietà **Maximum server sessions** , il numero massimo di conversazioni che possono esistere tra WebSphere Application Server e IBM MQ per questa specifica di attivazione viene calcolato come:

```
Maximum number of conversations = Maximum server sessions + 1
```

Ad esempio:

```
    = 10 + 1
    = 11
```

Se questa specifica di attivazione si connette a IBM MQ utilizzando un canale che ha la proprietà **SHARECNV** impostata su 10, il numero di istanze del canale create viene calcolato come segue:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Ad esempio:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Porte listener in esecuzione in modalità ASF (Application Server Facilities)

Le porte listener in esecuzione in modalità ASF utilizzate dalle applicazioni bean basate sui messaggi creano conversazioni per ogni sessione server. Uno controlla una destinazione per i messaggi appropriati e un altro esegue un'istanza del bean basato sui messaggi per elaborare i messaggi. Il numero di conversazioni per ogni porta del listener può essere calcolato da un massimo di sessioni.

Per impostazione predefinita, le porte listener verranno eseguite in modalit ... ASF come parte della specifica 1.1 che definisce il meccanismo che i server delle applicazioni devono utilizzare per rilevare i messaggi e consegnarli ai bean basati sui messaggi per l'elaborazione. Le applicazioni MDB (Message - Driven Bean) impostate per utilizzare le porte listener in questa modalità predefinita di operazione creano conversazioni:

Una conversazione per la porta del listener per monitorare una destinazione per i messaggi appropriati

Le porte listener sono configurate per utilizzare un factory di connessione JMS . Quando viene avviata una porta listener, viene effettuata una richiesta per una connessione JMS dal pool libero della factory di connessione. La connessione viene restituita al lotto libero quando la porta listener viene arrestata. Per ulteriori informazioni sul modo in cui viene utilizzato il pool di connessione e sul modo in cui ciò influisce sul numero di conversazioni in IBM MQ, consultare "Factory di connessione JMS" a pagina 504.

Una conversazione per ogni sessione del server utilizzata per eseguire un'istanza del bean basato sui messaggi per elaborare i messaggi

La proprietà della porta listener **Maximum sessions** specifica il numero massimo di sessioni server che possono essere attive contemporaneamente per una determinata porta listener. Questa proprietà ha il valore predefinito 10. Le sessioni server vengono create in base alle necessità e utilizzano le sessioni JMS prese dal pool di sessioni associato alla connessione JMS utilizzata dalla porta listener.

Se una sessione server è stata inattiva per il periodo di tempo specificato dalla proprietà personalizzata Servizio listener dei messaggi **SERVER.SESSION.POOL.UNUSED.TIMEOUT**, la sessione viene chiusa e la sessione JMS utilizzata viene restituita al pool libero del pool di sessioni. La sessione JMS rimarrà nel pool libero del pool di sessione fino a quando non è necessaria oppure viene chiusa perché è stata inattiva nel pool libero per un periodo di tempo superiore al valore della proprietà **Unused timeout** del pool di sessione.

Per ulteriori informazioni su come viene utilizzato il pool di sessioni e su come vengono gestite le conversazioni tra WebSphere Application Server e IBM MQ , consultare "Factory di connessione JMS" a pagina 504.

Per ulteriori informazioni sulla proprietà personalizzata del servizio listener dei messaggi **SERVER.SESSION.POOL.UNUSED.TIMEOUT**, consultare Monitoring server session pools for listener ports nella documentazione del prodotto WebSphere Application Server .

Calcolo del numero massimo di conversazioni da una singola porta listener a IBM MQ

È possibile calcolare il numero massimo di conversazioni da una singola porta listener a IBM MQ utilizzando la formula seguente:

```
Maximum number of conversations = Maximum sessions + 1
```

Il numero di istanze del canale che verranno create per consentire questo numero di conversazioni può essere calcolato utilizzando il seguente calcolo:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Tutto il resto di questo calcolo può essere arrotondato per eccesso.

Per una porta listener semplice che utilizza il valore predefinito per la proprietà **Maximum sessions** , il numero massimo di conversazioni che possono esistere tra WebSphere Application Server e IBM MQ per questa porta listener viene calcolato come:

```
Maximum number of conversations = Maximum sessions + 1
```

Ad esempio:

```
    = 10 + 1
    = 11
```

Se questa porta listener si connette a IBM MQ utilizzando un canale con la proprietà **SHARECNV** impostata su 10, il numero di istanze del canale che verranno create viene calcolato come segue:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Ad esempio:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Porte listener in esecuzione in modalità non ASF (non Application Server Facilities)

Le porte listener in esecuzione in modalità non ASF possono essere configurate per monitorare la destinazione della coda e la destinazione dell'argomento utilizzando le sessioni server. Le sessioni server possono avere più conversazioni, il cui numero massimo può essere calcolato in ogni caso.

Le porte listener possono essere configurate per essere eseguite in modalità non - ASF, che modifica il modo in cui le porte listener monitorano le destinazioni JMS . Le applicazioni MDB (Message - Driven Bean), che utilizzano le porte listener in modalità operativa non ASF, creano una conversazione per ogni sessione server utilizzata per eseguire un'istanza MDB (Message - Driven Bean) per elaborare i messaggi. La proprietà della porta listener **numero massimo di sessioni** specifica il numero massimo di sessioni server che possono essere attive contemporaneamente per una determinata porta listener. Il valore predefinito per questa proprietà è 10.

Durante l'esecuzione in modalità non - ASF, una porta listener che controlla una destinazione coda creerà automaticamente il numero di sessioni server specificato dalla proprietà della porta listener **Numero massimo di sessioni**. Tutte queste sessioni server utilizzano le sessioni JMS prese dal pool di sessioni associato alla connessione JMS che la porta listener sta utilizzando e monitorano continuamente una destinazione JMS per i messaggi adatti.

Se la porta listener è configurata per monitorare una destinazione argomento, il valore di **Numero massimo di sessioni** viene ignorato e viene utilizzata una singola sessione server.

Le sessioni server utilizzate da una porta listener in esecuzione in modalità non ASF restano attive fino a quando la porta listener non viene arrestata, a questo punto le sessioni JMS utilizzate vengono restituite al pool libero di sessioni per la connessione JMS utilizzata dalla porta listener.

Per ulteriori informazioni su come viene utilizzato il pool di sessioni e su come vengono gestite le conversazioni tra WebSphere Application Server e IBM MQ , consultare “Factory di connessione JMS” a pagina 504.

Per ulteriori informazioni sulla modalità operativa ASF e non ASF con WebSphere Application Servere su come configurare le porte listener per utilizzare la modalità non ASF, consultare Elaborazione dei messaggi in modalità ASF e non ASF.

Calcolo del numero massimo di conversazioni durante il monitoraggio di una destinazione coda

Il numero massimo di conversazioni da una singola porta listener, in esecuzione in modalità non ASF e monitorando una destinazione della coda in IBM MQ , può essere calcolato utilizzando la formula seguente:

```
Maximum number of conversations = Maximum sessions
```

Il numero di istanze del canale che verranno create per consentire l'esecuzione di questo numero di conversazioni può essere trovato utilizzando il seguente calcolo:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Tutto il resto di questo calcolo può essere arrotondato per eccesso.

Per una porta listener semplice in esecuzione in modalità non ASF che utilizza il valore predefinito per la proprietà **Numero massimo di sessioni** e per il monitoraggio di una destinazione coda, il numero

massimo di conversazioni che possono esistere tra WebSphere Application Server e IBM MQ per questa porta listener è:

```
Maximum number of conversations = Maximum sessions
```

Ad esempio:

```
     = 10
```

Se questa porta listener si connette a IBM MQ utilizzando un canale con la proprietà **SHARECNV** impostata su 10, il numero di istanze del canale create viene calcolato come segue:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Ad esempio:

```
     = 10 / 10
     = 1
```

Calcolo del numero massimo di conversazioni durante il monitoraggio di una destinazione argomento

Per una porta listener in esecuzione in modalità non ASF e configurata per monitorare una destinazione argomenti, il numero di conversazioni dalla porta listener a IBM MQ è:

```
 Maximum number of conversations = 1
```

Il numero di istanze del canale che verranno create per consentire l'esecuzione di questo numero di conversazioni può essere trovato utilizzando il seguente calcolo:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Tutto il resto di questo calcolo può essere arrotondato per eccesso.

Per una porta listener semplice in esecuzione in modalità non - ASF che utilizza il valore predefinito per la proprietà **Numero massimo di sessioni** e per il monitoraggio di una destinazione argomenti, il numero massimo di conversazioni che possono esistere tra WebSphere Application Server e IBM MQ per questa porta listener è:

```
Maximum number of conversations = Maximum sessions
```

Ad esempio:

```
     = 10
```

Se questa porta listener si connette a IBM MQ utilizzando un canale con la proprietà **SHARECNV** impostata su 10, il numero di istanze del canale create viene calcolato come segue:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Ad esempio:

```
= 10 / 10
= 1
```

Configurazione degli alias di autenticazione per proteggere la connessione WebSphere Application Server a IBM MQ

Gli alias di autenticazione sono associati a una combinazione di nome utente e password che può essere utilizzata per proteggere la connessione WebSphere Application Server a IBM MQ. È possibile configurare una factory di connessione con un alias di autenticazione.

Utilizzo degli alias di autenticazione con le applicazioni enterprise

Quando un'applicazione enterprise in esecuzione all'interno di WebSphere Application Server tenta di creare una connessione JMS a IBM MQ, l'applicazione cerca una definizione di factory di connessione del provider di messaggistica IBM MQ dal repository Java Naming Directory Interface (JNDI) del server delle applicazioni.

Quando la definizione della factory di connessione del fornitore di messaggistica IBM MQ si trova dall'interno del repository JNDI del server delle applicazioni, viene richiamato uno dei seguenti metodi:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Se il factory di connessione è stato configurato con un alias di autenticazione J2C definito, il nome utente e la password nell'alias di autenticazione possono essere trasferiti a IBM MQ quando il factory di connessione viene utilizzato per creare una connessione.

Factory di connessione e alias di autenticazione

I factory di connessione del provider dei messaggi IBM MQ contengono informazioni su come connettersi ai gestori code IBM MQ . Le applicazioni enterprise in esecuzione all'interno di WebSphere Application Server possono utilizzare le factory di connessione per creare JMS connessioni a IBM MQ.

WebSphere Application Server memorizza le definizioni di factory di connessione in un repository a cui è possibile accedere utilizzando JNDI. Quando viene creato un factory di connessione, al factory di connessione viene fornito un nome JNDI per identificarlo in modo univoco nell'ambito del server delle applicazioni (l'ambito Cella, Nodo o Server) in cui è stato definito.

Ad esempio, una factory di connessione del provider dei messaggi IBM MQ definita nell'ambito cella WebSphere Application Server contiene informazioni su come connettersi al gestore code (`myQM`) utilizzando il trasporto BINDINGS. A questa produzione connessioni viene assegnato il JNDI nome `jms/myCF` per identificarla in modo univoco.

I factory di connessione possono essere configurati anche per utilizzare un alias di autenticazione. Gli alias di autenticazione sono associati a una combinazione di nome utente e password. A seconda del modo in cui viene utilizzato il factory di connessione, il nome utente e la password nell'alias di autenticazione potrebbero o meno essere trasferiti a IBM MQ quando viene creata la connessione JMS .

Importante: Prima di IBM MQ 8.0, l'OAM (Object Authority Manager) IBM MQ predefinito eseguiva un controllo di autorizzazione, solo per garantire che il nome utente trasmesso a IBM MQ, quando viene effettuata una connessione, avesse l'autorità per accedere al gestore code.

Non è stato effettuato alcun controllo per convalidare la parola d'ordine specificata. Per eseguire un controllo di autenticazione e convalidare la corrispondenza tra l'identificativo utente e la password, è necessario scrivere un'uscita di sicurezza del canale IBM MQ . I dettagli su come eseguire questa operazione sono disponibili in “Programmi di uscita di sicurezza del canale” a pagina 971.

Da IBM MQ 8.0, il gestore code controlla la password oltre al nome utente.

Utilizzo del factory di connessione

I seguenti argomenti contengono informazioni sull'utilizzo della factory di connessione mediante ricerche dirette e indirette:

- “Utilizzo della factory di connessione tramite una ricerca diretta” a pagina 514
- “Utilizzo del factory di connessione tramite una ricerca indiretta” a pagina 515

Utilizzo del trasporto CLIENT

I factory di connessione configurati per utilizzare il trasporto CLIENT devono specificare quale canale di connessione server IBM MQ (SVRCONN) utilizzerà per connettersi al gestore code.

Se la proprietà MCAUSER (IBM MQ channel agent user identifier) rimane vuota per il canale che il factory di connessione è stato configurato per utilizzare, il factory di connessione può essere utilizzato con una ricerca diretta o indiretta.

Se la proprietà MCAUSER è impostata su un identificativo utente, questo identificativo utente viene trasmesso a IBM MQ quando il factory di connessione viene utilizzato per creare una connessione a IBM MQ, indipendentemente dal fatto che l'applicazione enterprise stia utilizzando una ricerca diretta o indiretta.

Tabelle di riepilogo

Le seguenti tabelle riepilogano gli identificativi utente che vengono trasferiti a IBM MQ quando vengono utilizzati rispettivamente il trasporto BINDINGS e il trasporto CLIENT:

Tabella 71. modalità BINDINGS

Configurazione	**L'applicazione richiama `ConnectionFactory.createConnection()`**	**L'applicazione richiama `ConnectionFactory.createConnection(String username, String password)`**
Il descrittore di distribuzione dell'applicazione non contiene un riferimento risorsa per il factory di connessione	L'identificativo utente per il processo del server delle applicazioni viene confluito in IBM MQ.	L'identificativo utente e la password passati nel metodo `ConnectionFactory.createConnection(String username, String password)` vengono trasmessi a IBM MQ.
Il descrittore di distribuzione dell'applicazione contiene un riferimento risorsa per la factory di connessione e la proprietà **`res-auth`** è impostata su "Applicazione"	L'identificativo utente per il processo del server delle applicazioni viene confluito in IBM MQ.	L'identificativo utente e la password passati nel metodo `ConnectionFactory.createConnection(String username, String password)` vengono trasmessi a IBM MQ.
Il descrittore di distribuzione dell'applicazione contiene un riferimento risorsa per il factory di connessione e la proprietà **`res-auth`** è impostata su "Contenitore"	L'identificativo utente e password specificati nell'alias di autenticazione per il factory di connessioni vengono trasferiti a IBM MQ.	L'identificativo utente e password specificati nell'alias di autenticazione per il factory di connessioni vengono trasferiti a IBM MQ.

Tabella 71. modalità BINDINGS (Continua)

Configurazione	**L'applicazione richiama `ConnectionFactory.createConnection()`**	**L'applicazione richiama `ConnectionFactory.createConnection(String username, String password)`**
Il descrittore di distribuzione dell'applicazione contiene un riferimento risorsa per la factory di connessione che ha la proprietà **res-auth** impostata su "Contenitore" e l'applicazione è stata configurata con un alias di autenticazione	L'identificativo utente e la password specificati nell'alias di autenticazione che l'applicazione è stata configurata per utilizzare vengono trasferiti a IBM MQ.	L'identificativo utente e la password specificati nell'alias di autenticazione che l'applicazione è stata configurata per utilizzare vengono trasferiti a IBM MQ.

Tabella 72. Modalità CLIENT

Configurazione	**L'applicazione richiama `ConnectionFactory.createConnection()`**	**L'applicazione richiama `ConnectionFactory.createConnection(String username, String password)`**
Il descrittore di distribuzione dell'applicazione non contiene un riferimento risorsa per la factory di connessione e la factory di connessione è configurata per utilizzare un canale IBM MQ con la proprietà MCAUSER non impostata	L'identificativo utente per il processo del server delle applicazioni viene confluito in IBM MQ.	L'identificativo utente e la password passati nel metodo `ConnectionFactory.createConnection(String username, String password)` vengono trasmessi a IBM MQ.
Il descrittore di distribuzione dell'applicazione non contiene un riferimento di risorsa per il factory di connessione e il factory di connessione è configurato per utilizzare un canale IBM MQ con la proprietà MCAUSER impostata su un identificativo utente	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per l'utilizzo viene trasmesso a IBM MQ.	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per l'utilizzo viene trasmesso a IBM MQ.
Il descrittore di distribuzione dell'applicazione contiene un riferimento di risorsa per il factory di connessione con la proprietà **res-auth** impostata su *Applicazione* e il factory di connessione è configurato per utilizzare un canale IBM MQ con la proprietà MCAUSER non impostata	L'identificativo utente per il processo del server delle applicazioni viene confluito in IBM MQ.	L'identificativo utente e la password passati nel metodo `ConnectionFactory.createConnection(String username, String password)` vengono trasmessi a IBM MQ.

Tabella 72. Modalità CLIENT (Continua)

Configurazione	L'applicazione richiama `ConnectionFactory.createConnection()`	L'applicazione richiama `ConnectionFactory.createConnection(String username, String password)`
Il descrittore di distribuzione dell'applicazione contiene un riferimento risorsa per il factory di connessione con la proprietà **res-auth** impostata su *Applicazione* e il factory di connessione è configurato per utilizzare un canale IBM MQ con la proprietà MCAUSER impostata su un identificativo utente	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per utilizzare viene trasmesso a IBM MQ.	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per utilizzare viene trasmesso a IBM MQ.
Il descrittore di distribuzione dell'applicazione contiene un riferimento alla risorsa per il factory di connessione che ha la proprietà **res-auth** impostata su "*Contenitore* e il factory di connessione è configurato per utilizzare un canale IBM MQ con la proprietà MCAUSER non impostata	L'identificativo utente e password specificati nell'alias di autenticazione per il factory di connessioni vengono trasferiti a IBM MQ.	L'identificativo utente e password specificati nell'alias di autenticazione per il factory di connessioni vengono trasferiti a IBM MQ.
Il descrittore distribuzione dell'applicazione contiene un riferimento risorsa per il factory di connessione che ha la proprietà **res-auth** impostata su "*Contenitore* e il factory di connessione è configurato per utilizzare un canale IBM MQ che ha la proprietà MCAUSER impostata su un identificativo utente	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per utilizzare viene trasmesso a IBM MQ.	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per utilizzare viene trasmesso a IBM MQ.
Il descrittore di distribuzione dell'applicazione contiene un riferimento di risorsa per il factory di connessione che ha la proprietà **res-auth** impostata su "*Contenitore* e l'applicazione è stata configurata con un alias di autenticazione e il factory di connessione è configurato per utilizzare un canale IBM MQ che ha la proprietà MCAUSER non impostata	L'identificativo utente e la password specificati nell'alias di autenticazione che l'applicazione è stata configurata per utilizzare vengono trasferiti a IBM MQ.	L'identificativo utente e la password specificati nell'alias di autenticazione che l'applicazione è stata configurata per utilizzare vengono trasferiti a IBM MQ.

Tabella 72. Modalità CLIENT (Continua)		
Configurazione	**L'applicazione richiama `ConnectionFactory.createConnection()`**	**L'applicazione richiama `ConnectionFactory.createConnection(String username, String password)`**
Il descrittore di distribuzione dell'applicazione contiene un riferimento di risorsa per il factory di connessione che ha la proprietà **`res-auth`** impostata su *Contenitore* e l'applicazione è stata configurata con un alias di autenticazione e il factory di connessione è configurato per utilizzare un canale IBM MQ che ha MCAUSER impostato su un identificativo utente	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per utilizzare viene trasmesso a IBM MQ.	L'identificativo utente specificato dalla proprietà MCAUSER sul canale IBM MQ che il factory di connessione è configurato per utilizzare viene trasmesso a IBM MQ.

Utilizzo della factory di connessione tramite una ricerca diretta

Una volta definita una factory di connessione del fornitore di messaggistica IBM MQ , un'applicazione enterprise può ricercare la definizione della factory di connessione e utilizzarla per creare una JMS connessione a un gestore code IBM MQ . Questo può essere fatto attraverso una ricerca diretta.

Per utilizzare una ricerca diretta, un'applicazione enterprise si connette al repository JNDI del server delle applicazioni, effettuando la seguente chiamata al metodo:

```
InitialContext ctx = new InitialContext();
```

Una volta connesso al repository JNDI, l'applicazione enterprise identifica la definizione del factory di connessione utilizzando il nome JNDI del factory di connessione, come segue:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

Note:

- Lo sviluppatore dell'applicazione deve conoscere il nome JNDI della factory di connessione richiesta quando l'applicazione enterprise viene sviluppata. Poiché il nome JNDI è codificato all'interno dell'applicazione, se il nome JNDI cambia, è necessario riscrivere e ridistribuire l'applicazione.
- Quando una definizione del factory di connessione viene utilizzata in questo modo, il nome utente e la parola d'ordine specificati nell'alias di autenticazione (che il factory di connessione è stato configurato per utilizzare) non vengono trasferiti a IBM MQ. Ciò impedisce alle applicazioni non autorizzate di identificare il factory di connessione e di utilizzarlo per connettersi a sistemi IBM MQ sicuri.

Il nome utente e la password che vengono trasferiti a IBM MQ dipendono dal metodo utilizzato per creare la connessione JMS dalla factory di connessione.

Se un'applicazione crea una connessione JMS utilizzando il metodo:

```
ConnectionFactory.createConnection()
```

l'identità utente predefinita viene trasmessa a IBM MQ. Si tratta del nome utente e della password che hanno avviato il server delle applicazioni su cui è in esecuzione l'applicazione enterprise.

In alternativa, un'applicazione può creare una connessione JMS richiamando il metodo:

```
ConnectionFactory.createConnection(String username, String password)
```

Se un'applicazione ha eseguito una ricerca diretta di un factory di connessione e quindi ha richiamato questo metodo, il nome utente e la password passati nel metodo `createConnection()` vengono trasmessi a IBM MQ.

Importante: Prima di IBM MQ 8.0, IBM MQ aveva elaborato un controllo di autorizzazione, solo per assicurarsi che il nome utente che era stato passato, avesse l'autorizzazione ad accedere al gestore code.

Non è stato effettuato alcun controllo sulla password. Per eseguire un controllo di autenticazione e verificare che il nome utente e la password siano validi, è necessario scrivere un'uscita di sicurezza del canale IBM MQ . I dettagli su come eseguire questa operazione sono disponibili in “Programmi di uscita di sicurezza del canale” a pagina 971.

Da IBM MQ 8.0, il gestore code controlla la password oltre al nome utente.

Utilizzo del factory di connessione tramite una ricerca indiretta

Quando si scrive un'applicazione enterprise, se il nome JNDI della factory di connessione è sconosciuto o se l'applicazione deve essere installata su server delle applicazioni differenti utilizzando una factory di connessione differente, con un nome JNDI differente (in base al server delle applicazioni su cui è installato), è possibile cercare la factory di connessione utilizzando un riferimento di risorsa. Ciò può essere fatto attraverso una ricerca indiretta.

Esempio

Invece di cercare direttamente la factory di connessione utilizzando `jms/myCF`, un'applicazione enterprise contiene un riferimento di risorsa con il nome JNDI locale: `jms/myResourceReferenceCF`.

Per utilizzare questo nome JNDI , l'applicazione si connette al repository JNDI del server delle applicazioni, come se l'applicazione stesse eseguendo una ricerca diretta:

```
InitialContext ctx = new InitialContext();
```

Invece di identificare direttamente `jms/myCF` , l'applicazione identifica ora il JNDI nome del riferimento della risorsa:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

È necessario il prefisso `java:comp/env` per il nome JNDI locale, per indicare al server delle applicazioni che l'applicazione enterprise sta eseguendo una ricerca indiretta.

Quando l'applicazione viene distribuita, l'utente associa il JNDI nome del riferimento risorsa `jms/myResourceReferenceCF` al nome JNDI del factory di connessione che l'applicazione ha già creato: `jms/myCF`.

Quando l'applicazione viene eseguita, ricerca un factory di connessione di JMS utilizzando il nome JNDI locale, associato al server delle applicazioni: `jms/myCF`. Questo factory di connessione viene quindi utilizzato dall'applicazione per creare una connessione a IBM MQ.

Alias di autenticazione e ricerche indirette

Un riferimento di risorsa consente inoltre di definire ulteriori proprietà che modificano il funzionamento della factory di connessione fornita. Una delle proprietà di un riferimento risorsa è **`res-auth`**. Il valore di questa proprietà specifica se l'applicazione enterprise deve utilizzare l'alias di autenticazione del factory di connessione associato al riferimento della risorsa durante la creazione di una connessione a IBM MQ (se è stato definito un alias di autenticazione) o se l'applicazione sta specificando il proprio nome utente e la propria password.

Il valore predefinito di questa proprietà è *Applicazione*. Ciò significa che il nome utente e la password trasferiti al gestore code, quando viene creata una connessione JMS , sono determinati dall'applicazione stessa. L'alias di autenticazione della factory di connessione a cui è associato il riferimento risorsa non viene utilizzato.

Le applicazioni possono creare connessioni JMS utilizzando uno dei seguenti metodi:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Se un'applicazione utilizza `ConnectionFactory.createConnection()`, e **res-auth** è impostato su *Applicazione*, l'identità utente predefinita viene riprodotta in IBM MQ. Si tratta del nome utente e della password che hanno avviato il server delle applicazioni su cui è in esecuzione l'applicazione enterprise.

Se un'applicazione utilizza `ConnectionFactory.createConnection(String username, String password)` e **res-auth** è impostato su *Applicazione*, il nome utente e la parola d'ordine trasmessi al metodo vengono inviati a IBM MQ.

Per utilizzare l'alias di autenticazione definito sul factory di connessione a cui è associato il riferimento risorsa durante la creazione di una connessione, è necessario impostare la proprietà **res-auth** sul valore *Contenitore*. Quando un'applicazione crea una connessione JMS , vengono utilizzati i dettagli dell'alias di autenticazione, anche se la chiamata `createConnection` specifica un nome utente e una password.

Sovrascrittura dell'alias di autenticazione quando si utilizza una ricerca indiretta

Se un'applicazione utilizza un riferimento risorsa con la proprietà **res-auth** impostata su *Contenitore*, è possibile sovrascrivere l'alias di autenticazione utilizzato quando vengono create le connessioni JMS .

Per sovrascrivere l'alias di autenticazione, il riferimento della risorsa deve includere una proprietà supplementare denominata **authDataAlias**, che si associa a un alias di autenticazione esistente già creato nell'ambiente del server delle applicazioni in cui verrà distribuita l'applicazione. È possibile specificare questa proprietà su qualsiasi riferimento di risorsa creato utilizzando la strumentazione Rational fornita da IBM.

Utilizzando questo metodo, è possibile utilizzare un alias di autenticazione diverso quando si utilizza un factory di connessione JMS che è stato ricercato indirettamente. Se l'alias di autenticazione specificato non esiste, è possibile specificarne uno nuovo dopo l'installazione dell'applicazione enterprise. Per ulteriori informazioni, consultare *Riferimenti alle risorse* nella documentazione del prodotto WebSphere Application Server .

Informazioni correlate per WebSphere Application Server 8.5.5

Riferimenti di risorse

Informazioni correlate per WebSphere Application Server 8.0

Riferimenti di risorse

Informazioni correlate per WebSphere Application Server 7.0

Riferimenti di risorse

Bilanciamento del carico di lavoro per i bean basati sui messaggi quando si utilizzano i cluster WebSphere Application Server

Quando si utilizzano applicazioni MDB (message driven bean) distribuite in cluster WebSphere Application Server 7.0 e WebSphere Application Server 8.0 e configurate per l'esecuzione in modalità normale del provider di messaggistica IBM MQ , uno dei membri del cluster elabora la maggior parte dei messaggi. È possibile bilanciare il workload dei membri del cluster per distribuire l'elaborazione dei messaggi su più di un membro del cluster.

IBM MQ include una funzione denominata **Asynchronous consume**, che consente alle applicazioni di utilizzare i messaggi in modo asincrono da una coda utilizzando le API denominate **MQCB** e **MQCTL**.

Le applicazioni MDB (message driven bean) in esecuzione all'interno di WebSphere Application Server 7.0 e WebSphere Application Server 8.0, che utilizzano la modalità normale del provider di

messaggistica IBM MQ utilizzeranno automaticamente questa funzione. Quando le applicazioni vengono avviate, imposteranno un consumer asincrono sulla destinazione JMS che sono state configurate per il monitoraggio richiamando **MQCB**. L'API **MQCTL** viene quindi richiamata per indicare che l'applicazione è pronta a ricevere messaggi dalla destinazione JMS .

Quando le applicazioni MDB (message - driven bean) sono state distribuite in un cluster WebSphere Application Server , ciascun membro del cluster imposterà un consumer asincrono per la destinazione JMS che il MDB (message - driven bean) sta monitorando per i messaggi. Il gestore code IBM MQ che ospita la destinazione JMS è quindi responsabile della notifica al membro del cluster quando è presente un messaggio appropriato sulla destinazione JMS per l'elaborazione.

Quando WebSphere Application Server si connette a un gestore code IBM MQ , i messaggi che arrivano su una destinazione JMS verranno distribuiti in modo più uniforme a tutti i consumer asincroni registrati su tale destinazione JMS . Per le applicazioni bean basate sui messaggi distribuite all'interno di un cluster WebSphere Application Server 7.0 e WebSphere Application Server 8.0 , ciò significa che i messaggi verranno distribuiti in modo più uniforme tra i membri del cluster.

Attività correlate

Configurazione della proprietà JMS **PROVIDERVERSION**

Utilizzo del pacchetto IBM MQ Headers

Il pacchetto di intestazioni IBM MQ fornisce una serie di classi e interfacce helper che è possibile utilizzare per manipolare le intestazione IBM MQ di un messaggio. In genere, si utilizza il pacchetto Intestazioni IBM MQ perché si desidera eseguire i servizi di gestione utilizzando il server dei comandi (utilizzando i messaggi PCF (Programmable Command Format)).

Informazioni su questa attività

Il pacchetto IBM MQ Headers si trova nei pacchetti `com.ibm.mq.headers` e `com.ibm.mq.headers.pcf` . È possibile utilizzare questa funzione per entrambe le due API alternative fornite da IBM MQ per l'utilizzo nelle applicazioni Java :

- IBM MQ classes for Java (indicato anche come IBM MQ Base Java).
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, indicato anche come IBM MQ JMS).

Le applicazioni IBM MQ Base Java in genere manipolano gli oggetti MQMessage e le classi di supporto delle intestazioni possono interagire direttamente con tali oggetti, poiché comprendono nativamente le interfacce IBM MQ Base Java .

In IBM MQ JMS, il payload per un messaggio è in genere una stringa o un oggetto array di byte, che può essere manipolato con i flussi DataInput e DataOutput . Il pacchetto di intestazioni IBM MQ può essere utilizzato per interagire con questi flussi di dati ed è adatto per gestire qualsiasi messaggio MQ inviato e ricevuto dalle applicazioni IBM MQ JMS .

Pertanto, sebbene il package IBM MQ Headers contenga riferimenti al package IBM MQ Base Java , è anche destinato all'utilizzo all'interno delle applicazioni IBM MQ JMS ed è adatto per l'utilizzo in ambienti Java Platform, Enterprise Edition (Java EE).

Un modo tipico in cui è possibile utilizzare il pacchetto IBM MQ Headers consiste nel manipolare i messaggi di gestione in PCF (Programmable Command Format), ad esempio per uno dei seguenti motivi:

- Per accedere ai dettagli su una risorsa IBM MQ .
- Per monitorare la profondità di una coda.
- Per impedire l'accesso a una coda.

Utilizzando i messaggi PCF con IBM MQ JMS API, questo tipo di gestione delle risorse incentrate sull'applicazione può essere eseguito dall'interno delle applicazioni Java EE senza dover ricorrere all'API IBM MQ Base Java .

Procedura

- Per utilizzare il pacchetto IBM MQ Headers per manipolare le intestazioni dei messaggi per IBM MQ classes for Java, consultare “Utilizzo con IBM MQ classes for Java” a pagina 518.
- Per utilizzare il pacchetto IBM MQ Headers per manipolare le intestazioni dei messaggi per IBM MQ classes for JMS, consultare “Utilizzo con IBM MQ classes for JMS” a pagina 518.

Utilizzo con IBM MQ classes for Java

Le applicazioni IBM MQ classes for Java di solito manipolano gli oggetti MQMessage e le classi di supporto delle intestazioni possono interagire direttamente con tali oggetti, poiché comprendono nativamente le interfacce IBM MQ classes for Java .

Informazioni su questa attività

IBM MQ fornisce alcune applicazioni di esempio che dimostrano come utilizzare il pacchetto IBM MQ Headers con l'API IBM MQ Base Java (IBM MQ classes for Java).

Gli esempi mostrano due cose:

- Come creare un messaggio PCF per eseguire un'azione di gestione ed analizzare il messaggio di risposta.
- Come inviare questo messaggio PCF utilizzando IBM MQ classes for Java.

In base alla piattaforma utilizzata, questi esempi vengono installati nella directory `pcf` nella directory `samples` o `tools` dell'installazione di IBM MQ (consultare “Directory di installazione per IBM MQ classes for Java” a pagina 357).

Procedura

1. Creare un messaggio PCF per eseguire un'azione di gestione e analizzare il messaggio di risposta.
2. Inviare questo messaggio PCF utilizzando IBM MQ classes for Java.

Concetti correlati

“Gestione delle intestazioni dei messaggi IBM MQ con IBM MQ classes for Java” a pagina 385
Vengono fornite classi Java che rappresentano diversi tipi di intestazione del messaggio. Sono fornite anche due classi helper.

“Gestione dei messaggi PCF con IBM MQ classes for Java” a pagina 390
Le classi Java vengono fornite per creare e analizzare messaggi strutturati PCF e per facilitare l'invio di richieste PCF e la raccolta di risposte PCF.

Utilizzo con IBM MQ classes for JMS

Per utilizzare le intestazioni IBM MQ con IBM MQ classes for JMS, è necessario eseguire le stesse operazioni essenziali di IBM MQ classes for Java. Il messaggio PCF può essere creato e la risposta analizzata esattamente nello stesso modo utilizzando il pacchetto di intestazioni IBM MQ e lo stesso codice di esempio di IBM MQ classes for Java.

Informazioni su questa attività

Per inviare un messaggio PCF utilizzando l'API IBM MQ , il payload del messaggio deve essere scritto in un messaggio di byte JMS e inviato utilizzando le API JMS standard. L'unica considerazione è che il messaggio non deve contenere una JMS RFH2 o altre intestazioni con valori specifici in MQMD.

Per inviare un messaggio PCF, completare le seguenti operazioni. Il modo in cui viene creato il messaggio PCF e le informazioni vengono estratte dal messaggio di risposta è lo stesso di IBM MQ classes for Java (consultare “Utilizzo con IBM MQ classes for Java” a pagina 518).

Procedura

1. Creare una destinazione coda JMS che rappresenti il SISTEMA SYSTEM.ADMIN.COMMAND.QUEUE.

   Le applicazioni IBM MQ JMS inviano i messaggi PCF al SISTEMA SYSTEM.ADMIN.COMMAND.QUEUE, e deve accedere a un oggetto di destinazione JMS che rappresenta questa coda. La destinazione deve avere le seguenti proprietà impostati:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Se si utilizza WebSphere Application Server, è necessario definire queste proprietà come proprietà personalizzate sulla destinazione.

   Per creare la destinazione in modo programmatico dall'interno di un'applicazione, utilizzare il codice seguente:

   ```
   Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
   ((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
       WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQQueue) q1).setMQMDWriteEnabled(true);
   ```

2. Convertire un messaggio PCF in un messaggio JMS Bytes contenente i valori MQMD corretti.

   È necessario creare un messaggio di byte JMS e scrivervi il messaggio PCF. È necessario creare una coda di risposta, ma questa non deve avere impostazioni specifiche.

   Il seguente frammento di codice di esempio mostra come creare un JMS Bytes Message e scrivervi un oggetto com.ibm.mq.headers,pcf.PCFMessage . L'oggetto PCFMessage (pcfCmd) è stato precedentemente creato utilizzando il package IBM MQ Headers. (Si noti che il pacchetto per caricare PCFMessage è com.ibm.mq.headers.pcf.PCFMessage).

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();

   // Create the wrapping streams to put the bytes into the message payload
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   DataOutput dataOutput = new DataOutputStream(baos);

   // Set the JMSReplyTo so the answer comes back
   msg.setJMSReplyTo(new MQQueue("adminResp"));

   // write the pcf into the stream
   pcfCmd.write(dataOutput);
   baos.flush();
   msg.writeBytes(baos.toByteArray());

   // we have taken control of the MD, so need to set all
   // flags in the MD that we require - main one is the format
   msg.setJMSPriority(4);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
     CMQC.MQPER_NOT_PERSISTENT);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
       CMQC.MQRO_PASS_CORREL_ID);
   msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

   // and send the message
   sender.send(msg);
   ```

3. Invia il messaggio e ricevi la risposta utilizzando le API JMS standard.
4. Convertire il messaggio di risposta in un messaggio PCF per l'elaborazione.

   Per richiamare il messaggio di risposta ed elaborarlo come messaggio PCF, utilizzare il codice seguente:

   ```
    // Get the message back
   BytesMessage msg = (BytesMessage) consumer.receive();

   // get the size of the bytes message & read into an array
   int bodySize = (int) msg.getBodyLength();
   byte[] data = new byte[bodySize];
   msg.readBytes(data);
   ```

```
// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

Concetti correlati

“JMS messaggi” a pagina 146
I messaggi JMS sono composti da un'intestazione, proprietà e un corpo. JMS definisce cinque tipi di corpo del messaggio.

IBM i Configurazione di IBM MQ su IBM i con Java e JMS

Questa raccolta di argomenti fornisce una panoramica su come impostare e verificare IBM MQ con Java e JMS su IBM i utilizzando i comandi CL o l'ambiente qshell.

Nota:

- Da IBM MQ 8.0, `ldap.jar`, `jndi.jar` e `jta.jar` fanno parte di JDK.
- V 9.3.0 JM 3.0 V 9.3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 continua a supportare JMS 2.0 per applicazioni esistenti. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, consultare Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging.

Utilizzo dei comandi CL

Il CLASSPATH impostato è per la verifica con MQ base Java, JMS con JNDI e JMS senza JNDI.

Se non si utilizza un file `.profile` nella directory `/home/Userprofile` , sarà necessario impostare le seguenti variabili di ambiente a livello di sistema. È possibile verificare se sono impostati utilizzando il comando **WRKENVVAR** .

1. Per visualizzare le variabili di ambiente per l'intero sistema immettere il comando **WRKENVVAR LEVEL (*SYS)**
2. Per visualizzare le variabili di ambiente specifiche per il lavoro, immettere il comando: **WRKENVVAR LEVEL (*JOB)**
3. Se CLASSPATH non è impostato, immettere il seguente comando:

   JM 3.0

   ```
   ADDENVVAR ENVVAR(CLASSPATH)
          VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
          :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
          :/QIBM/ProdData/mqm/java/samples/base
          :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
          :/QIBM/ProdData/mqm/java/lib/jms.jar
          :/QIBM/ProdData/mqm/java/lib/providerutil.jar
          :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
   ```

   JMS 2.0

   ```
   ADDENVVAR ENVVAR(CLASSPATH)
          VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
          :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
          :/QIBM/ProdData/mqm/java/samples/base
          :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
          :/QIBM/ProdData/mqm/java/lib/jms.jar
          :/QIBM/ProdData/mqm/java/lib/providerutil.jar
          :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
   ```

4. Se QIBM_MULTI_THREADED non è impostato, immettere il seguente comando:

   ```
   ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
   ```

5. Se QIBM_USE_DESCRIPTOR_STDIO non è impostato, immettere il seguente comando:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Se QSH_REDIRECTION_TEXTDATA non è impostato, immettere il seguente comando:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

Utilizzo dell'ambiente qshell

Se si utilizza l'ambiente QSHELL, è possibile impostare un `.profile` nella directory `/home/Userprofile` . Per ulteriori informazioni fare riferimento alla documentazione Qshell Interpreter (qsh).

Specificare quanto segue in `.profile`. Si noti che l'istruzione CLASSPATH deve essere su una singola riga o separata su righe differenti utilizzando il carattere \ come mostrato.

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

Verificare che la libreria QMQMJAVA si trovi nell'elenco librerie immettendo il comando **DSPLIBL**.

Se la libreria QMQMJAVA non è presente nell'elenco, aggiungerla utilizzando il seguente comando: **ADDLIBLE LIB (QMQMJAVA)**

IBM i Test di IBM MQ su IBM i con Java

Come verificare IBM MQ con Java utilizzando il programma di esempio MQIVP.

Verifica di IBM MQ base Java

Effettuare la seguente procedura:

1. Verificare che il gestore code sia avviato e che lo stato del gestore code sia ATTIVO, emettendo il seguente comando:

```
WRKMQM MQMNAME(QMGRNAME)
```

2. Verificare che JAVA.CHANNEL è stato creato immettendo il seguente comando:

```
WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

   a. Se JAVA.CHANNEL non esiste, immettere il seguente comando:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

3. Verificare che il listener del gestore code sia in esecuzione per la porta 1414 o qualsiasi porta si stia utilizzando, immettendo il comando **WRKMQMLSR** .

   a. Se non è stato avviato alcun listener per il gestore code, immettere il seguente comando:

```
STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
```

Esecuzione del programma di verifica di esempio MQIVP

1. Avviare qshell dalla riga comandi immettendo il comando STRQSH
2. Verificare che sia impostato il CLASSPATH corretto emettendo il comando **export** , quindi immettere il comando **cd** come riportato di seguito:

```
cd /qibm/proddata/mqm/java/samples/wmqjava/samples
```

3. Eseguire il programma **java** immettendo il seguente comando:

```
java MQIVP
```

È possibile premere il tasto INVIO quando viene richiesto:

- Tipo di connessione
- Indirizzo IP
- Nome del gestore code

per utilizzare i valori predefiniti. Ciò verifica i bind del prodotto, che possono essere trovati nella libreria QMQMJAVA.

Si riceve un output simile al seguente esempio. Si noti che la dichiarazione di copyright dipende dalla versione del prodotto che si sta utilizzando.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

Test della connessione client IBM MQ Java

È necessario specificare:

- Tipo di connessione
- Indirizzo IP
- PORT
- Canale di connessione server
- Gestore code

Si riceve un output simile al seguente esempio. Si noti che la dichiarazione di copyright dipende dalla versione del prodotto che si sta utilizzando.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i Test di IBM MQ su IBM i con JMS

Modalità di test di IBM MQ con JMS con e senza JNDI

Verifica di JMS senza JNDI utilizzando l'esempio IVTRun

Effettuare la seguente procedura:

1. Verificare che il gestore code sia avviato e che lo stato del gestore code sia ATTIVO, emettendo il seguente comando:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Avviare qshell, dalla riga comandi, immettendo il comando **STRQSH** .
3. Utilizzare il comando **cd** per modificare la directory nel modo seguente:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

4. Eseguire il file di script:

   ```
   IVTRun -nojndi [-m qmgrname]
   ```

Si riceve un output simile al seguente esempio. Si noti che le dichiarazioni di copyright dipendono dalle versioni dei prodotti che si stanno utilizzando:

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
```

```
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

Verifica della modalità client IBM MQ JMS senza JNDI

Effettuare la seguente procedura:

1. Verificare che il gestore code sia avviato e che lo stato del gestore code sia ATTIVO, emettendo il seguente comando:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Verificare che il canale di connessione del server sia stato creato, immettendo il seguente comando:

   ```
   WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
   MQMNAME(QMGRNAME)
   ```

3. Verificare che il listener sia avviato per la porta corretta, immettendo il comando **WRKMQMLSR**
4. Avviare qshell, dalla riga comandi, immettendo il comando **STRQSH** .
5. Verificare che CLASSPATH sia corretto immettendo il comando **export** .
6. Utilizzare il comando **cd** per modificare la directory nel modo seguente:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

7. Eseguire il file di script:

   ```
   IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
   ```

Si riceve un output simile al seguente esempio. Si noti che le dichiarazioni di copyright dipendono dalle versioni dei prodotti che si stanno utilizzando.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Verifica di IBM MQ JMS con JNDI

Verificare che il gestore code sia avviato e che lo stato del gestore code sia ATTIVO, emettendo il seguente comando:

```
WRKMQM MQMNAME(QMGRNAME)
```

Utilizzo dello script di test di esempio IVTRun

Effettuare la seguente procedura:

1. Apportare le modifiche appropriate al file `JMSAdmin.config` . Per modificare questo file utilizzare il comando **EDTF** (Modifica file) da una riga comandi IBM i

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. Per utilizzare LDAP per Weblogic, rimuovere il commento da:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
```

b. Per utilizzare LDAP per WebSphere Application Server, rimuovere il commento da:

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
```

c. Per verificare il file system, rimuovere il commento da:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

d. Assicurarsi di aver selezionato PROVIDER_URL corretto, rimuovendo il commento dalla riga appropriata.

e. Impostare come commento tutte le altre linee utilizzando il simbolo # .

f. Una volta completate tutte le modifiche, premere **F2=Save** e **F3=Exit**.

2. Avviare qshell, dalla riga comandi, immettendo il comando **STRQSH** .
3. Verificare che CLASSPATH sia corretto immettendo il comando **export** .
4. Utilizzare il comando **cd** per modificare la directory nel modo seguente:

```
cd /qibm/proddata/mqm/java/bin
```

5. Avviare lo script **IVTSetup** per creare gli oggetti gestiti (*MQQueueConnectionFactory* e *MQQueue*), immettendo il comando **IVTSetup** .
6. Eseguire lo script IVTRun immettendo il seguente comando:

```
IVTRun -url providerURL [-icf initCtxFact]
```

Si riceve un output simile al seguente esempio. Si noti che le dichiarazioni di copyright dipendono dalle versioni dei prodotti che si stanno utilizzando.

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again
```

```
Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Sviluppo di applicazioni Java utilizzando un repository Maven

Quando sviluppi un'applicazione Java per IBM MQ, utilizzando un repository Maven per installare automaticamente le dipendenze, non devi installare esplicitamente nulla prima di utilizzare le interfacce IBM MQ .

Repository centrale Maven

Maven è uno strumento per la creazione di applicazioni e fornisce anche un repository per contenere le risorse utente a cui la tua applicazione potrebbe voler accedere.

Il repository Maven (o Central Repository) ha una struttura che consente ai file come i file JAR di avere versioni distinte che vengono poi facilmente rilevate con un meccanismo di denominazione noto. Gli strumenti di build possono quindi utilizzare tali nomi per eseguire dinamicamente il pull delle dipendenze per la propria applicazione. Nella definizione della tua applicazione, che, quando utilizzi Maven come strumento di build, viene chiamato file POM, denomini le dipendenze e il processo di build sappia cosa fare da lì.

IBM MQ file client

Le copie delle interfacce client IBM MQ Java sono disponibili in Central Repository in `com.ibm.mq` `GroupId`. È possibile trovare il file `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) e il file `com.ibm.mq.allclient.jar` (JMS 2.0). Questi file vengono generalmente utilizzati per programmi autonomi. È inoltre possibile trovare il file `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) e il file `wmq.jmsra.rar` (JMS 2.0), da utilizzare nei server delle applicazioni Java EE). `jakarta.client.jar` e `allclient.jar` contengono entrambi IBM MQ classes for JMS e IBM MQ classes for Java.

Importante: L'utilizzo del formato Apache Maven Assembly Plugin *jar - with - dependencies* per creare un'applicazione che include il file JAR rilocabile IBM MQ non è supportato.

In un file `pom.xml` elaborato dal comando maven, aggiungi le dipendenze per tali file JAR come mostrato nei seguenti esempi:

- V 9.3.0 JM 3.0 V 9.3.0 Per mostrare la relazione tra il tuo codice dell'applicazione e `com.ibm.mq.jakarta.client.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
```

```
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 Per mostrare la relazione tra il tuo codice dell'applicazione e `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- V 9.3.0 JM 3.0 V 9.3.0 Per utilizzare l'adattatore di risorse Jakarta EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 Per utilizzare l'adattatore di risorse JMS 2.0 Java EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Per un esempio di un progetto semplice in Eclipse per eseguire un progetto JMS , consulta l'articolo IBM Developer Developing Java applications for MQ appena diventato più semplice con Maven.

Sviluppo di applicazioni C + +

IBM MQ fornisce classi C+ + equivalenti a oggetti IBM MQ e alcune classi aggiuntive equivalenti a tipi di dati array. Fornisce un certo numero di funzioni non disponibili tramite MQI.

IBM WebSphere MQ 7.0, i miglioramenti alle interfacce di programmazione IBM MQ non vengono applicati alle classi C + +.

IBM MQ C++ fornisce le seguenti funzioni:

- Inizializzazione automatica delle strutture dati IBM MQ .
- Connessione e apertura della coda del gestore code just-in-time.
- Chiusura implicita della coda e disconnessione del gestore code.
- Ricezione e trasmissione dell'intestazione della lettera non recapitata.
- Trasmissione e ricezione dell'intestazione bridge IMS .
- Trasmissione e ricezione dell'intestazione del messaggio di riferimento.
- Attivare la ricevuta del messaggio.
- Trasmissione e ricezione dell'intestazione CICS bridge .
- Trasmissione e ricezione intestazione lavoro.
- Definizione di canale client.

I seguenti diagrammi di classe Booch mostrano che tutte le classi sono ampiamente parallele alle entità IBM MQ nell'MQI procedurale (ad esempio utilizzando C) che hanno handle o strutture di dati. Tutte le classi ereditano dalla classe ImqError (consultare ImqError C++ class), che consente di associare una condizione di errore a ciascun oggetto.

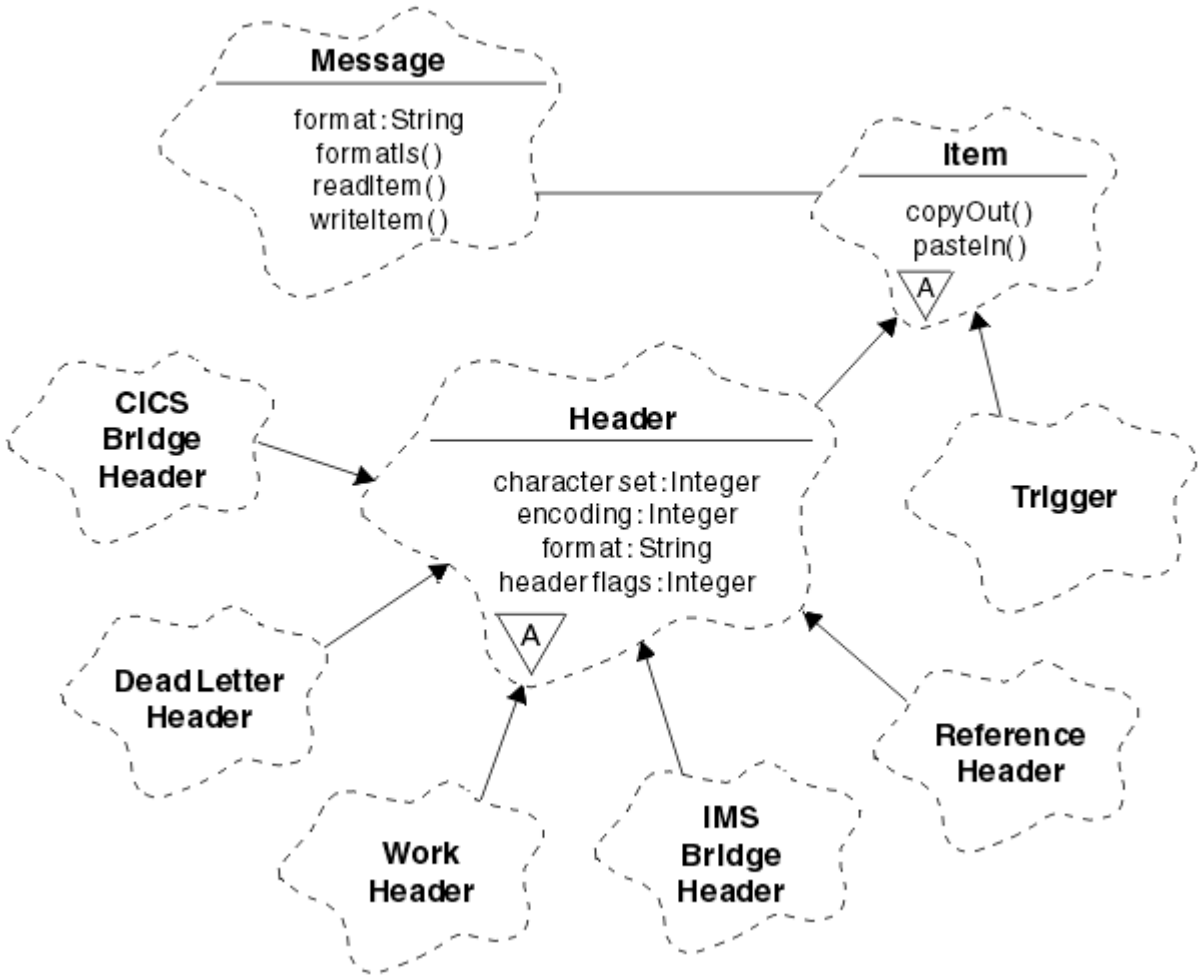


Figura 50. Classi C++ IBM MQ (gestione elementi)

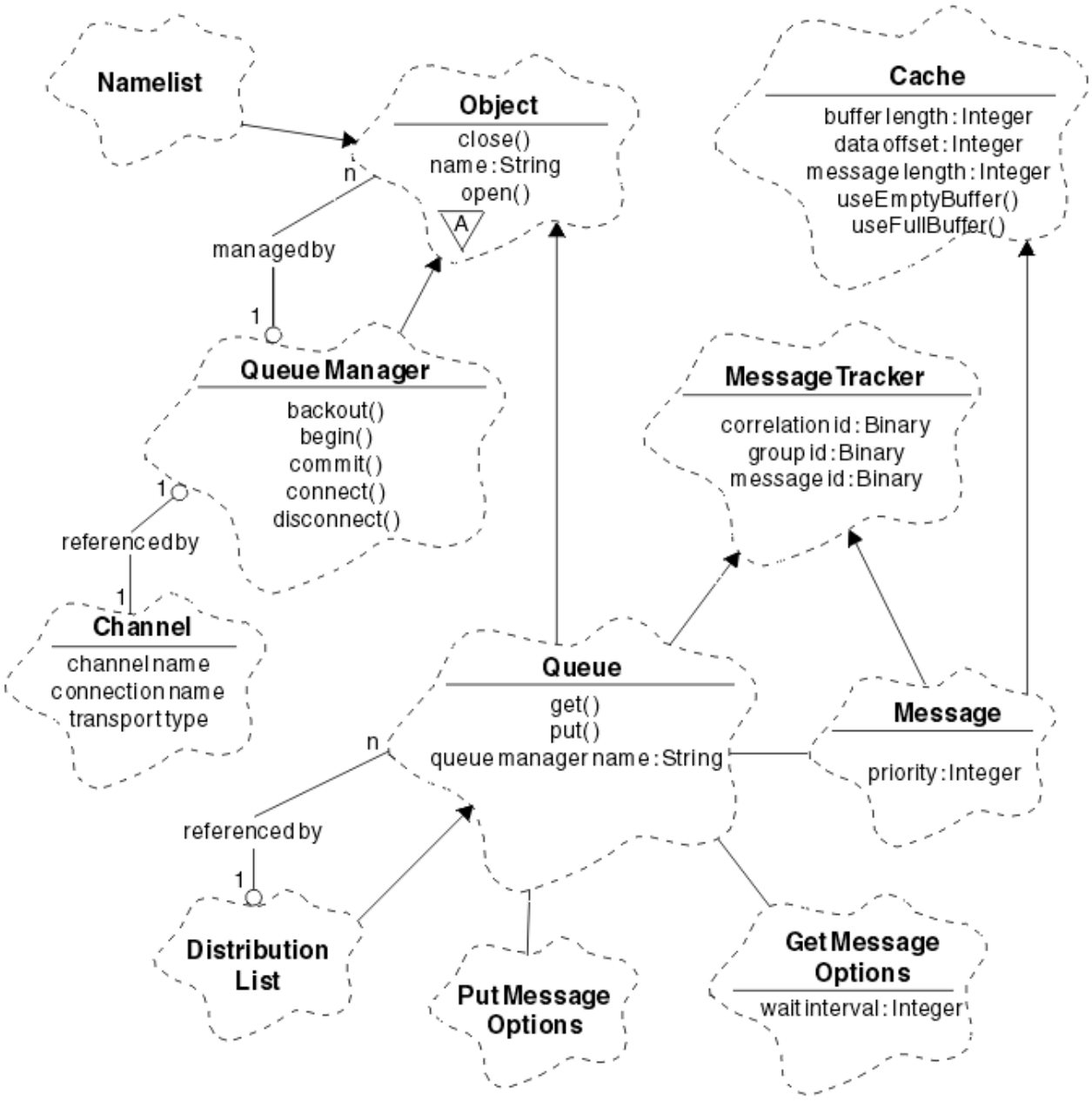


Figura 51. Classi IBM MQ C+ + (gestione code)

Per interpretare correttamente i diagrammi di classe Booch, tenere presente le seguenti convenzioni:

- I metodi e gli attributi degni di nota sono riportati sotto il nome della *classe* .
- Un piccolo triangolo all'interno di un cloud denota una *classe astratta*.
- L' *eredità* è indicata da una freccia alla classe parent.
- Una linea non decorata tra i cloud denota una *relazione di cooperazione* tra le classi.
- Una riga decorata con un numero indica una *relazione referenziale* tra due classi. Il numero indica il numero di oggetti che possono partecipare a una particolare relazione in qualsiasi momento.

Le seguenti classi e tipi di dati vengono utilizzati nelle firme del metodo C + + delle classi di gestione code (consultare Figura 51 a pagina 530) e le classi di gestione elementi (consultare Figura 50 a pagina 529):

- La classe ImqBinary (consultare ImqBinary C++ class), che incapsula gli array di byte come MQBYTE24.
- Il tipo di dati ImqBoolean , definito come **typedef unsigned char ImqBoolean**.

- La classe ImqString (consultare ImqString classe C++), che incapsula gli array di caratteri come MQCHAR64.

Le entità con strutture di dati vengono utilizzate all'interno delle classi di oggetti appropriate. Singoli campi della struttura dati (consultare Riferimento incrociato C++ e MQI) si accede con i metodi.

Le entità con handle rientrano nella gerarchia di classi ImqObject (consultare ImqObject Classe C++) e fornire interfacce incapsulate all'MQI. Gli oggetti di queste classi presentano un comportamento intelligente che può ridurre il numero di richiami del metodo richiesti rispetto all'MQI procedurale. Ad esempio, è possibile stabilire ed eliminare le connessioni del gestore code in base alle esigenze oppure è possibile aprire una coda con le opzioni appropriate, quindi chiuderla.

La classe ImqMessage (consultare ImqMessage Classe C + +) incapsulare la struttura dei dati MQMD e funge anche da punto di attesa per i dati utente e *elementi* (consultare “Lettura dei messaggi in C++” a pagina 540) fornendo funzioni di buffer memorizzate nella cache. È possibile fornire buffer a lunghezza fissa per i dati utente e utilizzare il buffer molte volte. La quantità di dati presenti nel buffer può variare da un utilizzo all'altro. In alternativa, il sistema può fornire e gestire un buffer di lunghezza flessibile. Sia la dimensione del buffer (la quantità disponibile per la ricezione dei messaggi) che la quantità effettivamente utilizzata (o il numero di byte per la trasmissione o il numero di byte effettivamente ricevuti) diventano considerazioni importanti.

Concetti correlati

Panoramica tecnica

“Programmi di esempio C++” a pagina 531
Vengono forniti quattro programmi di esempio, per dimostrare la ricezione e l'inserimento di messaggi.

“Considerazioni sul linguaggio C++” a pagina 535
Questa raccolta di argomenti descrive in dettaglio gli aspetti dell'utilizzo del linguaggio C++ e le convenzioni che è necessario considerare quando si scrivono programmi applicativi che utilizzano MQI (Message Queue Interface).

“Preparazione dei dati dei messaggi in C + +” a pagina 539
I dati del messaggio vengono preparati in un buffer, che può essere fornito dal sistema o dall'applicazione. Ci sono vantaggi in entrambi i metodi. Vengono forniti esempi di utilizzo di un buffer.

“Sviluppo di applicazioni per IBM MQ” a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

Riferimenti correlati

“Creazione di programmi C++ IBM MQ” a pagina 546
Viene elencato l' URL dei compilatori supportati, insieme ai comandi da utilizzare per compilare, collegare ed eseguire programmi C + + ed esempi su piattaforme IBM MQ .

Riferimento incrociato C++ e MQI

Classi C++ IBM MQ

Programmi di esempio C++

Vengono forniti quattro programmi di esempio, per dimostrare la ricezione e l'inserimento di messaggi.

I programmi di esempio sono:

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

I programmi di esempio si trovano nelle directory mostrate in Tabella 73 a pagina 532.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Tabella 73. Ubicazione dei programmi di esempio		
Ambiente	**Directory contenente l'origine**	**Directory contenente la build Programmi**
AIX AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
V 9.3.5 AIX AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (vedere la nota “1” a pagina 532)
IBM i IBM i	`/QIBM/ProdData/mqm/samp/`	(vedere la nota “2” a pagina 532)
Linux Linux	*MQ_INSTALLATION_PATH*/samp	Nessuna
Windows Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\esempi \bin\vn (vedere la nota “3” a pagina 532)
z/OS z/OS	thlqual.SCSQCPPS	

Note:

1. V 9.3.5 AIX I programmi creati utilizzando il compilatore XLC 17 si trovano nella cartella "`ca`", mentre i programmi creati utilizzando il compilatore XLC 16 si trovano nella cartella "`ia`".
2. IBM i I programmi creati utilizzando il compilatore ILE C+ + per IBM i si trovano nella libreria QMQM. I file di origine si trovano in `/QIBM/ProdData/mqm/samp`.
3. Windows I programmi creati utilizzando Microsoft Visual Studio Visual Studio si trovano in *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn`. Per ulteriori informazioni su questi compilatori, consultare “Creazione di programmi C++ su Windows” a pagina 551.

Programma di esempio HELLO WORLD (imqwrld.cpp)

Questo programma di esempio C++ mostra come inserire e richiamare un datagramma regolare (struttura C) utilizzando la classe ImqMessage .

Questo programma mostra come inserire e richiamare un datagramma regolare (struttura C) utilizzando la classe ImqMessage . Questo esempio utilizza pochi richiami del metodo, sfruttando i richiami del metodo impliciti come **open**, **close**e **disconnect**.

Su tutte le piattaforme tranne z/OS

Se si utilizza una connessione server a IBM MQ, seguire una delle procedure riportate di seguito:

- Per utilizzare la coda predefinita esistente, SYSTEM.DEFAULT.LOCAL.QUEUE, eseguire il programma **`imqwrlds`** senza passare alcun parametro
- Per utilizzare una coda assegnata dinamicamente temporanea, eseguire **`imqwrlds`** inoltrando il nome della coda modello predefinita, SYSTEM.DEFAULT.MODEL.QUEUE.

Se si utilizza una connessione client a IBM MQ, seguire una delle seguenti procedure:

- Impostare la variabile di ambiente MQSERVER (per ulteriori informazioni, consultare *MQSERVER*) ed eseguire **`imqwrldc`**oppure

- Eseguire **imqwrldc** passando come parametri **queue-name**, **queue-manager-name**e **channel-definition**, dove un tipico **channel-definition** potrebbe essere SYSTEM.DEF.SVRCONN/TCP/*nomehost* (1414)

Attivoz/OS

z/OS

Creare ed eseguire un lavoro batch, utilizzando il JCL di esempio **imqwrldr**.

Consultare z/OS Batch, RRS Batch e CICS per ulteriori informazioni.

Esempio di codice

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );
```

```
      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

Programmi di esempio SPUT (imqsput.cpp) e SGET (imqsget.cpp)

Questi programmi C+ + inseriscono e richiamano messaggi da una coda denominata.

Questi esempi mostrano l'utilizzo delle seguenti classi:

- ImqError (consultare ImqError C++ class)
- ImqMessage (consultare ImqMessage C++ class)
- ImqObject (consultare ImqObject C++ class)
- ImqQueue (consultare ImqQueue classe C++)

- ImqQueueManager (consultare ImqQueueClasse C++ Manager)

Seguire le istruzioni appropriate per eseguire i programmi.

Su tutte le piattaforme tranne z/OS

1. Eseguire **imqsputs** *nome - coda*.
2. Immettere le righe di testo nella console. Queste righe vengono poste come messaggi nella coda specificata.
3. Immettere una riga nulla per terminare l'input.
4. Eseguire **imqsgets** *nome - coda* per recuperare tutte le righe e visualizzarle nella console.

z/OS Per ulteriori informazioni, fare riferimento a "Creazione di programmi C+ + su z/OS Batch, RRS Batch e CICS" a pagina 553.

Attivoz/OS

z/OS

1. Creare ed eseguire un lavoro batch utilizzando il JCL di esempio **imqsputr**. I messaggi vengono letti dal dataset SYSIN.
2. Creare ed eseguire un lavoro batch utilizzando l'esempio JCL **imqsgetr**. I messaggi vengono richiamati dalla coda e inviati al dataset SYSPRINT.

Programma di esempio DPUT (imqdput.cpp)

Questo programma di esempio C++ inserisce i messaggi in un elenco di distribuzione composto da due code.

DPUT mostra l'utilizzo della classe List ImqDistribution(consultare ImqDistributionList C++ class). Questo esempio non è supportato su z/OS.

1. Eseguire **imqdputs** *queue-name-1 queue-name-2* per inserire i messaggi nelle due code denominate.
2. Eseguire **imqsgets** *queue-name-1* e **imqsgets** *queue-name-2* per recuperare i messaggi da queste code.

Considerazioni sul linguaggio C++

Questa raccolta di argomenti descrive in dettaglio gli aspetti dell'utilizzo del linguaggio C++ e le convenzioni che è necessario considerare quando si scrivono programmi applicativi che utilizzano MQI (Message Queue Interface).

File di intestazione C+ +

I file di intestazione vengono forniti come parte della definizione di MQI, per consentire la scrittura di programmi applicativi IBM MQ nel linguaggio C + +.

Questi file di intestazione sono riepilogati nella seguente tabella.

Tabella 74. File di intestazione C/C++

Nome file	**Indice**
IMQI.HPP	Classi C++ MQI (include CMQC.H e IMQTYPE.H)
IMQTYPE.H	Definisce il tipo di dati **ImqBoolean**
CMQC.H	Strutture di dati MQI e costanti manifest

Per migliorare la portabilità delle applicazioni, codificare il nome del file di intestazione in minuscolo nella direttiva del preprocessore **#include** :

```
#include <imqi.hpp> // C++ classes
```

Metodi e attributi C++

I nomi dei metodi sono in caratteri misti. Varie considerazioni si applicano ai parametri e ai valori di ritorno. È possibile accedere agli attributi utilizzando i metodi set e get, come appropriato.

I parametri dei metodi *const* sono solo per l'immissione. Parametri con firme che includono un puntatore (*) o un riferimento (&) sono passati per riferimento. I valori di ritorno che non includono un puntatore o un riferimento vengono passati in base al valore; nel caso di oggetti restituiti, si tratta di nuove entità che diventano responsabilità del chiamante.

Alcune firme del metodo includono elementi che, se non specificati, assumono un valore predefinito. Tali elementi si trovano sempre alla fine delle firme e sono indicati da un segno uguale (=); il valore dopo il segno uguale indica il valore predefinito che si applica se l'elemento viene omesso.

Tutti i nomi dei metodi in queste classi sono maiuscoli e minuscoli. Ogni parola, tranne la prima all'interno di un nome metodo, inizia con una lettera maiuscola. Le abbreviazioni non sono usate a meno che il loro significato non sia ampiamente compreso. Le abbreviazioni utilizzate includono *id* (per l'identità) e *sync* (per la sincronizzazione).

È possibile accedere agli attributi dell'oggetto utilizzando i metodi set e get. Un metodo set inizia con la parola *set* ; un metodo get non ha prefisso. Se un attributo è di *sola lettura*, non esiste alcun metodo set.

Gli attributi vengono inizializzati in stati validi durante la realizzazione dell'oggetto e lo stato di un oggetto è sempre congruente.

Tipi di dati in C++

Tutti i dati sono definiti dall'istruzione C **typedef** .

Il tipo **ImqBoolean** è definito come **unsigned char** in IMQTYPE.H e può avere i valori TRUE e FALSE.
È possibile utilizzare gli oggetti della classe **ImqBinary** al posto degli array **MQBYTE** e gli oggetti della classe **ImqString** al posto di **char ***. Molti metodi restituiscono oggetti invece di puntatori **char** o **MQBYTE** per semplificare la gestione della memoria. Tutti i valori di ritorno diventano responsabilità del chiamante e, nel caso di un oggetto restituito, la memoria può essere eliminata utilizzando l'eliminazione.

Manipolazione di stringhe binarie in C + +

Le stringhe di dati binari vengono dichiarate come oggetti della classe **ImqBinary** . Gli oggetti di questa classe possono essere copiati, confrontati e impostati utilizzando gli operatori C familiari. Viene fornito un codice di esempio.

Il seguente esempio di codice mostra le operazioni su una stringa binaria:

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
...
```

Manipolazione delle stringhe di carattere in C + +

I dati carattere vengono spesso restituiti in oggetti della classe **ImqString** che possono essere convertiti in **char *** utilizzando un operatore di conversione. La classe ImqString contiene i metodi per assistere nell'elaborazione delle stringhe di caratteri.

Quando i dati carattere vengono accettati o restituiti utilizzando i metodi MQI C + +, i dati carattere terminano sempre con valore null e possono essere di qualsiasi lunghezza. Tuttavia, alcuni limiti sono imposti da IBM MQ che potrebbero causare il troncamento delle informazioni. Per semplificare la gestione della memorizzazione, i dati di caratteri vengono spesso restituiti negli oggetti della classe **ImqString** . Questi oggetti possono essere assegnati a **char *** utilizzando l'operatore di conversione fornito e utilizzati per scopi di *sola lettura* in molte situazioni in cui è richiesto un **char *** .

Nota: Il risultato della conversione **char *** da un oggetto della classe **ImqString** potrebbe essere null.

Anche se le funzioni C possono essere utilizzate su **char ***, esistono metodi speciali della classe **ImqString** che sono preferibili; **operator length** () è l'equivalente di **strlen** e **storage** () indica la memoria assegnata per i dati carattere.

Stato iniziale degli oggetti in C + +

Tutti gli oggetti hanno uno stato iniziale coerente riflesso dai relativi attributi. I valori iniziali sono definiti nelle descrizioni delle classi.

Utilizzo di C da C++

Quando si utilizzano le funzioni C da un programma C + +, includere le intestazioni appropriate.

Il seguente esempio mostra `string.h` incluso in un programma C + +:

```
extern "C" {
#include <string.h>
}
```

Convenzioni notazionali C++

Questo esempio mostra come richiamare i metodi e dichiarare i parametri.

Questo esempio di codice utilizza i metodi e parametri **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

Dichiarare e utilizzare i parametri come segue:

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

...
}
```

Operazioni implicite in C++

Diverse operazioni possono verificarsi in modo implicito, *just in time*, per soddisfare le condizioni prerequisite per la corretta esecuzione di un metodo. Queste operazioni implicite sono connect, open,

reopen, close e disconnect. È possibile controllare la connessione e aprire il comportamento implicito utilizzando gli attributi della classe.

Connetti

Un oggetto gestore ImqQueueviene connesso automaticamente per qualsiasi metodo che risulti in una chiamata a MQI (consultare Riferimento incrociato C++ e MQI).

Apri

Un oggetto ImqObject viene aperto automaticamente per qualsiasi metodo che risulta in una chiamata MQGET, MQINQ, MQPUT o MQSET. Utilizzare il metodo **openFor** per specificare uno o più valori **open option** rilevanti.

Riapri

Un ImqObject viene riaperto automaticamente per qualsiasi metodo che risulta in una chiamata MQGET, MQINQ, MQPUT o MQSET, dove l'oggetto è già aperto, ma le **opzioni di apertura** esistenti non sono sufficienti per consentire la riuscita della chiamata MQI. L'oggetto viene temporaneamente chiuso utilizzando un valore **opzioni di chiusura** temporaneo di MQCO_NONE. Utilizzare il metodo **openFor** per aggiungere un**opzione di apertura**.

La riapertura può causare problemi in circostanze specifiche:

- Una coda dinamica temporanea viene eliminata quando viene chiusa e non può mai essere riaperta.
- Una coda aperta per l'input esclusivo (esplicitamente o per impostazione predefinita) potrebbe essere accessibile da altri utenti nella finestra di opportunità durante la chiusura e la riapertura.
- Una posizione del cursore di ricerca viene persa quando una coda viene chiusa. Questa situazione non impedisce la chiusura e la riapertura, ma impedisce l'uso successivo del cursore fino a quando MQGMO_BROWSE_FIRST non viene utilizzato nuovamente.
- Il contesto dell'ultimo messaggio richiamato viene perso quando una coda viene chiusa.

Se si verifica o è possibile prevedere una di queste circostanze, evitare le riaperture impostando esplicitamente le **opzioni di apertura** adeguate prima che un oggetto venga aperto (esplicitamente o implicitamente).

L'impostazione esplicita delle **opzioni di apertura** per situazioni di gestione delle code complesse consente di ottenere prestazioni migliori ed evita i problemi associati all'utilizzo della riapertura.

Chiudi

Un ImqObject viene chiuso automaticamente in qualsiasi punto in cui lo stato dell'oggetto non è più valido, ad esempio se un riferimento di connessione ImqObject viene interrotto o se un oggetto ImqObject viene eliminato.

Disconnetti

Un gestore ImqQueueviene disconnesso automaticamente in qualsiasi momento in cui la connessione non è più praticabile, ad esempio se un riferimento di connessione ImqObject viene interrotto o se un oggetto gestore ImqQueueviene eliminato.

Stringhe binarie e di caratteri in C++

La classe ImqString incapsula il formato dati *char ** tradizionale. La classe ImqBinary incapsula la schiera di byte binari. Alcuni metodi che impostano i dati carattere potrebbero troncare i dati.

Metodi che impostano il carattere (**char ***) I dati prendono sempre una copia dei dati, ma alcuni metodi potrebbero troncare la copia, poiché alcuni limiti sono imposti da IBM MQ.

La classe ImqString (vedere ImqString Classe C + +) incapsula il tradizionale **char *** e fornisce supporto per:

- Confronto
- Concatenamento
- In fase di copia
- Conversione da intero a testo e da testo a intero
- Estrazione token (parola)
- Traduzione in maiuscolo

La classe ImqBinary (consultare ImqBinary Classe C++) incapsulare array di byte binari di dimensione arbitraria. In particolare, viene utilizzato per contenere i seguenti attributi:

- **token di account** (MQBYTE32)
- **tag di connessione** (MQBYTE128)
- **ID correlazione** (MQBYTE24)
- **token funzione** (MQBYTE8)
- **ID gruppo** (MQBYTE24)
- **ID istanza** (MQBYTE24)
- **ID messaggio** (MQBYTE24)
- **token messaggio** (MQBYTE16)
- **ID istanza transazione** (MQBYTE16)

Dove questi attributi appartengono agli oggetti delle seguenti classi:

- ImqCICSBridgeHeader (consultare ImqCICSBridgeHeader C++ class)
- ImqGetMessageOptions (vedere ImqGetMessageOptions classe C++)
- ImqIMSBridgeHeader (consultare ImqIMSBridgeHeader C++ class)
- ImqMessageTracker (consultare ImqMessageClasse C++ Tracker)
- ImqQueueManager (consultare ImqQueueClasse C++ Manager)
- ImqReferenceIntestazione (consultare ImqReferenceclasse C++ intestazione)
- ImqWorkIntestazione (vedere ImqWorkclasse C++ intestazione)

La classe ImqBinary fornisce anche supporto per il confronto e la copia.

Funzioni non supportate in C++

Le classi e i metodi IBM MQ C+ + sono indipendenti dalla piattaforma IBM MQ . Potrebbero quindi offrire alcune funzioni che non sono supportate su determinate piattaforme.

Se si tenta di utilizzare una funzione su una piattaforma su cui non è supportato, la funzione viene rilevata da IBM MQ ma non dai bind di linguaggio C++. IBM MQ riporta l'errore al programma, come qualsiasi altro errore MQI.

Messaggistica in C++

Questa raccolta di argomenti descrive in dettaglio come preparare, leggere e scrivere i messaggi in C + +.

Preparazione dei dati dei messaggi in C + +

I dati del messaggio vengono preparati in un buffer, che può essere fornito dal sistema o dall'applicazione. Ci sono vantaggi in entrambi i metodi. Vengono forniti esempi di utilizzo di un buffer.

Quando si invia un messaggio, i dati del messaggio vengono preparati per la prima volta in un buffer gestito da un oggetto ImqCache (consultare ImqCache classe C++). Un buffer viene associato (per eredità) a ciascun oggetto ImqMessage (consultare ImqMessage C++ class): può essere fornito dall'applicazione (utilizzando il metodo **useEmptyBuffer** o **useFullBuffer**) o automaticamente dal sistema. Il vantaggio dell'applicazione che fornisce il buffer dei messaggi è che in molti casi non

è necessaria alcuna copia dei dati, poiché l'applicazione può utilizzare direttamente le aree di dati preparate. Lo svantaggio è che il buffer fornito è di lunghezza fissa.

Il buffer può essere riutilizzato e il numero di byte trasmessi può essere modificato ogni volta, utilizzando il metodo **setMessageLength** prima della trasmissione.

Quando vengono forniti automaticamente dal sistema, il numero di byte disponibili viene gestito dal sistema e i dati possono essere copiati nel buffer dei messaggi utilizzando, ad esempio, il metodo ImqCache **write** o il metodo ImqMessage **writeItem** . Il buffer dei messaggi cresce in base alle esigenze. Quando il buffer cresce, non si verifica alcuna perdita di dati precedentemente scritti. Un messaggio di grandi dimensioni o a più parti può essere scritto in parti sequenziali.

I seguenti esempi mostrano gli invii di messaggi semplificati.

1. Utilizzare i dati preparati in un buffer fornito dall'utente

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. Utilizzare i dati preparati in un buffer fornito dall'utente, dove la dimensione del buffer supera la dimensione dei dati

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. Copiare i dati in un buffer fornito dall'utente

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. Copia i dati in un buffer fornito dal sistema

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. Copiare i dati in un buffer fornito dal sistema utilizzando oggetti (oggetti che impostano il formato del messaggio e il contenuto)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

Lettura dei messaggi in C++

Un buffer può essere fornito dall'applicazione o dal sistema. È possibile accedere ai dati direttamente dal buffer o leggerli in modo sequenziale. Esiste una classe equivalente a ciascun tipo di messaggio. Viene fornito un codice di esempio.

Quando si ricevono i dati, l'applicazione o il sistema possono fornire un buffer di messaggi adatto. Lo stesso buffer può essere utilizzato sia per la trasmissione multipla che per la ricezione multipla per un particolare oggetto ImqMessage . Se il buffer di messaggi viene fornito automaticamente, aumenta in modo da contenere la lunghezza dei dati ricevuti. Tuttavia, un buffer di messaggi fornito dall'applicazione potrebbe non essere abbastanza grande da contenere i dati ricevuti. Quindi, potrebbe verificarsi un troncamento o un errore, a seconda delle opzioni utilizzate per la ricezione del messaggio.

È possibile accedere ai dati in entrata direttamente dal buffer dei messaggi, nel qual caso la lunghezza dei dati indica la quantità totale di dati in entrata. In alternativa, i dati in entrata possono essere letti sequenzialmente dal buffer di messaggi. In questo caso, il puntatore dati si indirizza al byte successivo di dati in entrata e il puntatore dati e la lunghezza dati vengono aggiornati ogni volta che i dati vengono letti.

Gli *elementi* sono parti di un messaggio, tutte nell'area utente del buffer di messaggi, che devono essere elaborate in modo sequenziale e separato. A parte i dati utente regolari, un elemento potrebbe essere un'intestazione dead-letter o un messaggio trigger. Gli elementi sono sempre associati ai formati dei messaggi; i formati dei messaggi **non** vengono associati sempre agli elementi.

Esiste una classe di oggetti per ogni elemento che corrisponde a un formato di messaggio IBM MQ riconoscibile. Ce n' è uno per un'intestazione di lettera non recapitabile e uno per un messaggio trigger. Non esiste alcuna classe di oggetto per i dati utente. Vale a dire, una volta esauriti i formati riconoscibili, l'elaborazione del resto viene lasciata al programma applicativo. Le classi per i dati utente possono essere scritte specializzando la classe ImqItem .

Il seguente esempio mostra una ricevuta di messaggio che tiene conto di un numero di potenziali elementi che possono precedere i dati dell'utente, in una situazione immaginaria. I dati utente non elementi sono definiti come qualsiasi cosa che si verifica dopo gli elementi che possono essere identificati. Un buffer automatico (predefinito) viene utilizzato per contenere una quantità arbitraria di dati del messaggio.

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }
```

```
    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

In questo esempio, FMT_USERCLASS è una costante che rappresenta il nome formato di 8 caratteri associato a un oggetto della classe `UserClass`ed è definito dall'applicazione.

`UserClass` deriva dalla classe ImqItem (vedere ImqItem C++ class) e implementa i metodi **copyOut** e **pasteIn** virtuali di tale classe.

I successivi due esempi mostrano il codice dalla classe ImqDeadLetterHeader (consultare ImqDeadLetterHeader C++ class). Il primo esempio mostra un messaggio incapsulato personalizzato - codice di *scrittura* .

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }
```

```
  return bSuccess ;
}
```

Il secondo esempio mostra il messaggio incapsulato personalizzato - codice di *lettura* .

```
// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

Con un buffer automatico, la memoria buffer è *volatile*. Vale a dire, i dati del buffer potrebbero essere conservati in un'altra ubicazione fisica dopo ogni richiamo del metodo **get** . Pertanto, ogni volta che si fa riferimento ai dati del buffer, utilizzare i metodi **bufferPointer** o **dataPointer** per accedere ai dati del messaggio.

È possibile che si desideri che un programma riservi un'area fissa per la ricezione dei dati del messaggio. In questo caso, richiamare il metodo **useEmptyBuffer** prima di utilizzare il metodo **get** .

L'uso di un'area fissa e non automatica limita i messaggi ad una dimensione massima, pertanto è importante considerare l'opzione MQGMO_ACCEPT_TRUNCATED_MSG dell'oggettoMessageOptions ImqGet. Se questa opzione non viene specificata (impostazione predefinita), è possibile che sia previsto il codice motivo MQRC_TRUNCATED_MSG_FAILED. Se questa opzione viene specificata, è possibile che il codice motivo MQRC_TRUNCATED_MSG_ACCEPTED sia previsto in base alla progettazione dell'applicazione.

L'esempio successivo mostra come è possibile utilizzare un'area fissa di memoria per ricevere i messaggi:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

In questo frammento di codice, il buffer può sempre essere indirizzato direttamente, con *pszBuffer*, invece di utilizzare il metodo **bufferPointer** . Tuttavia, è preferibile utilizzare il metodo **dataPointer** per l'accesso per scopi generici. L'applicazione (non l'oggetto classe ImqCache) deve eliminare un buffer definito dall'utente (non automatico).

Attenzione: la specifica di un puntatore null e di una lunghezza zero con **useEmptyBuffer** non denomina un buffer a lunghezza fissa di lunghezza zero come potrebbe essere previsto. Questa combinazione viene interpretata come una richiesta di ignorare qualsiasi buffer definito dall'utente precedente e di tornare invece all'utilizzo di un buffer automatico.

Scrittura di un messaggio nella coda di messaggi non recapitabili in C + +

Codice programma di esempio per la scrittura di un messaggio nella coda di messaggi non recapitabili.

Un caso tipico di un messaggio multipart è quello che contiene un'intestazione di lettera non recapitabile. I dati da un messaggio che non può essere elaborato vengono accodati all'intestazione del messaggio non instradabile.

```
ImqQueueManager mgr ;        // The queue manager.
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

Scrittura di un messaggio sul bridge IMS in C++

Codice di programma di esempio per la scrittura di un messaggio sul bridge IMS .

I messaggi inviati al bridge IBM MQ - IMS potrebbero utilizzare un'intestazione speciale. L'intestazione del bridge IMS ha come prefisso i dati dei messaggi regolari.

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.
```

```
// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

Scrittura di un messaggio in CICS bridge in C+ +

Codice programma di esempio per la scrittura di un messaggio in CICS bridge.

I messaggi inviati a IBM MQ for z/OS utilizzando CICS bridge richiedono un'intestazione speciale. L'intestazione CICS bridge ha come prefisso i dati dei messaggi regolari.

```
ImqQueueManager mgr ;         // The queue manager.
ImqQueue queueIn ;            // Incoming message queue.
ImqQueue queueBridge ;        // CICS bridge message queue.
ImqMessage msg ;              // Incoming and outgoing message.
ImqCicsBridgeHeader header ;  // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

Scrittura di un messaggio con un'intestazione di lavoro in C++

Codice di esempio del programma per la scrittura di un messaggio destinato a una coda gestita da z/OS Workload Manager.

I messaggi inviati a IBM MQ for z/OS, destinati a una coda gestita da z/OS Workload Manager, richiedono un'intestazione speciale. L'intestazione del lavoro ha come prefisso i dati dei messaggi regolari.

```
ImqQueueManager mgr ;             // The queue manager.
ImqQueue queueIn ;                // Incoming message queue.
ImqQueue queueWLM ;               // WLM managed queue.
ImqMessage msg ;                  // Incoming and outgoing message.
ImqWorkHeader header ;            // Work header information

// Retrieve the message to be forwarded.
```

```
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

Creazione di programmi C++ IBM MQ

Viene elencato l' URL dei compilatori supportati, insieme ai comandi da utilizzare per compilare, collegare ed eseguire programmi C + + ed esempi su piattaforme IBM MQ .

Per un elenco dei compilatori per ogni versione e piattaforma supportata di IBM MQ, consultare Requisiti di sistema per IBM MQ.

Il comando necessario per compilare e collegare il programma IBM MQ C + + dipende dall'installazione e dai requisiti. Gli esempi che seguono mostrano i tipici comandi di compilazione e collegamento per alcuni compilatori che utilizzano l'installazione predefinita di IBM MQ su diverse piattaforme.

AIX Creazione di programmi C++ su AIX

Creare programmi IBM MQ C++ su AIX utilizzando il compilatore XL C Enterprise Edition .

V 9.3.5 Per ulteriori informazioni sulla diversa associazione delle opzioni del compilatore tra compilatori XLC 16 e XLC 17, consultare Associazione delle opzioni.

V 9.3.5 Deprecated Il supporto per il compilatore XL C/C++ per AIX 16 su AIX è obsoleto da IBM MQ 9.3.5.

Client

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Applicazione senza thread a 32 bit

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

Applicazione con thread a 32 bit

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Applicazione senza thread a 64 bit

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Applicazione con thread a 64 bit

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.3.5 **Applicazione senza thread a 32 bit (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **Applicazione con thread a 32 bit (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.3.5 **Applicazione senza thread a 64 bit (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **Applicazione con thread a 64 bit (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

Server

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Applicazione senza thread a 32 bit

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

Applicazione con thread a 32 bit

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Applicazione senza thread a 64 bit

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Applicazione con thread a 64 bit

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.3.5 **Applicazione senza thread a 32 bit (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 **Applicazione con thread a 32 bit (XLC 17)**

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.3.5 **Applicazione senza thread a 64 bit (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 **Applicazione con thread a 64 bit (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i Creazione di programmi C++ su IBM i

Creare programmi IBM MQ C+ + su IBM i utilizzando il compilatore ILE C++.

IBM ILE C++ per IBM i è un compilatore nativo per programmi C + +. Le seguenti istruzioni descrivono come utilizzare questo compilatore per creare applicazioni IBM MQ C+ + utilizzando *Hello World!* Programma di esempio IBM MQ come esempio.

1. Installare il compilatore ILE C++ per IBM i come indicato in *Leggimi prima!* manuale che accompagna il prodotto.
2. Assicurarsi che la libreria QCXXN si trovi nell'elenco librerie.
3. Creare il programma di esempio HELLO WORLD:
   a. Creare un modulo:

   ```
   CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
   SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
   INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
   TERASPACE(*YES)
   ```

   L'origine per i programmi di esempio C + + è disponibile in `/QIBM/ProdData/mqm/samp` e i file di inclusione in `/QIBM/ProdData/mqm/inc`.

   In alternativa, l'origine può essere trovata nella libreria `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)`.

   b. Collegarlo ai programmi di servizio forniti da IBM MQper produrre un oggetto programma:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
   ```

   Per creare un'applicazione con thread, utilizzare i programmi di servizio rientranti:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
   ```

   c. Eseguire il programma di esempio HELLO WORLD utilizzando SYSTEM.DEFAULT.LOCAL.QUEUE:

   ```
   CALL PGM(MYLIB/IMQWRLD)
   ```

Linux Creazione di programmi C++ su Linux

Creare programmi IBM MQ C+ + su Linux utilizzando il compilatore GNU g + +.

System p

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Client: System p

Applicazione senza thread a 32 bit

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
```

```
-limqc23gl
-limqb23gl -lmqic
```

Applicazione con thread a 32 bit

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

Applicazione senza thread a 64 bit

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Applicazione con thread a 64 bit

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Server: System p

Applicazione senza thread a 32 bit

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Applicazione con thread a 32 bit

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

Applicazione senza thread a 64 bit

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Applicazione con thread a 64 bit

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Client: IBM Z

Applicazione senza thread a 32 bit

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Applicazione con thread a 32 bit

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

Applicazione senza thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Applicazione con thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Server: IBM Z

Applicazione senza thread a 32 bit

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Applicazione con thread a 32 bit

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Applicazione senza thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Applicazione con thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 bit)

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Client: x86-64 (32 bit)

Applicazione senza thread a 32 bit

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

Applicazione con thread a 32 bit

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Applicazione senza thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

Applicazione con thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Server: x86-64 (32 bit)

Applicazione senza thread a 32 bit

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

Applicazione con thread a 32 bit

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Applicazione senza thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Applicazione con thread a 64 bit

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows Creazione di programmi C++ su Windows

Creare programmi C++ IBM MQ su Windows utilizzando il compilatore C++ Microsoft Visual Studio .

Attenzione: Le librerie fornite da IBM MQ sono librerie dinamiche e non statiche. IBM MQ fornisce qualcosa noto come "`import libraries`" che è possibile utilizzare solo durante il tempo di compilazione. Per il runtime, è necessario utilizzare le librerie dinamiche.

Da IBM MQ 8.0.0 Fix Pack 4, IBM MQ fornisce client ridistribuibili, contenenti le librerie richieste per l'esecuzione delle applicazioni IBM MQ . Queste librerie possono essere compresse e

ridistribuite con le applicazioni client. Per ulteriori informazioni, consultare Redistributable clients on Windows.

I file della libreria (.lib) e i file dll da utilizzare con applicazioni a 32 bit sono installati in *MQ_INSTALLATION_PATH*\Tools\Lib. I file da utilizzare con applicazioni a 64 - bit sono installati in *MQ_INSTALLATION_PATH*\Tools\Lib64. *MQ_INSTALLATION_PATH* rappresenta la directory di livello superiore in cui è installato IBM MQ .

Client

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

Server

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Installazione del runtime C universale

Se si utilizza Windows 8.1 o Windows Server 2012 R2, è necessario installare l'Universal C runtime update (Universal CRT) da Microsoft. Questo runtime è incluso come parte di Windows 10 e Windows Server 2016.

L'aggiornamento CRT universale è Microsoft update KB3118401. È possibile verificare se si dispone di questo aggiornamento ricercando un file denominato `ucrtbase.dll` nella directory `C:\Windows\System32` . In caso contrario, è possibile scaricare l'aggiornamento dalla seguente pagina Microsoft : https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401.

Se si tenta di eseguire un programma IBM MQ o un programma che si compila utilizzando Microsoft Visual Studio 2017, senza il runtime installato, si verificano errori come il seguente:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Fornitura di runtime per programmi Microsoft Visual Studio 2012

Se è stato compilato un programma IBM MQ utilizzando Microsoft Visual Studio 2012, tenere presente che il programma di installazione IBM MQ non installa i runtime Microsoft Visual Studio 2012 C/C++. Se la versione precedente di IBM MQ è stata installata sullo stesso computer, i runtime Microsoft Visual Studio 2012 sono disponibili da tale installazione.

Tuttavia, se si utilizza un programma creato utilizzando Microsoft Visual Studio 2012 e non è stata installata alcuna versione precedente di IBM MQ , è necessario effettuare una delle seguenti operazioni:

- Scaricare e installare **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** da Microsoft.
- Ricompilare il programma con Microsoft Visual Studio 2017o un altro livello Microsoft Visual Studio per cui sono installati i runtime.

Librerie client C++ create utilizzando il compilatore Microsoft Visual Studio 2015 .

IBM MQ fornisce librerie client C+ + create con il compilatore C+ + Microsoft Visual Studio 2015 e il compilatore C++ Microsoft Visual Studio 2017 .

Vengono fornite sia le versioni a 32 bit che a 64 bit delle librerie IBM MQ C + +. Le librerie a 32 bit sono installate nella cartella `bin\vs2015` e le librerie a 64 bit sono installate nelle cartelle `bin64\vs2015` .

Per default, IBM MQ è configurato per utilizzare le librerie Microsoft Visual Studio 2017 . Per utilizzare le librerie Microsoft Visual Studio 2015 , è necessario impostare la variabile di ambiente

MQ_PREFIX_VS_LIBRARIES su MQ_PREFIX_VS_LIBRARIES=vs2015 prima di installare IBM MQo prima di utilizzare il comando **setmqenv** o **setmqinst** .

Utilizzo di librerie IBM MQ C++ denominate in modo diverso

IBM MQ fornisce alcune librerie client C++ aggiuntive denominate in modo diverso. Queste librerie vengono create con i compilatori C++ Microsoft Visual Studio 2015 e Microsoft Visual Studio 2017 . Queste librerie vengono fornite in aggiunta alle librerie C + + esistenti create anche con il compilatore Microsoft Visual Studio 2017 C + +. Poiché queste librerie IBM MQ C++ aggiuntive hanno nomi differenti, è possibile eseguire applicazioni IBM MQ C + + create utilizzando IBM MQ C+ + e compilate con Microsoft Visual Studio 2017 e versioni precedenti del prodotto sullo stesso computer.

Le ulteriori librerie Microsoft Visual Studio 2017 hanno i seguenti nomi:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

Le ulteriori librerie Microsoft Visual Studio 2015 hanno i seguenti nomi:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

Vengono fornite sia le versioni a 32 bit che a 64 bit di tali librerie. Le librerie a 32 bit sono installate sotto la cartella `bin` e le librerie a 64 - bit sono installate nella cartella `bin64` . Le librerie di importazione corrispondenti sono installate nelle directory `Tools\lib` e `Tools\lib64` .

Se l'applicazione utilizza i file `imq*vs2015.lib` , è necessario compilarli utilizzando il compilatore Microsoft Visual Studio 2015 . Per eseguire le applicazioni IBM MQ C++ compilate con Microsoft Visual Studio 2015o le applicazioni compilate con una versione precedente del prodotto sullo stesso computer, la variabile di ambiente PATH deve avere il prefisso come mostrato nei seguenti esempi:

- Per applicazioni a 32 bit:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- Per le applicazioni a 64 bit:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

Concetti correlati
Windows: modifiche da IBM MQ 8.0

z/OS Creazione di programmi C+ + su z/OS Batch, RRS Batch e CICS

Creare i programmi IBM MQ C++ su z/OS per gli ambienti Batch, RRS batch o CICS ed eseguire i programmi di esempio.

È possibile scrivere programmi C++ per tre ambienti supportati da IBM MQ for z/OS :

- Batch
- Batch RRS
- CICS

Compila, precollega e collega

Crea un'applicazione z/OS compilando, precollegando e modificando il tuo codice sorgente C + +.

IBM MQ C++ per z/OS è implementato come DLL z/OS per la lingua IBM C++ per z/OS . Utilizzando le DLL, è possibile concatenare i sidedeck di definizione forniti con l'output del compilatore in fase di precollegamento. Ciò consente al linker di controllare le chiamate alle funzioni del membro C++ IBM MQ .

Nota: Ci sono tre serie di ponti laterali per ciascuno dei tre ambienti.

Per creare un'applicazione IBM MQ for z/OS C+ +, creare e eseguire JCL. Utilizzare la seguente procedura:

1. Se l'applicazione viene eseguita in CICS, utilizzare la procedura fornita da CICSper convertire i comandi CICS nel programma.

   Inoltre, per applicazioni CICS è necessario:

   a. Aggiungere la libreria SCSQLOAD alla concatenazione DFHRPL.

   b. Definire il gruppo CEDA CSQCAT1 utilizzando il membro IMQ4B100 nella libreria SCSQPROC.

   c. Installare CSQCAT1.
2. Compilare il programma per produrre il codice oggetto. Il JCL per la compilazione deve includere istruzioni che rendono i file di definizione dei dati del prodotto disponibili per il compilatore. Le definizioni dei dati vengono fornite nelle seguenti librerie IBM MQ for z/OS :

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Accertarsi di specificare l'opzione del compilatore `/cxx` .

   Nota: Il nome **thlqual** è il qualificatore di livello superiore della libreria di installazione IBM MQ su z/OS.
3. Pre - collegare il codice oggetto creato nel passo “2” a pagina 554, inclusi i seguenti sidedeck di definizione, forniti in **thlqual**.SCSQDEFS:

   a. imqs23dm e imqb23dm per batch

   b. imqs23dr e imqb23dr per batch RRS

   c. imqs23dc e imqb23dc per CICS

   Queste sono le DLL corrispondenti.

   a. imqs23im e imqb23im per batch

   b. imqs23ir e imqb23ir per batch RRS

   c. imqs23ic e imqb23ic per CICS
4. Modificare il link del codice oggetto creato nel passo “3” a pagina 554per creare un modulo di caricamento e memorizzarlo nella libreria di caricamento dell'applicazione.

Per eseguire programmi batch o batch RRS, includere le librerie **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD nella concatenazione di dataset STEPLIB o JOBLIB.

Per eseguire un programma CICS , prima chiedere all'amministratore di sistema di definirlo in CICS come programma e transazione IBM MQ . È quindi possibile eseguirlo nel solito modo.

Eseguire i programmi di esempio

I programmi sono descritti in “Programmi di esempio C++” a pagina 531.

Le applicazioni di esempio vengono fornite solo in formato origine. I file sono:

Tabella 75. File di programma di esempio z/OS		
Esempio	**Programma di origine (nella libreria thlqual.SCSQCPPS)**	**JCL (nella libreria thlqual.SCSQPROC)**
Hello World	imqwrld	Imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

Per eseguire gli esempi, compilarli e modificarli come con qualsiasi programma C++ (consultare “Creazione di programmi C+ + su z/OS Batch, RRS Batch e CICS” a pagina 553). Utilizzare il JCL fornito per costruire ed eseguire un lavoro batch. È necessario inizialmente personalizzare il JCL, seguendo il commento incluso con esso.

z/OS Creazione di programmi C++ su z/OS UNIX System Services

Creare programmi IBM MQ C+ + su z/OS UNIX System Services (z/OS UNIX).

Per creare un'applicazione sotto la shell z/OS UNIX , è necessario fornire al compilatore l'accesso ai file di inclusione IBM MQ (ubicati in `thlqual.SCSQC370` e `hlqual.SCSQHPPS`) e il collegamento rispetto a due dei sidedeck DLL (ubicati in `thlqual.SCSQDEFS`). Al runtime, l'applicazione deve accedere ai IBM MQ dataset `thlqual.SCSQLOAD`, `thlqual.SCSQAUTHe` a uno dei dataset specifici della lingua, ad esempio `thlqual.SCSQANLE`[6].

Compilazione

1. Copiare l'esempio nel file system utilizzando il comando TSO **oput** oppure utilizzare FTP. Il resto di questo esempio presuppone che l'esempio sia stato copiato in una directory denominata `/u/fred/samplee` che sia stato denominato `imqwrld.cpp`.
2. Accedi alla shell z/OS UNIX e passa alla directory in cui hai inserito l'esempio.
3. Configurare il compilatore C++ in modo che possa accettare i file DLL sidedeck e .cpp come input:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Compilare e collegare il programma di esempio. Il seguente comando collega il programma con i sidedeck batch; è possibile utilizzare invece i sidedeck batch RRS. Il carattere \ viene utilizzato per suddividere il comando su più di una riga. Non immettere questo carattere; immettere il comando come una singola riga:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

Per ulteriori informazioni sul comando TSO **oput** , fare riferimento al manuale z/OS UNIX Command Reference.

È anche possibile utilizzare il programma di utilità make per semplificare la creazione di programmi C + +. Ecco un makefile di esempio per creare il programma di esempio HELLO WORLD C++. Separa le fasi di compilazione e di collegamento. Impostare l'ambiente come nel passo “3” a pagina 555 prima di eseguire make.

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

[6] È possibile collegarsi a uno qualsiasi dei sidedeck elencati in " Pre - link del codice oggetto per eseguire il proprio z/OS UNIX in uno dei tre ambienti, “Creazione di programmi C+ + su z/OS Batch, RRS Batch e CICS” a pagina 553

```
imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Fare riferimento a z/OS UNIX System Services Strumenti di programmazione per ulteriori informazioni sull'utilizzo di make.

In esecuzione

1. Accedere alla shell z/OS UNIX e passare alla directory in cui è stato creato l'esempio.
2. Impostare la variabile di ambiente STEPLIB per includere i dataset IBM MQ :

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. Eseguire l'esempio:

```
/u/fred/sample:> ./imqwrld
```

Sviluppo di applicazioni .NET

IBM MQ classes for .NET consentire alle applicazioni .NET di connettersi a IBM MQ come IBM MQ MQI client o di connettersi direttamente a un server IBM MQ .

Se si dispone di applicazioni che utilizzano Microsoft .NET Framework e si desidera usufruire delle funzioni di IBM MQ, è necessario utilizzare IBM MQ classes for .NET. Per ulteriori informazioni, consultare “InstallazioneIBM MQ classes for .NET Framework” a pagina 563.

Da IBM MQ 9.1.1, IBM MQ supporta .NET Core per le applicazioni in ambienti Windows . Per ulteriori informazioni, consultare “InstallazioneIBM MQ classes for .NET” a pagina 557.

Da IBM MQ 9.1.2, IBM MQ supporta .NET Core per le applicazioni in ambienti Linux .

Da IBM MQ 9.1.4, le applicazioni gestite IBM MQ .NET sono in grado di bilanciare automaticamente le connessioni tra i gestori code con cluster. Sono supportate sia le librerie .NET Framework che .NET Standard . Per ulteriori informazioni, vedi Informazioni sui cluster uniformi e Bilanciamento automatico dell'applicazione.

L'interfaccia IBM MQ .NET orientata agli oggetti è diversa dall'interfaccia MQI in quanto utilizza metodi di oggetti piuttosto che utilizzare i verbi MQI.

L'API (application programming interface) procedurale IBM MQ si basa su verbi come quelli presenti nel seguente elenco:

```
       MQCONN, MQDISC, MQOPEN, MQCLOSE,
       MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Questi verbi prendono tutti, come parametro, un handle per l'oggetto IBM MQ su cui devono operare. Poiché .NET è orientato agli oggetti, l'interfaccia di programmazione .NET gira questo round. Il programma consiste in una serie di oggetti IBM MQ , su cui si agisce richiamando i metodi su tali oggetti. È possibile scrivere programmi in qualsiasi lingua supportata da .NET.

Quando si utilizza l'interfaccia procedurale, ci si disconnette da un gestore code utilizzando la chiamata MQDISC (*Hconn*, CompCode, Reason), dove *Hconn* è un handle per il gestore code.

Nell'interfaccia .NET , il gestore code è rappresentato da un oggetto della classe MQQueueManager. È possibile disconnettersi dal gestore code richiamando il metodo Disconnect () su tale classe.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

IBM MQ classes for .NET è una serie di classi che consentono alle applicazioni .NET di interagire con IBM MQ. Essi rappresentano i vari componenti di IBM MQ utilizzati dall'applicazione, ad esempio gestori code, code, canali e messaggi. Per i dettagli di queste classi, consultare Le classi e le interfacce IBM MQ .NET.

Prima di poter compilare le applicazioni scritte, è necessario che sia installato .NET Framework. Per istruzioni sull'installazione di IBM MQ classes for .NET e .NET Framework, consultare “InstallazioneIBM MQ classes for .NET Framework” a pagina 563.

Concetti correlati

Panoramica tecnica

“Sviluppo di applicazioni per IBM MQ” a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

Attività correlate

Come contattare il supporto IBM

Risoluzione dei problemi di IBM MQ .NET

“Sviluppo di applicazioni Microsoft Windows Communication Foundation con IBM MQ” a pagina 1270
Il canale personalizzato WCF (Microsoft Windows Communication Foundation) per IBM MQ invia e riceve messaggi tra i servizi e i client WCF.

“Sviluppo di applicazioni XMS .NET” a pagina 616
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) fornisce un'API (application programming interface) denominata XMS che ha la stessa serie di interfacce di Java Message Service (JMS) API. IBM MQ Message Service Client (XMS) for .NET contiene un'implementazione completamente gestita di XMS, che può essere utilizzata da qualsiasi linguaggio conforme a .NET .

Linux Windows InstallazioneIBM MQ classes for .NET

IBM MQ classes for .NET, inclusi gli esempi, sono stati installati con IBM MQ su Windows e Linux

Prerequisiti e installazione

Da IBM MQ 9.2.0, la libreria client IBM MQ .NET creata utilizzando .NET Standard è disponibile su Windows e Linux. Per eseguire IBM MQ classes for .NET Standard, è necessario installare Microsoft .NET Core. Microsoft.NET Core 3.1 è la versione minima richiesta per l'esecuzione di IBM MQ classes for .NET Standard.

V 9.3.0 V 9.3.0 Da IBM MQ 9.3.0, IBM MQ supporta le applicazioni .NET 6 utilizzando IBM MQ classes for .NET Standard. Se si utilizza un'applicazione .NET Core 3.1 , è possibile eseguire questa applicazione con una piccola modifica nel file `csproj` , impostando `targetframeworkversion` su `"net6.0"`, senza alcuna ricompilazione richiesta.

V 9.3.1 IBM MQ 9.3.1 fornisce una libreria client IBM MQ .NET creata rispetto a .NET 6 come framework di destinazione. Da IBM MQ 9.3.1, Microsoft .NET 6.0 è la versione minima richiesta per eseguire le applicazioni utilizzando le librerie IBM MQ create utilizzando .NET 6 come framework di destinazione.

V 9.3.1 Da IBM MQ 9.3.1, la libreria client IBM MQ .NET creata utilizzando .NET Standard è disponibile in una nuova cartella `netstandard2.0` e la libreria client IBM MQ .NET creata utilizzando .NET 6 come framework di destinazione è disponibile in *MQ_INSTALLATION_PATH*`/bin` su Windows e in *MQ_INSTALLATION_PATH*`/lib64` su Linux.

L'ultima versione di IBM MQ classes for .NET viene installata per impostazione predefinita come parte dell'installazione standard di IBM MQ nella funzione *Servizi Web e messaggistica Java e .NET* .

Windows Per ulteriori informazioni sui prerequisiti e sull'installazione su Windows:

- Consultare Requisiti per IBM MQ classes for .NET, per il software prerequisito per eseguire IBM MQ classes for .NET.
- Consultare Installazione del server IBM MQ su Windows o Installazione di un client IBM MQ su sistemi Windows per le istruzioni di installazione.

Linux Per ulteriori informazioni sui prerequisiti e sull'installazione su Linux:

- Consultare Requisiti per IBM MQ classes for .NET, per il software prerequisito per eseguire IBM MQ classes for .NET.
- Per istruzioni di installazione rpm, consultare Installazione di un client IBM MQ su sistemi Linux.
- Per Linux Ubuntu, utilizzando i pacchetti Debian , vedi Installazione di un client IBM MQ su sistemi Linux.

La libreria IBM MQ classes for .NET Standard , `amqmdnetstd.dll`, è disponibile per il download dal repository NuGet . Per ulteriori informazioni, consultare “Download di IBM MQ classes for .NET dal repository NuGet” a pagina 562.

`amqmdnetstd.dll` libreria

V 9.3.1 Da IBM MQ 9.3.1, la libreria `amqmdnetstd.dll` è disponibile nelle seguenti ubicazioni

Libreria creata utilizzando .NET Standard 2.0 come framework di destinazione

- Windows Su Windows *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`.
- Linux Su Linux *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`.

Deprecated Queste librerie sono obsolete e IBM intende rimuoverle nelle release future.

Libreria creata utilizzando .NET 6 come framework di destinazione

- Windows Su Windows *MQ_INSTALLATION_PATH*`\bin`. Le applicazioni di esempio sono installate in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux Su Linux *MQ_INSTALLATION_PATH*`\lib64`. Gli esempi .NET si trovano in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

LTS Per IBM MQ 9.3.0 Long Term Support, la libreria `amqmdnetstd.dll` è disponibile nelle seguenti ubicazioni:

- Windows Su Windows *MQ_INSTALLATION_PATH*`\bin`. Le applicazioni di esempio sono installate in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux Su Linux *MQ_INSTALLATION_PATH*`/lib64 path`. Gli esempi .NET si trovano in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

Attenzione: V 9.3.1 Deprecated Da IBM MQ 9.3.1, le IBM MQ .NET librerie client create utilizzando .NET Standard 2.0 come framework di destinazione sono obsolete e le applicazioni che fanno riferimento a tali librerie generano un'avvertenza CS0618 durante il tempo di compilazione.

Stabilized LTS La libreria `amqmdnet.dll` per .NET Framework è ancora fornita, ma questa libreria è stabilizzata; in altre parole, non verranno introdotte nuove funzioni. Per le funzioni più recenti, è necessario migrare alla libreria `amqmdnetstd.dll` . Tuttavia, è possibile continuare a utilizzare la libreria `amqmdnet.dll` su IBM MQ 9.1 o versioni successive Long Term Support o Continuous Delivery .

V 9.3.1 Se un'applicazione .NET Framework viene compilata utilizzando `amqmdnetstd.dll` o `amqmxmsstd.dll` da una versione precedente a IBM MQ 9.3.1 e la stessa applicazione viene eseguita utilizzando librerie client basate su .NET 6 IBM MQ , il seguente tipo di eccezione `FileLoad` viene generato da .NET:

```
Eccezione rilevata: System.IO.FileLoadException: Impossibile caricare il file o l'assembly
'amqmdnetstd, Version =x.x.x.x, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' o
una delle sue dipendenze. La definizione manifest dell'assembly individuato non corrisponde al
riferimento dell'assieme. (Eccezione da HRESULT: 0x80131040)

Nome file: ' amqmdnetstd, Version =x.x.x.x, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
```

Per risolvere questo errore, le librerie presenti in *MQ_INSTALLATION_PATH*`/bin/netstandard2.0` devono essere copiate nella directory da cui è in esecuzione l'applicazione .NET Framework .

`dspmqver` comando

È possibile utilizzare il comando **`dspmqver`** per visualizzare le informazioni sulla versione e sulla creazione del componente .NET Core .

Confronto di funzioni tra IBM MQ classes for .NET Framework e IBM MQ classes for .NET (librerie.NET Standard e .NET 6)

La seguente tabella elenca le funzioni per IBM MQ classes for .NET Framework rispetto alle funzioni per IBM MQ classes for .NET (librerie.NET Standard e .NET 6).

Tabella 76. Differenze tra IBM MQ classes for .NET Framework e IBM MQ classes for .NET (librerie.NET Standard e .NET 6)

Funzione	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET (librerie.NET Standard e .NET 6)**
Nomi classe (API)	Tutte le classi rimangono le stesse in ogni rete.	Tutte le classi rimangono le stesse in ogni rete.
Sistema operativo	Windows	Windows Contenitori Docker Linux macOS
File `app.config` (file di configurazione per abilitare la traccia nel client ridistribuibile)	Il file `app.config` viene utilizzato per abilitare la traccia per il pacchetto ridistribuibile e per il client IBM MQ .NET autonomo. Consultare Traccia di un client IBM MQ classes for .NET Framework utilizzando un file di configurazione dell'applicazione per ulteriori informazioni sulle variabili utilizzate per la traccia, inclusi **`MQTRACEPATH`** e **`MQTRACELEVEL`**.	`app.config` non è supportata. Utilizzare le variabili di ambiente.

Tabella 76. Differenze tra IBM MQ classes for .NET Framework e IBM MQ classes for .NET (librerie.NET Standard e .NET 6) (Continua)

Funzione	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET (librerie.NET Standard e .NET 6)**
Traccia	Per un'installazione client completa di IBM MQ, è possibile utilizzare il comando **strmqtrc** per abilitare la traccia per IBM MQ classes for .NET Framework. Per i client ridistribuibili, il file `app.config` viene utilizzato anche per abilitare la traccia. Per ulteriori informazioni, consultare Traccia delle applicazioni IBM MQ .NET. V 9.3.3 Da IBM MQ 9.3.3, è possibile abilitare e disabilitare la traccia utilizzando il file `mqclient.ini` e impostando le proprietà appropriate della stanza Trace. È inoltre possibile abilitare e disabilitare la traccia in modo dinamico con il file `mqclient.ini` . Per ulteriori informazioni, consultare Traccia delle applicazioni IBM MQ .NET con mqclient.ini.	La variabile di ambiente **MQDOTNET_TRACE_ON** viene utilizzata per abilitare la funzione di traccia per i client ridistribuibili. I valori uguali e inferiori a 0 non abilitano la traccia. Il valore 1 abilita la traccia di livello predefinito. Un valore maggiore di 1, abilita la traccia dettagliata. L'impostazione di questa variabile di ambiente su qualsiasi altro valore come la stringa non abilita la traccia. Consultare Traccia delle applicazioni IBM MQ .NET utilizzando le variabili di ambiente. La variabile di ambiente **MQDOTNET_TRACE_ON** controlla se la directory di traccia IBM MQ è disponibile o meno. Se la directory di traccia è disponibile, il file di traccia viene generato nella directory di traccia. Tuttavia, se IBM MQ non è installato, il file di traccia viene copiato nella directory di lavoro corrente. Altre variabili di ambiente, inclusi **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**e così via, utilizzate per IBM MQ classes for .NET Framework, possono essere utilizzate e funzionano nello stesso modo. V 9.3.3 Da IBM MQ 9.3.3, è possibile abilitare e disabilitare la traccia utilizzando il file `mqclient.ini` e impostando le proprietà appropriate della stanza Trace. È inoltre possibile abilitare e disabilitare la traccia in modo dinamico con il file `mqclient.ini` . Per ulteriori informazioni, consultare Traccia delle applicazioni IBM MQ .NET con mqclient.ini.
Modalità di trasporto	Gestito, non gestito e bind	Gestita

Tabella 76. Differenze tra IBM MQ classes for .NET Framework e IBM MQ classes for .NET (librerie.NET Standard e .NET 6) (Continua)

Funzione	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (librerie.NET Standard e .NET 6)
TLS	Il keystore Windows viene utilizzato per memorizzare i certificati.	Windows Su Windows, il keystore deve essere utilizzato per memorizzare i certificati. I valori consentiti sono *USER o *SYSTEM. In base all'input, il client IBM MQ .NET guarda all'archivio chiavi Windows dell'utente corrente o a livello di sistema. Linux Su Linux, si consiglia di utilizzare la classe X509Store per installare i certificati e .NET Core installa i certificati nella seguente ubicazione: "`.dotnet/corefx/cryptography/x509stores`".
CCDT	Supportato	Supportato e le impostazioni del percorso CCDT sono le stesse delle classi .NET Framework.
Riconnessione automatica del client	Supportato	Supportato
Transazioni distribuite	Supportato	Non supportato
Installazione delle DLL (dynamic linked library) nella GAC (global assembly cache)	Le DLL sono installate nella GAC come parte dell'installazione di IBM MQ.	Le DLL non sono installate nella GAC come parte dell'installazione di IBM MQ.

Nota: Windows SID (security identifier) Windows :

L'autenticazione a livello di dominio non è supportata per IBM MQ classes for .NET (librerie.NET Standard e .NET 6). L'ID utente collegato viene utilizzato per l'autenticazione.

Sviluppo di applicazioni IBM MQ .NET Core su macOS

macOS

Le applicazioni IBM MQ .NET Core possono essere sviluppate su macOS.

Le librerie IBM MQ .NET non sono fornite con il toolkit macOS , pertanto è necessario copiarle da un client Windows o Linux IBM MQ in macOS. È quindi possibile utilizzare queste librerie per sviluppare applicazioni IBM MQ .NET Core su macOS.

Una volta sviluppate, queste applicazioni possono essere eseguite supportate in ambienti Windows o Linux .

Concetti correlati

“InstallazioneIBM MQ classes for .NET Framework” a pagina 563
IBM MQ classes for .NET Framework, inclusi gli esempi, sono installati con IBM MQ. Esiste un requisito di Microsoft.NET Framework su Windows.

“InstallazioneIBM MQ classes for XMS .NET” a pagina 620
IBM MQ classes for XMS .NET, inclusi gli esempi, sono installati con IBM MQ su Windows e Linux.

Linux Windows Download di IBM MQ classes for .NET dal repository NuGet

I IBM MQ classes for .NET sono disponibili per il download dal repository NuGet , in modo che possano essere facilmente utilizzati da .NET Developers.

Informazioni su questa attività

NuGet è il gestore pacchetti per piattaforme di sviluppo Microsoft incluso .NET. Gli strumenti del client NuGet forniscono la possibilità di produrre e utilizzare pacchetti. Un pacchetto NuGet è un singolo file compresso con l'estensione `.nupkg` che contiene il codice compilato (DLL), altri file correlati a quel codice e un manifest descrittivo che include informazioni come il numero di versione del pacchetto.

È possibile scaricare il package `IBMMQDotnetClient` NuGet , che contiene la libreria `amqmdnetstd.dll` , dalla galleria NuGet , che è il repository di package centrale utilizzato da tutti gli autori e i consumer di package.

Nota: V 9.3.1 Da IBM MQ 9.3.1, il pacchetto NuGet contiene le librerie create utilizzando .NET Standard 2.0 e .NET 6 come framework di destinazione. Le librerie per .NET Standard 2.0 sono disponibili nella cartella `netstandard2.0` e le librerie per .NET 6 sono disponibili nella cartella `net6.0` .

Esistono tre metodi per scaricare il pacchetto `IBMMQDotnetClient` :

- Utilizzando Microsoft Visual Studio. NuGet è distribuito come estensione Microsoft Visual Studio . Da Microsoft Visual Studio 2012, NuGet è preinstallato per default.
- Dalla riga di comando utilizzando NuGet Package Manager o la CLI .NET .
- Utilizzando un browser Web.

Come per il pacchetto ridistribuibile, si abilita la traccia utilizzando la variabile di ambiente **MQDOTNET_TRACE_ON**.

Procedura

- Per scaricare il pacchetto `IBMMQDotnetClient` utilizzando la IU di Package Manager in Microsoft Visual Studio, completa la seguente procedura:
  a) Fare clic con il tasto destro del mouse su un progetto .NET e fare clic su **Gestisci package di nuget**.
  b) Fare clic sulla scheda **Sfoglia** e cercare `"IBMMQDotnetClient"`.
  c) Selezionare il package e fare clic su **Installa**.

  Durante l'installazione, Package Manager fornisce le informazioni sull'avanzamento sotto forma di istruzioni della console.
- Per scaricare il pacchetto `IBMMQDotnetClient` dalla riga di comando, scegliere una delle seguenti opzioni:
  - Utilizzando NuGet Package Manager, immettere il seguente comando:

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    Durante l'installazione, Package Manager fornisce le informazioni sull'avanzamento sotto forma di istruzioni della console. È possibile reindirizzare l'emissione a un file di log.
  - Utilizzando la CLI .NET , immetti il seguente comando:

    ```
    dotnet add package IBMMQDotnetClient --version 9.1.4
    ```
- Utilizzando un browser Web, scaricare il package `IBMMQDotnetClient` da https://www.nuget.org/packages/IBMMQDotnetClient.

Concetti correlati

Informazioni sulla licenza di IBM MQ client per .NET

Attività correlate

“Download di IBM MQ classes for XMS .NET dal repository NuGet” a pagina 623
I IBM MQ classes for XMS .NET sono disponibili per il download dal repository NuGet , in modo che possano essere facilmente utilizzati dagli sviluppatori .NET .

Windows InstallazioneIBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework, inclusi gli esempi, sono installati con IBM MQ. Esiste un requisito di Microsoft.NET Framework su Windows.

L'ultima versione di IBM MQ classes for .NET Framework viene installata per impostazione predefinita come parte dell'installazione standard di IBM MQ nella funzione *Servizi Web e messaggistica Java e .NET* . Per le istruzioni di installazione, consultare Installazione del server IBM MQ su Windows o Installazione di un client IBM MQ su sistemi Windows.

V 9.3.0 V 9.3.0 Da IBM MQ 9.3.0, per eseguire IBM MQ classes for .NET Framework , è necessario installare Microsoft.NET Framework V4.7.2 o versioni successive. Questa è una modifica da IBM MQ 9.2 dove la versione minima richiesta era V4.6.2.

V 9.3.0 V 9.3.0 Le applicazioni esistenti compilate con Microsoft.NET Framework V3.5 possono essere eseguite senza ricompilare aggiungendo la seguente tag nel file `app.config` dell'applicazione:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

Nota: Se Microsoft .NET Framework V4.7.2 o versioni successive non viene installato prima di installare IBM MQ, l'installazione del prodotto IBM MQ continua senza errori, ma IBM MQ classes for .NET non è disponibile. Se.NET Framework viene installato dopo l'installazione di IBM MQ, gli assemblaggi IBM MQ.NET devono essere registrati eseguendo lo script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , dove *WMQInstallDir* è la directory in cui è installato IBM MQ . Questo script installa gli assembly richiesti nella GAC (Global Assembly Cache). Una serie di file `amqi*.log` che registrano le azioni eseguite vengono creati nella directory %TEMP% . Non è necessario eseguire nuovamente lo script `amqiRegisterdotNet.cmd` se .NET viene aggiornato a V4.7.2 o versione successiva da una versione precedente, ad esempio da .NET V3.5.

In un ambiente di installazione multipla, se è stato precedentemente installato IBM MQ classes for .NET come pacchetto di supporto, non è possibile installare IBM MQ a meno che non si disinstalli prima il pacchetto di supporto. La funzione IBM MQ classes for .NET installata con IBM MQ contiene le stesse funzioni del pacchetto di supporto.

Vengono fornite anche applicazioni di esempio, inclusi i file di origine; consultare “Applicazioni di esempio per .NET” a pagina 564.

Per informazioni sull'utilizzo del canale personalizzato IBM MQ per Microsoft WCF con .NET, consultare “Sviluppo di applicazioni Microsoft Windows Communication Foundation con IBM MQ” a pagina 1270

Concetti correlati

“InstallazioneIBM MQ classes for .NET” a pagina 557
IBM MQ classes for .NET, inclusi gli esempi, sono stati installati con IBM MQ su Windows e Linux

Attività correlate

Traccia delle applicazioni IBM MQ .NET

Opzioni per la connessione di IBM MQ classes for .NET a un gestore code

Esistono tre modalità di connessione di IBM MQ classes for .NET a un gestore code. Considerare il tipo di connessione più adatto alle proprie esigenze.

Connessione bind client

Per utilizzare IBM MQ classes for .NET come IBM MQ MQI client, è possibile installarlo, con IBM MQ MQI client, sulla macchina server IBM MQ o su una macchina separata. Una connessione di bind client può utilizzare transazioni XA o non XA

Connessione bind server

Quando viene utilizzato in modalità bind del server, IBM MQ classes for .NET utilizza l'API del gestore code, piuttosto che comunicare attraverso una rete. Ciò fornisce prestazioni migliori per applicazioni IBM MQ rispetto all'uso di connessioni di rete.

Per utilizzare la connessione dei collegamenti, è necessario installare IBM MQ classes for .NET sul server IBM MQ .

Connessione client gestito

Una connessione effettuata in questa modalità si connette come un client IBM MQ a un server IBM MQ in esecuzione sulla macchina locale o remota.

La connessione IBM MQ classes for .NET in questa modalità rimane nel codice gestito .NET e non effettua chiamate ai servizi nativi. Per ulteriori informazioni sul codice gestito, fare riferimento alla documentazione di Microsoft .

Esistono diverse limitazioni all'utilizzo del client gestito. Per ulteriori informazioni, consultare “Connessioni client gestite” a pagina 580.

Applicazioni di esempio per .NET

Per eseguire le proprie applicazioni .NET , utilizzare le istruzioni per i programmi di verifica, sostituendo il proprio nome applicazione al posto delle applicazioni di esempio.

Vengono fornite le seguenti applicazioni di esempio:

- Un'applicazione di inserimento messaggi
- Un'applicazione di richiamo messaggi
- Un'applicazione 'hello world'
- Un'applicazione di pubblicazione / sottoscrizione
- Un'applicazione che utilizza le proprietà del messaggio

Tutte queste applicazioni di esempio sono fornite nel linguaggio C# e alcune sono fornite anche in C++ e Visual Basic. È possibile scrivere le applicazioni in qualsiasi lingua supportata da .NET.

Programma "Put message" SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)
Questo programma mostra come inserire un messaggio in una coda denominata. Il programma ha tre parametri:

- Il nome di una coda (obbligatorio), ad esempio SYSTEM.DEFAULT.LOCAL.QUEUE
- Il nome di un gestore code (facoltativo)
- La definizione di un canale (facoltativo), ad esempio, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Se non viene fornito alcun nome gestore code, il gestore code assume il valore predefinito del gestore code locale predefinito. Se un canale è definito, ha lo stesso formato della variabile di ambiente MQSERVER.

Programma "Get message" SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
Questo programma mostra come richiamare un messaggio da una coda denominata. Il programma ha tre parametri:

- Il nome di una coda (obbligatorio), ad esempio SYSTEM.DEFAULT.LOCAL.QUEUE
- Il nome di un gestore code (facoltativo)

- La definizione di un canale (facoltativo), ad esempio, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Se non viene fornito alcun nome gestore code, il gestore code assume il valore predefinito del gestore code locale predefinito. Se un canale è definito, ha lo stesso formato della variabile di ambiente MQSERVER.

Programma "Hello World" (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
Questo programma mostra come inserire e richiamare un messaggio. Il programma ha tre parametri:

- Il nome di una coda (facoltativo), ad esempio, SYSTEM.DEFAULT.LOCAL.QUEUE o SYSTEM.DEFAULT.MODEL.QUEUE
- Il nome di un gestore code (facoltativo)
- Una definizione di canale (facoltativa), ad esempio SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Se non viene fornito alcun nome coda, il nome assume il valore predefinito SYSTEM.DEFAULT.LOCAL.QUEUE. Se non viene fornito alcun nome gestore code, il gestore code assume il valore predefinito del gestore code locale predefinito.

Programma "Publish/subscribe" (MQPubSubSample.cs)
Questo programma mostra come utilizzare la pubblicazione / sottoscrizione IBM MQ . Viene fornito solo in C#. Il programma ha due parametri:

- Il nome di un gestore code (facoltativo)
- Una definizione di canale (facoltativo)

Programma "Message properties" (MQMessagePropertiesSample.cs)
Questo programma mostra come utilizzare le proprietà del messaggio. Viene fornito solo in C#. Il programma ha due parametri:

- Il nome di un gestore code (facoltativo)
- Una definizione di canale (facoltativo)

È possibile verificare l'installazione compilando ed eseguendo queste applicazioni.

Percorsi di installazione

Le applicazioni di esempio vengono installate nelle seguenti ubicazioni, in base alla lingua in cui sono scritte. *MQ_INSTALLATION_PATH* rappresenta la directory di livello superiore in cui è installato IBM MQ .

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

C++ gestito

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

Creazione delle applicazioni di esempio

Per creare le applicazioni di esempio, viene fornito un file batch per ogni lingua.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

Il file bldcssamp.bat contiene una riga per ciascun esempio, che è tutto ciò che è necessario per creare questo programma di esempio:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

C++ gestito

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

Il file bldmcpsamp.bat contiene una riga per ciascun esempio, che è tutto ciò che è necessario per creare questo programma di esempio:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Se si desidera compilare queste applicazioni su Microsoft Visual Studio 2003 /.NET SDKv1.1, sostituire il comando di compilazione:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

indicando

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

Il file bldvbsamp.bat contiene una riga per ciascun esempio, che è tutto ciò che è necessario per creare questo programma di esempio:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Esempi per l'utilizzo di IBM MQ con Microsoft .NET Core

Da IBM MQ 9.2.0, IBM MQ supporta .NET Core per applicazioni IBM MQ .NET in ambienti Windows . IBM MQ classes for .NET Standard, inclusi gli esempi, vengono installati per impostazione predefinita come parte dell'installazione standard di IBM MQ .

Le applicazioni di esempio per IBM MQ .NET sono installate in `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`. Viene inoltre fornito uno script, che può essere utilizzato per compilare gli esempi.

È possibile creare gli esempi utilizzando i file `build.bat` forniti. Esiste un `build.bat` per ogni esempio nella seguente posizione su Windows:

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux Da IBM MQ 9.2.0, IBM MQ supporta anche Core per applicazioni in ambienti Linux .

Per ulteriori informazioni sull'utilizzo di IBM MQ con Microsoft .NET Core, consultare “InstallazioneIBM MQ classes for .NET” a pagina 557.

Configurazione del gestore code per l'accettazione delle connessioni client TCP/IP

Configurare un gestore code per accettare le richieste di connessione in entrata dai client.

Informazioni su questa attività

Questa attività illustra i passi di base per configurare un gestore code per accettare le connessioni client TCP/IP. Per un sistema di produzione, è necessario considerare anche le implicazioni di sicurezza durante la configurazione dei gestori code.

Procedura

1. Definire un canale di connessione server:
   a. Avviare il gestore code.
   b. Definire un canale di esempio denominato NET.CHANNEL:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   Importante: Questo esempio è solo per l'utilizzo in un ambiente sandbox, in quanto non include alcuna considerazione delle implicazioni di sicurezza. Per un sistema di produzione, prendere in considerazione l'utilizzo di TLS o di un'uscita di sicurezza. Per ulteriori informazioni, consultare Sicurezza IBM MQ.
2. Avviare un listener:

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   Nota: Le parentesi quadre indicano parametri facoltativi; *qmname* non è richiesto per il gestore code predefinito e il numero di porta *portnum* non è richiesto se si utilizza il valore predefinito (1414).

Transazioni distribuite in .NET

Le transazioni distribuite o le transazioni globali consentono alle applicazioni client di includere diverse fonti di dati su due o più sistemi di rete in una transazione.

Nelle transazioni distribuite, un gestore transazioni coordina e gestisce la transazione tra due o più gestori risorse.

Le transazioni possono essere un processo di commit a fase singola o a due fasi. Il commit a fase singola è un processo in cui solo un gestore risorse partecipa alla transazione e il processo di commit a due fasi è in cui più di un gestore risorse partecipa alla transazione. Nel processo di commit a due fasi, il gestore transazioni invia una chiamata di preparazione per verificare se tutti i gestori risorse sono pronti per il commit. Quando riceve il riconoscimento da tutti i gestori risorse, viene emessa la chiamata di commit. Altrimenti, si verifica un rollback sull'intera transazione. Consultare Gestione delle transazioni e supporto per ulteriori dettagli. I gestori delle risorse devono informare i gestori delle transazioni della partecipazione alla transazione. Quando il gestore risorse informa il gestore transazioni della sua partecipazione, il gestore risorse riceve i callback dal gestore transazioni quando la transazione sta per eseguire il commit o il rollback.

Le classi IBM MQ .NET già supportano le transazioni distribuite in connessioni in modalità bind server e non gestite. In queste modalità, le classi IBM MQ .NET delegano tutte le chiamate al client di transazione esteso C, che gestisce l'elaborazione della transazione per conto di .NET.

Le classi IBM MQ.NET ora supportano le transazioni distribuite in modalità gestita dove le classi IBM MQ .NET utilizzano lo spazio dei nomi System.Transactions per il supporto delle transazioni distribuite. L'infrastruttura System.Transactions rende la programmazione transazionale semplice ed efficace, supportando le transazioni avviate in tutti i gestori risorse incluso IBM MQ. L'applicazione IBM MQ .NET può inserire e richiamare i messaggi utilizzando .NET il modello di programmazione delle transazioni implicite o esplicite. Nelle transazioni implicite, i limiti della transazione vengono creati dal programma applicativo che decide quando eseguire il commit, il rollback (per le transazioni esplicite) o completare la transazione. Nelle transazioni esplicite, è necessario specificare esplicitamente se si desidera eseguire il commit, il rollback e il completamento della transazione.

IBM MQ.NET utilizza Microsoft distributed transaction coordinator (MS DTC) come gestore transazioni, che coordina e gestisce la transazione tra più gestori risorse. IBM MQ viene utilizzato come gestore risorse. Notare che non è possibile utilizzare TLS con le transazioni XA. È necessario utilizzare CCDT. Per ulteriori informazioni, consultare Utilizzo del client transazionale esteso con canali TLS.

IBM MQ.NET segue il modello DTP (Distributed Transaction Processing) X/Open. Il modello X/Open Distributed Transaction Processing è un modello di elaborazione delle transazioni distribuite proposto da Open Group, un consorzio di fornitori. Questo modello è uno standard tra la maggior parte dei fornitori commerciali nell'elaborazione delle transazioni e nei domini di database. La maggior parte dei prodotti di gestione delle transazioni commerciali supporta il modello X/DTP.

Modalità di transazione

- “Transazioni distribuite in modalità gestita .NET” a pagina 569
- Transazioni distribuite per modalità non gestita

Coordinamento delle transazioni in vari scenari

- Una connessione potrebbe partecipare a diverse transazioni, ma solo una transazione è attiva in qualsiasi momento.
- Durante una transazione, MQQueueManager.La chiamata di disconnessione viene rispettata. In questo caso, viene richiesto il rollback della transazione.
- Durante una transazione, viene rispettata la chiamata MQQueue.Close o MQTopic.Close . In questo caso viene richiesto il rollback della transazione.
- I limiti di transazione vengono creati dal programma applicativo che decide quando eseguire il commit, il rollback (per le transazioni esplicite) o il completamento (per le transazioni implicite) della transazione.
- Se l'applicazione client si interrompe durante una transazione con un errore non previsto prima di emettere una chiamata Put o Get su una chiamata della coda o dell'argomento, viene eseguito il rollback della transazione e viene generata una MQException.
- Se il codice motivo MQCC_FAILED viene restituito durante una chiamata Put o Get su una coda o su una chiamata Topic, viene generata un'eccezione MQException con codice motivo e viene eseguito il rollback della transazione. Se una chiamata di preparazione è già stata emessa dal gestore transazioni, IBM MQ .NET restituisce la richiesta di preparazione eseguendo il rollback forzato della transazione. Quindi, il gestore transazioni DTC causa un rollback sul lavoro corrente con tutti i gestori risorse nelle transazioni dell'ambiente correnti.
- Durante una transazione che coinvolge più gestori risorse se alcuni motivi ambientali causano il blocco indefinito della chiamata Put o Get, il gestore transazioni attende fino a un tempo stabilito. Dopo che il tempo è terminato, provoca il rollback di tutte le operazioni correnti con tutti i gestori risorse nelle transazioni dell'ambiente corrente. Se questa attesa indefinita si verifica durante la fase di preparazione, il gestore transazioni potrebbe scadere o emettere una chiamata in dubbio sulla risorsa, nel cui caso viene eseguito il rollback della transazione.
- Le applicazioni che utilizzano le transazioni devono inserire o richiamare i messaggi in SYNC_POINT. Se una chiamata Put o Get del messaggio viene emessa in un contesto transazionale che non si trova in SYNC_POINT, la chiamata ha esito negativo con il codice motivo MQRC_UNIT_OF_WORK_NOT_STARTED.

Differenze comportamentali tra il supporto delle transazioni del client gestito e non gestito utilizzando lo spazio dei nomi Microsoft.NET System.Transactions

Le transazioni nidificate hanno un TransactionScope all'interno di un altro TransactionScope

- Il client completamente gestito IBM MQ .NET supporta TransactionScope nidificato
- IBM MQ Il client non gestito .NET non supporta TransactionScope nidificato

Transazioni dipendenti da System.Transactions

- Il client completamente gestito IBM MQ .NET supporta la funzione di transazioni dipendenti fornita da System.Transactions.
- Il client non gestito IBM MQ .NET non supporta la funzione di transazioni dipendenti fornita da System.Transactions.

Esempi di prodotti

Gli esempi di prodotto `SimpleXAPute` `SimpleXAGet` sono disponibili in `WebSphere MQ\tools\dotnet\samples\cs\base`. Gli esempi sono applicazioni C#, che dimostrano l'uso di MQPUT e MQGET in Transazioni distribuite utilizzando lo spazio nomi SystemTransactions . Per ulteriori informazioni su questi esempi, consultare “Creazione di semplici messaggi put e get in un TransactionScope” a pagina 572.

Transazioni distribuite in modalità gestita .NET

Le classi IBM MQ .NET utilizzano lo spazio dei nomi System.Transactions per il supporto delle transazioni distribuite in modalità gestita. Nella modalità gestita, MS DTC coordina e gestisce le transazioni distribuite su tutti i server elencati in una transazione.

Le classi IBM MQ .NET forniscono un modello di programmazione esplicito basato sulla classe System.Transactions.Transaction e un modello di programmazione implicito che utilizza la classe System.Transactions.TransactionScope, in cui le transazioni vengono gestite automaticamente dall'infrastruttura.

Transazione implicita

La seguente parte di codice descrive come un'applicazione IBM MQ .NET inserisce un messaggio utilizzando la programmazione di transazioni implicite .NET .

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

Spiegazione del flusso di codice della transazione implicita

Il codice crea *TransactionScope* e inserisce il messaggio nell'ambito. Richiama quindi *Completa* per informare il coordinatore della transazione del completamento della transazione. Il coordinatore della transazione ora emette *prepare* e *commit* per completare la transazione. Se viene rilevato un problema, viene richiamato un *rollback* .

Transazione esplicita

Il seguente codice descrive il modo in cui un'applicazione IBM MQ .NET inserisce i messaggi utilizzando il modello di programmazione della transazione esplicito .NET .

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
```

```
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

Spiegazione del flusso di codice della transazione esplicita

La parte di codice crea la transazione utilizzando la classe *CommitableTransaction* . Inserisce un messaggio in tale ambito e richiama esplicitamente *commit* per completare la transazione. In caso di problemi, viene richiamato il comando *rollback* .

Transazioni distribuite in modalità non gestita .NET

Le classi IBM MQ.NET supportano le connessioni non gestite (client) utilizzando il client della transazione estesa e COM + /MTS come coordinatore della transazione, utilizzando il modello di programmazione della transazione implicito o esplicito. In modalità non gestita, le classi di IBM MQ .NET delegano tutte le relative chiamate al client di transazioni estese C che gestisce l'elaborazione della transazione per conto di .NET.

L'elaborazione della transazione è controllata da un gestore transazioni esterno, che coordina l'unità di lavoro globale sotto il controllo dell'API del gestore transazioni. I verbi MQBEGIN, MQCMIT e MQBACK non sono disponibili. IBM MQ Le classi .NET espongono questo supporto tramite la modalità di trasporto non gestita (client C). Consultare Configurazione di gestori transazioni compatibili con XA

MTS si è evoluto come sistema TP (transaction processing) per fornire le stesse funzioni su Windows NT disponibili in CICS, Tuxedo e su altre piattaforme. Quando MTS è installato, un servizio separato viene aggiunto a Windows NT chiamato Microsoft Distributed Transaction Coordinator (MSDTC). MSDTC coordina le transazioni che si estendono a archivi dati o risorse separati. Per funzionare, è necessario che ogni archivio dati implementi il proprio gestore risorse proprietario.

IBM MQ diventa compatibile con MSDTC implementando un'interfaccia (interfaccia del gestore risorse proprietarie) in cui riesce ad associare le chiamate XA DTC alle chiamate IBM MQ(X/Open). IBM MQ svolge il ruolo di gestore risorse.

Quando un componente come COM + richiede l'accesso a un IBM MQ, COM di solito verifica con l'oggetto di contesto MTS appropriato se è richiesta una transazione. Se è richiesta una transazione, il COM informa il DTC e avvia automaticamente una transazione IBM MQ integrale per questa operazione. Quindi il COM lavora con i dati attraverso il software MQMTS, inserendo e ottenendo i messaggi come richiesto. L'istanza dell'oggetto ottenuta da COM richiama il metodo SetComplete o SetAbort una volta che tutte le azioni sui dati sono state completate. Quando l'applicazione emette SetComplete, la chiamata segnala al DTC che l'applicazione ha completato la transazione e il DTC può procedere con il processo di commit a due fasi. Il DTC emette quindi chiamate a MQMTS che a loro volta emettono chiamate a IBM MQ per eseguire il commit o il rollback della transazione.

Scrittura di una applicazione IBM MQ .NET utilizzando un client non gestito

Per essere eseguita nel contesto di COM +, una classe .NET deve ereditare da System.EnterpriseServices.ServicedComponent. Le regole e i consigli per creare assiemi che utilizzano componenti assistiti sono i seguenti:

Nota: Le seguenti operazioni sono rilevanti solo se si utilizza la modalità System.EnterpriseServices .

- La classe e il metodo avviati in COM + devono essere entrambi pubblici (nessuna classe interna e nessun metodo protetto o statico).
- Gli attributi della classe e del metodo: l'attributo TransactionOption indica il livello di transazione della classe, ovvero se le transazioni sono disabilitate, supportate o richieste. L'attributo AutoComplete nel

metodo ExecuteUOW() indica a COM + di eseguire il commit della transazione se non viene generata alcuna eccezione non gestita.

- Forte - denominazione di un assieme: l'assieme deve avere un nome forte e deve essere registrato nella GAC (Global Assembly Cache). L'assemblea è registrata in COM + esplicitamente o per registrazione pigra dopo che è stata registrata nel GAC.
- Registrazione di un assembly in COM +: preparare l'assembly per essere esposto ai client COM. Quindi, creare una libreria di tipi utilizzando lo strumento di registrazione Assembly, regasm.exe.

```
regasm UnmanagedToManagedXa.dll
```

- Registrare l'assieme in GAC `gacutil /i UnmanagedToManagedXa.dll`.
- Registrare l'assemblaggio in COM + utilizzando lo strumento del programma di installazione dei servizi .NET , regsvcs.exe. Consultare il tipo di libreria creata da regasm.exe:

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- L'assemblaggio viene distribuito nel GAC, e successivamente viene registrato in COM + tramite registrazione lazy. Il framework .NET si occupa della registrazione dopo che il codice viene eseguito per la prima volta.

Il flusso di codice di esempio utilizzando il modello System.EnterpriseServices e System.Transactions con COM + sono descritti nelle seguenti sezioni:

Esempio di flusso di codice utilizzando il modello System.EnterpriseServices

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

Esempio di flusso di codice che utilizza System.Transactions per interazioni con COM +

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

Creazione di semplici messaggi put e get in un TransactionScope

Le applicazioni C# di esempio del prodotto sono disponibili in IBM MQ. Queste semplici applicazioni dimostrano l'inserimento e il richiamo di messaggi in un TransactionScope. Alla fine dell'attività, sarà possibile inserire e richiamare i messaggi da una coda o da un argomento.

Prima di iniziare

Il servizio MSDTC deve essere in esecuzione e abilitato per le Transazioni XA.

Informazioni su questa attività

L'esempio è una semplice applicazione, `SimpleXAPut` e `SimpleXAGet`. I programmi `SimpleXAPut` e `SimpleXAGet` sono applicazioni C# disponibili in IBM MQ. `SimpleXAPut` dimostra l'utilizzo di MQPUT, in Transazioni distribuite utilizzando lo spazio dei nomi SystemTransactions . `SimpleXAGet` dimostra di utilizzare MQGET, in Transazioni distribuite utilizzando lo spazio dei nomi SystemTransactions .

`SimpleXAPut` si trova in `MQ\tools\dotnet\samples\cs\base`

Procedura

Le applicazioni possono essere eseguite con i parametri della riga comandi da `tools\dotnet\samples\cs\base\bin`

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

dove i parametri sono:

`-destinationURI`

Può essere una coda o un argomento. Per una coda, specificare `queue://queueName` e per un argomento specificare `topic://topicName`.

-host

Può essere un nome host come localhost o un indirizzo IP.

-port

La porta su cui è in esecuzione il gestore code.

-channel

Il canale di connessione utilizzato. Il valore predefinito è `SYSTEM.DEF.SVRCONN`

-transaction

Il risultato della transazione, ad esempio commit o rollback.

-mode

La modalità di trasporto, ad esempio gestita o non gestita.

-numberOfMsgs

Il numero di messaggi. Il valore predefinito è 1.

Esempio

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

Ripristino delle transazioni in IBM MQ .NET

Questa sezione descrive il processo di recupero delle transazioni in IBM MQ .NET XA utilizzando la modalità gestita.

Informazioni su questa attività

Nell'elaborazione di transazioni distribuite, le transazioni possono essere completate correttamente, ma possono verificarsi scenari in cui una transazione potrebbe avere esito negativo per molti motivi. Questi motivi potrebbero includere un errore di sistema, un errore hardware, un errore di rete, dati non corretti o non validi, errori dell'applicazione o disastri naturali o causati dall'uomo. Non è possibile impedire errori di transazione. Il sistema di transazione distribuito deve essere in grado di gestire questi errori. Deve essere in grado di rilevare e correggere gli errori quando si verificano. Questo processo è noto come ripristino della transazione.

Un aspetto importante di DTP (Distributed Transaction Processing) è il recupero delle transazioni incomplete o in dubbio. È essenziale eseguire il ripristino poiché la parte Unità di lavoro di una particolare transazione viene bloccata fino a quando non viene ripristinata. Microsoft.NET dalla libreria della classe System.Transactions fornisce l'opzione per il ripristino di transazioni incomplete / in dubbio. Questo supporto di ripristino prevede che il Resource Manager gestisca i log di transazione ed esegua il ripristino quando necessario.

Nel modello di ripristino delle transazioni di Microsoft .NET , il Gestore transazioni (System.Transactionso Microsoft Distributed Transaction coordinator (MS DTC) o entrambi), avvia, coordina e controlla il ripristino delle transazioni. I gestori risorse basati sul protocollo OLE Tx (il protocollo Microsoft XA) forniscono le opzioni per configurare il DTC per guidare, coordinare e controllare il ripristino. A tale scopo, i gestori risorse devono registrare XA_Switch con MS DTC utilizzando l'interfaccia nativa.

XA_Switch fornisce i punti di ingresso delle funzioni XA come xa_start, xa_end e xa_recover nel Resource Manager al Distributed Transaction Coordinator.

Ripristino mediante DTC (Distributed Transaction coordinator) Microsoft :

Microsoft Distributed Transaction Coordinator fornisce due tipi di processi di recupero.

Recupero a freddo

Il ripristino a freddo viene eseguito se il processo del gestore transazioni ha esito negativo mentre è aperta una connessione a un gestore risorse XA. Quando il gestore transazioni viene riavviato, legge i log del gestore transazioni e ristabilisce la connessione al gestore risorse XA, quindi avvia il ripristino.

Ripristino a caldo

Il ripristino a caldo viene eseguito se il gestore transazioni rimane attivo mentre la connessione tra il gestore transazioni e il gestore risorse XA ha esito negativo perché il gestore risorse XA o la rete hanno esito negativo. Dopo l'errore, il gestore transazioni tenta periodicamente di riconnettersi al gestore risorse XA. Quando la connessione viene ristabilita, il gestore transazioni avvia il ripristino XA.

Lo spazio dei nomi System.Transactions fornisce l'implementazione gestita delle transazioni distribuite basate su MS DTC come gestore transazioni. Fornisce funzioni simili a quelle dell'interfaccia nativa di MS DTC, ma in un ambiente completamente gestito. L'unica differenza riguarda il ripristino della transazione. System.Transactions si aspetta che i gestori risorse guidino il recupero da soli e si coordinino con i gestori transazioni (MS DTC). Il Resource Manager deve richiedere il recupero di una particolare transazione incompleta e quindi il Gestore transazioni la accetta e la coordina in base al risultato effettivo di quella particolare transazione.

Processo di recupero della transazione per IBM MQ .NET

Questa sezione descrive come è possibile ripristinare le transazioni distribuite con le classi IBM MQ .NET .

Panoramica

Per ripristinare una transazione incompleta, sono richieste le informazioni di ripristino. Le informazioni di recupero della transazione devono essere registrate nella memorizzazione dai gestori risorse. IBM MQ Le classi .NET seguono un percorso simile. Le informazioni di ripristino della transazione vengono registrate in una coda di sistema denominata SYSTEM.DOTNET.XARECOVERY.QUEUE.

Il ripristino della transazione in IBM MQ .NET è un processo in due fasi:

1. Registrazione delle informazioni di ripristino della transazione in SYSTEM.DOTNET.XARECOVERY.QUEUE.
2. Ripristino delle transazioni utilizzando l'applicazione XA Monitor WmqDotnetXAMonitor.

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM.DOTNET.XARECOVERY.QUEUE è una coda di sistema che contiene le informazioni di recupero delle transazioni incomplete. Questa coda viene creata quando viene creato un gestore code.

Per ogni transazione, durante la fase di preparazione viene aggiunto a SYSTEM.DOTNET.XARECOVERY.QUEUE. Il messaggio viene eliminato se la chiamata di commit ha esito positivo.

Nota: Non eliminare SYSTEM.DOTNET.XARECOVERY.QUEUE .

Applicazione WMQDotnetXAMonitor

IBM MQ .NET L'applicazione XA Monitor, WmqDotnetXAMonitor, è un'applicazione gestita .NET che monitora un gestore code ed elabora i messaggi in SYSTEM.DOTNET.XARECOVERY.QUEUE e recupera le transazioni incomplete

Se l'agent MCA (Message Channel Agent) non è in grado di inserire il messaggio nella coda di destinazione, genera un report di eccezioni contenente il messaggio originale e lo inserisce in una coda di trasmissione da inviare alla coda di risposta specificata nel messaggio originale. (Se la coda di risposta si trova sullo stesso gestore code dell'MCA, il messaggio viene inserito direttamente in quella coda, non in una coda di trasmissione.)

Le seguenti operazioni sono considerate incomplete e vengono recuperate:

- Se la transazione è preparata ma COMMIT non è stato completato entro il periodo di timeout.
- Se la transazione è preparata ma il gestore code IBM MQ è stato disattivo.
- Se la transazione è preparata, ma il gestore transazioni è stato disattivo.

  L'applicazione XA Monitor deve essere eseguita dallo stesso sistema su cui è in esecuzione l'applicazione client IBM MQ .NET . Se sono presenti applicazioni in esecuzione su più sistemi e che si collegano allo stesso gestore code, l'applicazione XAMonitor WmqDotnetdeve essere eseguita da tutti i sistemi. Sebbene ogni macchina client disponga di un'istanza dell'applicazione XA Monitor in esecuzione per ripristinare l'applicazione, ogni istanza XA Monitor dovrebbe essere in grado di identificare il messaggio che corrisponde alla transazione che l'attuale MS DTC locale di XA Monitor stava coordinando in modo da poterla rielencare e completare.

Concetti correlati

“Casi di utilizzo del ripristino della transazione per IBM MQ .NET” a pagina 575
Esistono diversi casi di utilizzo da cui potrebbe essere necessario ripristinare le transazioni.

Attività correlate

“Utilizzo dell'applicazione WMQDotnetXAMonitor” a pagina 576
Il client IBM MQ .NET fornisce un'applicazione XA Monitor, WmqDotnetXAMonitor, che è possibile utilizzare per ripristinare tutte le transazioni distribuite incomplete. L'applicazione XAMonitor WmqDotnetstabilisce una connessione al gestore code in cui le transazioni sono in dubbio e quindi risolve la transazione in base ai parametri impostati.

Casi di utilizzo del ripristino della transazione per IBM MQ .NET

Esistono diversi casi di utilizzo da cui potrebbe essere necessario ripristinare le transazioni.

- **IBM MQ Applicazione che utilizza un singolo DTC e una singola istanza del gestore code:** in questo caso d'uso, quando ci si connette al gestore code ed si esegue l'unità di lavoro (UoW) nella transazione, e se la transazione non riesce e diventa incompleta, l'applicazione XA Monitor recupera la transazione e la completa.

  In questo caso d'uso, ci sarà una singola istanza dell'applicazione XA Monitor in esecuzione, poiché un singolo gestore code è associato alle transazioni.
- **Più applicazioni IBM MQ che utilizzano un singolo DTC e una singola istanza del gestore code:** in questo caso di utilizzo, esistono più applicazioni IBM MQ in un singolo DTC e tutte si collegano allo stesso gestore code ed eseguono UoW nelle transazioni.

  Se le transazioni non riescono e diventano incomplete, l'applicazione XA Monitor le recupera e completa le transazioni relative a tutte le applicazioni.

  In questo caso, viene eseguita una singola istanza dell'applicazione XA Monitor, poiché nelle transazioni viene utilizzato un gestore code.
- **Più applicazioni IBM MQ , più DTC, diverse istanze del gestore code:** in questo caso di utilizzo, esistono più di un'applicazione IBM MQ in diversi DTC (ossia, ogni applicazione è in esecuzione su una macchina differente) e si connette a gestori code differenti.

  Se si verifica un errore e la transazione diventa incompleta, l'applicazione di monitoraggio controlla il TransactionManagerWhereabouts nel messaggio per determinare l'indirizzo DTC. Se il valore di TransactionManagerWhereabouts corrisponde all'indirizzo DTC sotto il quale il monitoraggio è in esecuzione, completa il recupero, altrimenti continua la ricerca fino a quando non viene trovato il messaggio corrispondente al suo DTC.

  In questo caso d'uso, ci sarà solo un'istanza dell'applicazione XA Monitor in esecuzione per client (utente o computer) poiché ogni client ha il proprio gestore code utilizzato nelle transazioni.
- **Più applicazioni IBM MQ , più DTC, più istanze dello stesso gestore code:** in questo caso d'uso, esistono più di un'applicazione IBM MQ in diversi DTC (ossia ogni applicazione è in esecuzione su una macchina diversa) e tutti si collegano allo stesso gestore code.

Se si verifica un errore e la transazione diventa incompleta, l'applicazione di monitoraggio verifica i messaggi TransactionManagerWhereabouts nel messaggio per controllare se l'indirizzo e il valore DTC corrispondono al DTC in cui è in esecuzione il controllo. Se entrambi i valori corrispondono, completa il ripristino, altrimenti continua la ricerca fino a quando non trova il messaggio corrispondente al suo DTC.

In questo caso di utilizzo, ci sarà una sola istanza dell'applicazione XA Monitor in esecuzione per client (utente o computer), poiché ogni client ha la propria associazione del gestore code utilizzata nelle transazioni.

- **Più applicazioni IBM MQ , DTC singolo, diverse istanze del gestore code:** in questo caso d'uso, esistono più di un'applicazione IBM MQ in un singolo DTC (ossia, su un computer, sono in esecuzione più di una applicazione IBM MQ) e che si connette a gestori code differenti.

  Se la transazione ha esito negativo e diventa incompleta, l'applicazione di monitoraggio recupera la transazione.

  In questo caso d'uso, ci saranno tante istanze di applicazione di controllo in esecuzione come gestori code a cui si è connessi, poiché ogni applicazione ha il proprio gestore code utilizzato nelle transazioni e ognuna di esse deve essere recuperata.

  Nota: Se l'applicazione di monitoraggio XA non è in esecuzione in background, è possibile avviarla.

Concetti correlati

“Processo di recupero della transazione per IBM MQ .NET” a pagina 574
Questa sezione descrive come è possibile ripristinare le transazioni distribuite con le classi IBM MQ .NET .

Attività correlate

“Utilizzo dell'applicazione WMQDotnetXAMonitor” a pagina 576
Il client IBM MQ .NET fornisce un'applicazione XA Monitor, WmqDotnetXAMonitor, che è possibile utilizzare per ripristinare tutte le transazioni distribuite incomplete. L'applicazione XAMonitor WmqDotnetstabilisce una connessione al gestore code in cui le transazioni sono in dubbio e quindi risolve la transazione in base ai parametri impostati.

Utilizzo dell'applicazione WMQDotnetXAMonitor

Il client IBM MQ .NET fornisce un'applicazione XA Monitor, WmqDotnetXAMonitor, che è possibile utilizzare per ripristinare tutte le transazioni distribuite incomplete. L'applicazione XAMonitor WmqDotnetstabilisce una connessione al gestore code in cui le transazioni sono in dubbio e quindi risolve la transazione in base ai parametri impostati.

Informazioni su questa attività

L'applicazione WMQDotnetXAMonitor deve essere eseguita manualmente. Può essere avviato in qualsiasi momento. È possibile avviarlo quando vengono visualizzati i messaggi sul SISTEMA SYSTEM.DOTNET.XARECOVERY.QUEUE oppure è possibile mantenerla in esecuzione in background prima di eseguire qualsiasi operazione transazionale con le applicazioni scritte utilizzando le classi IBM MQ .NET .

È possibile impostare i valori dei parametri per WMQDotnetXAMonitor tramite la riga comandi o utilizzando un file di configurazione dell'applicazione. I valori forniti tramite il file di configurazione dell'applicazione hanno la precedenza sui valori impostati tramite la riga comandi.

Prima di IBM MQ 9.3.0, la connessione stabilita da WMQDotnetXAMonitor era una connessione non sicura.

V 9.3.0 Da IBM MQ 9.3.0, è possibile stabilire una connessione sicura al gestore code impostando ulteriori parametri per WMQDotnetXAMonitor.

Procedura

- Per fornire l'input a WmqDotNETXAMonitor utilizzando un file di configurazione dell'applicazione, consultare “Impostazioni del file di configurazione dell'applicazione WmqDotNETXAMonitor” a pagina 578.

- Per avviare l'applicazione WMQDotnetXAMonitor dalla riga comandi, utilizzare il comando seguente con i parametri richiesti:

  Prima di IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  V 9.3.0 Da IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  I parametri che è possibile specificare sono i seguenti:

  – **-m *QueueManagerNome***
  Il nome del gestore code.

  Facoltativo

  -n *ConnectionName*
  Il nome connessione in formato host (porta). *ConnectionName* può contenere più di un nome connessione. Più nomi di connessione devono essere forniti in un elenco separato da virgole, ad esempio `localhost (1414), localhost (1415), localhost (1416)`. L'applicazione WMQDotnetXAMonitor esegue il ripristino per ciascuno dei nomi di connessione specificati nell'elenco separato da virgole.

  -c *ChannelName*
  Il nome del canale.

  -i
  Completamento ramo euristico.

  Facoltativo

  V 9.3.0 **-k *Repository chiavi SSL***
  Il nome del repository chiavi SSL. I valori supportati sono:

  - *SYSTEM (questo è il valore predefinito)

  - *UTENTE

  Facoltativo

  V 9.3.0 **-s *Specifica cifratura***
  La CipherSpec impostata deve essere una delle CipherSpecs per la versione supportata e può essere preferibilmente uguale a quella specificata nella politica di gruppo Windows . Per ulteriori informazioni, consultare “Supporto CipherSpec per il client .NET gestito” a pagina 599.

  Obbligatorio per stabilire una connessione sicura al gestore code.

  V 9.3.0 **-dn *Nome peer SSL***
  Il nome peer SSL utilizzato per controllare il DN (Distinguished Name) del certificato dal gestore code peer.

  Facoltativo

  V 9.3.0 **-cl *Etichetta certificato***
  Il nome dell'etichetta che identifica il certificato.

  Facoltativo

  V 9.3.0 **-sn *OutboundSNI***
  Se SNI (Server Name Indication) deve essere impostato sul nome del canale IBM MQ di destinazione sul sistema remoto quando si avvia una connessione TLS o sul nome host. I valori supportati per questa opzione sono:

- CHANNEL (questo è il valore predefinito)
- HOSTNAME
- *

Se non viene impostato alcun valore, viene utilizzato il valore predefinito CHANNEL.

Facoltativo

V 9.3.0 **-cr *Controllo revoca certificato***

Indica se deve essere eseguito il controllo della revoca della certificazione. I valori supportati per questa opzione sono:

- vero, true
- false (questo è l'impostazione predefinita)

Facoltativo

V 9.3.0 **-kr *KeyResetConteggio***

Il numero totale di byte non codificati inviati e ricevuti sul canale prima che la chiave segreta utilizzata per la crittografia venga rinegoziata.

Il valore predefinito 0 indica che le chiavi segrete non vengono mai rinegoziate

Facoltativo

L'applicazione WMQDotnetXAMonitor effettua quanto segue:

1. Verifica la profondità della coda di SYSTEM.DOTNET.XARECOVERY.QUEUE ad un intervallo di 100 secondi.
2. Se la profondità della coda è maggiore di zero, ricerca i messaggi nella coda e verifica se i messaggi soddisfano i criteri di transazione incompleti.
3. Se un messaggio soddisfa i criteri di transazione incompleti, lo estrae e richiama le informazioni di ripristino della transazione.
4. Determina se le informazioni di ripristino sono relative al Microsoft Distributed Transaction coordinator (MS DTC) locale. In questo caso, WMQDotnetXAMonitor procede al recupero della transazione, altrimenti torna indietro per sfogliare il messaggio successivo.
5. Effettua chiamate al gestore code per ripristinare la transazione incompleta.

Impostazioni del file di configurazione dell'applicazione WmqDotNETXAMonitor

È possibile fornire l'input per l'applicazione IBM MQ .NET XA Monitor, WmqDotNETXAMonitor, utilizzando un file di configurazione dell'applicazione. Un file di configurazione dell'applicazione di esempio viene fornito con IBM MQ .NET. È possibile modificare questo file di esempio in base ai propri requisiti.

I valori di input forniti tramite il file di configurazione dell'applicazione hanno la precedenza più alta. Se si forniscono valori di input sia sulla riga comandi come descritto in “Utilizzo dell'applicazione WMQDotnetXAMonitor” a pagina 576 che nel file di configurazione dell'applicazione, i valori dal file di configurazione dell'applicazione hanno la precedenza.

File di configurazione dell'applicazione di esempio per IBM MQ 9.3.0precedente.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
```

```
</dnetxa>
</dnetxa>
</configuration>
```

V 9.3.0 File di configurazione dell'applicazione di esempio da IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

Log dell'applicazione WmqDotNetXAMonitor
L'applicazione Monitor crea un file di log nella directory dell'applicazione per registrare l'avanzamento del monitoraggio e lo stato di ripristino della transazione. La registrazione inizia con il nome della connessione e i dettagli del canale per mostrare il gestore code corrente per cui è in esecuzione il recupero.

Una volta avviato il ripristino, verrà registrato MessageId del messaggio di ripristino della transazione, TransactionId della transazione incompleta e del risultato effettivo della transazione come per Transaction Manager Coordination.

File di log di esempio:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

Scrittura e distribuzione di programmi IBM MQ .NET

Per utilizzare IBM MQ classes for .NET per accedere alle code IBM MQ , scrivere i programmi in qualsiasi lingua supportata da .NET contenente le chiamate che inserono i messaggi e ricevono i messaggi dalle code IBM MQ .

La documentazione IBM MQ contiene informazioni solo sui linguaggi C#, C+ + e Visual Basic.

Questa raccolta di argomenti fornisce informazioni utili per la scrittura di applicazioni per interagire con i sistemi IBM MQ . Per i dettagli sulle singole classi, consultare Le classi e le interfacce IBM MQ .NET.

Differenze di connessione

Il modo in cui si programma per IBM MQ.NET ha alcune dipendenze dalle modalità di connessione che si desidera utilizzare.

Quando IBM MQ classes for .NET viene utilizzato come client gestito, esistono diverse differenze rispetto a IBM MQ MQI clientstandard, poiché alcune funzioni non sono disponibili per un client gestito.

IBM MQ.NET determina quale tipo di connessione utilizzare dalle impostazioni specificate per il nome connessione, il nome canale, il valore di personalizzazione NMQ_MQ_LIB e la proprietà MQC.TRANSPORT_PROPERTY.

Connessioni client gestite

Quando IBM MQ classes for .NET viene utilizzato come client gestito, esistono diverse differenze da un IBM MQ MQI clientstandard.

Le seguenti funzioni non sono disponibili per un client gestito:

- Compressione canale
- Concatenamento uscita canale

Se si tenta di utilizzare queste funzioni con un client gestito, verrà restituita un'eccezione MQException. Se l'errore viene rilevato alla fine del client di una connessione, utilizzerà il codice di errore MQRC_ENVIRONMENT_ERROR. Se viene rilevato all'estremità del server, verrà utilizzato il codice di errore restituito dal server.

Le uscite del canale scritte per un client non gestito non funzionano. È necessario scrivere nuove uscite specificamente per il client gestito. Verificare che non vi siano uscite canale non valide specificate nella tabella di definizione del canale client (CCDT).

Il nome di un'uscita canale gestito può avere una lunghezza massima di 999 caratteri. Tuttavia, se si utilizza la CCDT per specificare il nome dell'uscita del canale, è limitato a 128 caratteri.

La comunicazione è supportata solo su TCP/IP.

Quando si arresta un gestore code utilizzando il comando **endmqm** , la chiusura di un canale di connessione server a un client gestito .NET può richiedere più tempo rispetto ai canali di connessione server ad altri client.

Se *NMQ_MQ_LIB* è stato impostato su `gestito` per utilizzare la diagnostica dei problemi IBM MQ , nessuno dei parametri -i, -p, -s, -b o -c del comando **strmqtrc** è supportato.

Un'applicazione .NET gestita che utilizza transazioni XA non funzionerà con un gestore code z/OS .
Un client .NET gestito che tenta di connettersi a un gestore code z/OS non riesce con un errore, MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354), nella chiamata MQOPEN. Tuttavia, un'applicazione .NET gestita che utilizza transazioni XA funzionerà con il gestore code distribuito.

Definizione del tipo di connessione da utilizzare

Il tipo di connessione è determinato dall'impostazione del nome della connessione, del nome canale, del valore di personalizzazione NMQ_MQ_LIB e della proprietà MQC.TRANSPORT_PROPERTY.

È possibile specificare il nome della connessione nel modo seguente:

- Esplicitamente su un costruttore MQQueueManager :

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Impostando le proprietà MQC.HOST_NAME_PROPERTY e, facoltativamente, MQC.PORT_PROPERTY in una voce hashtable su un costruttore MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Come valori MQEnvironment espliciti

```
MQEnvironment.Hostname
```

  `MQEnvironment.Port` (facoltativo).
- Impostando le proprietà MQC.HOST_NAME_PROPERTY e, facoltativamente, MQC.PORT_PROPERTY nell'hashtable MQEnvironment.properties .

È possibile specificare il nome del canale come segue:

- Esplicitamente su un costruttore MQQueueManager :

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Impostando la proprietà MQC.CHANNEL_PROPERTY in una voce hashtable su un costruttore di MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Come valore MQEnvironment esplicito

```
MQEnvironment.Channel
```

- Impostando la proprietà MQC.CHANNEL_PROPERTY nell'hashtable MQEnvironment.properties .

È possibile specificare la proprietà di trasporto come segue:

- Impostando la proprietà MQC.TRANSPORT_PROPERTY in una voce hashtable su un costruttore MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Impostando la proprietà MQC.TRANSPORT_PROPERTY nell'hashtable MQEnvironment.properties .

Selezionare il tipo di connessione richiesto utilizzando uno dei valori riportati di seguito:

MQC.TRANSPORT_MQSERIES_BINDINGS - connettersi come server
MQC.TRANSPORT_MQSERIES_CLIENT - connettersi come client non XA
MQC.TRANSPORT_MQSERIES_XACLIENT - connettersi come client XA
MQC.TRANSPORT_MQSERIES_MANAGED - connettersi come client gestito non XA

È possibile impostare il valore di personalizzazione NMQ_MQ_LIB per scegliere esplicitamente il tipo di connessione come mostrato nella seguente tabella.

Valore NMQ_MQ_LIB	**Tipo di connessione**
mqic.dll	Connetti come client non XA
mqicxa.dll	Connetti come client XA
mqm.dll	Connetti come server o come client non XA
gestito	Connetti come client gestito non XA

Nota: I valori di mqic32.dll e mqic32xa.dll sono accettati come sinonimi di mqic.dll e mqicxa.dll per la compatibilità con le release precedenti. Tuttavia, mqm.dll e mqm.pdb sono solo parte del pacchetto client da IBM WebSphere MQ 7.1 in poi.

Se si sceglie un tipo di collegamento non disponibile nel proprio ambiente, ad esempio si specifica mqic32xa.dll e non si dispone del supporto XA, IBM MQ.NET genera un'eccezione.

L'impostazione di NMQ_MQ_LIB su "gestito" fa sì che il client utilizzi i test di diagnostica dei problemi IBM MQ gestiti, la conversione dei dati .NET e altre funzioni gestite di basso livello IBM MQ .

Tutti gli altri valori per NMQ_MQ_LIB fanno sì che il processo .NET utilizzi i test diagnostici dei problemi IBM MQ non gestiti e la conversione dei dati e altre funzioni IBM MQ di basso livello non gestite (supponendo che sul sistema sia installato un IBM MQ MQI client o un server).

IBM MQ.NET sceglie il tipo di connessione come segue:

1. Se MQC.TRANSPORT_PROPERTY è specificato, si connette in base al valore di MQC.TRANSPORT_PROPERTY.

   Notare, tuttavia, che l'impostazione di MQC.TRANSPORT_PROPERTY in MQC.TRANSPORT_MQSERIES_MANAGED non garantisce che il processo client venga eseguito gestito. Anche con questa impostazione, il cliente non viene gestito nei seguenti casi:

   - Se un altro thread nel processo si è collegato a MQC.TRANSPORT_PROPERTY impostato su un valore diverso da MQC.TRANSPORT_MQSERIES_MANAGED.
   - Se NMQ_MQ_LIB non è impostato su "gestito", i test diagnostici dei problemi, la conversione dei dati e altre funzioni di basso livello non sono completamente gestiti (supponendo che sul sistema sia installato un IBM MQ MQI client o un server).
2. Se un nome di connessione è stato specificato senza un nome di canale o un nome di canale è stato specificato senza un nome di connessione, viene generato un errore.
3. Se sono stati specificati sia un nome connessione che un nome canale:
   - Se NMQ_MQ_LIB è impostato su mqic32xa.dll, si collega come un client XA.
   - Se NMQ_MQ_LIB è impostato su gestito, si connette come client gestito.
   - Altrimenti si connette come un client non XA.
4. Se viene specificato NMQ_MQ_LIB, si connette in base al valore di NMQ_MQ_LIB.
5. Se è installato un server IBM MQ , si connette come server.
6. Se un IBM MQ MQI client è installato, si connette come un client non XA.
7. Altrimenti, si connette come client gestito.

Windows Utilizzo del modello di progetto IBM MQ .NET

Il client IBM MQ .NET offre la possibilità di utilizzare un template di progetto per assistere l'utente nello sviluppo di applicazioni .NET Core .

Prima di iniziare

È necessario disporre di Microsoft Visual Studio 2017o versione successiva e di .NET Core 2.1 sul sistema.

È necessario copiare il template .NET dal

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip

nella directory

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

dove:

- *& MQ_INSTALL_ROOT* è la directory root dell'installazione
- *& USER_HOME_DIRECTORY* è la tua home directory.

È necessario arrestare e riavviare Microsoft Visual Studio per prelevare il template.

Informazioni su questa attività

Il modello di progetto .NET include del codice comune che è possibile utilizzare per sviluppare le applicazioni. Con il codice integrato, puoi connetterti al gestore code IBM MQ ed eseguire un'operazione put o get semplicemente modificando le proprietà nel codice integrato.

Procedura

1. Aprire Microsoft Visual Studio.
2. Fare clic su **File**, seguito da **Nuovo** e quindi da **Progetto**.
3. Nella finestra *Crea un nuovo progetto*, selezionare `IBM MQ .NET Client App (.NET Core)` e fare clic su **Avanti**.
4. Nella finestra *Configura il nuovo progetto* , modificare il *Nome progetto* del progetto e fare clic su **Crea** per creare il progetto .NET .

   `MQDotnetApp.cs` è il file creato insieme al file del progetto. Questo file contiene il codice che si connette al gestore code ed esegue un'operazione put e get.

   Le proprietà di connessione sono impostate su valori predefiniti:

   - MQC.CONNECTION_NAME_PROPERTY è impostata su *localhost (1414)*
   - MQC.CHANNEL_PROPERTY è impostata su *DOTNET.SVRCONN*

   La coda è impostata su *Q1*ed è possibile modificare queste proprietà di conseguenza.
5. Compilare ed eseguire l'applicazione.

Concetti correlati

Componenti e funzioni di IBM MQ

.NET runtime applicazione - solo Windows

File di configurazione per IBM MQ classes for .NET

Un'applicazione client .NET può utilizzare un file di configurazione IBM MQ MQI client e, se si sta utilizzando il tipo di connessione gestita, un file di configurazione dell'applicazione .NET . Le impostazioni nel file di configurazione dell'applicazione hanno priorità.

File di configurazione client

Un'applicazione client IBM MQ classes for .NET può utilizzare un file di configurazione client allo stesso modo di qualsiasi altra IBM MQ MQI client. Questo file è in genere denominato `mqclient.ini`, ma è possibile specificare un nome file diverso. Per ulteriori informazioni sul file di configurazione del client, consultare IBM MQ MQI client file di configurazione, `mqclient.ini`.

Solo i seguenti attributi in un file di configurazione IBM MQ MQI client sono rilevanti per IBM MQ classes for .NET. Se si specificano altri attributi, non ha alcun effetto.

Tabella 77. Attributi del file di configurazione client rilevanti per IBM MQ classes for .NET

stanza	**Attributo**
Canali	CCSID
Canali	Directory ChannelDefinition
Canali	File ChannelDefinition
Canali	ReconDelay

Tabella 77. Attributi del file di configurazione client rilevanti per IBM MQ classes for .NET (Continua)	
stanza	**Attributo**
Canali	DefRecon
Canali	MQReconnectTimeout
Canali	Parametri ServerConnection
Canali	Put1DefaultAlwaysSync
Canali	PasswordProtection
ClientExitClientExit	ExitsDefaultPath
ClientExitClientExit	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
V 9.3.0 V 9.3.0 Sicurezza	File MQIInitialKey
V 9.3.0 V 9.3.0 SSL	SSLKeyRepository
V 9.3.0 V 9.3.0 SSL	Password SSLKeyRepository
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

È possibile sovrascrivere uno qualsiasi di questi attributi utilizzando la variabile di ambiente appropriata.

File di configurazione dell'applicazione

Se si è in esecuzione con il tipo di connessione gestita, è anche possibile sovrascrivere il file di configurazione del client IBM MQ e le variabili di ambiente equivalenti utilizzando il file di configurazione dell'applicazione .NET .

Le impostazioni del file di configurazione dell'applicazione .NET vengono utilizzate solo durante l'esecuzione con tipo di connessione gestita e vengono ignorate per altri tipi di connessione.

Il file di configurazione dell'applicazione .NET e il relativo formato sono definiti da Microsoft per un utilizzo generale all'interno del framework .NET , ma i particolari nomi di sezione, le chiavi e i valori indicati in questa documentazione sono specifici di IBM MQ.

Il formato del file di configurazione dell'applicazione .NET è un numero di *sezioni*. Ogni sezione contiene una o più *chiavi*e ogni chiave ha un *valore*associato. Il seguente esempio mostra le sezioni, chiavi e valori utilizzati in un file di configurazione dell'applicazione .NET per controllare la proprietà TCP/IP KeepAlive :

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

Le parole chiave utilizzate nei nomi e nelle chiavi della sezione del file di configurazione dell'applicazione .NET corrispondono esattamente alle parole chiave per le stanze e gli attributi definiti nel file di configurazione client.

La sezione `<configSections>` deve essere il primo elemento child dell'elemento `<configuration>` .

Consultare la documentazione Microsoft per ulteriori informazioni.

Frammento di codice C# di esempio da utilizzare con .NET

Un frammento di codice C# che dimostra che un'applicazione si connette a un gestore code, inserisce un messaggio in una coda e riceve una risposta.

Il seguente frammento di codice C# illustra un'applicazione che esegue tre operazioni:

1. Connetterti a un gestore code
2. Inserire un messaggio in SYSTEM.DEFAULT.LOCAL.QUEUE
3. Richiama il messaggio

Mostra anche come modificare il tipo di connessione.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }
```

```
    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define an IBM MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                           // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again

      // First define an IBM MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

Impostazione dell'ambiente IBM MQ

Prima di utilizzare la connessione client per connettersi a un gestore code, è necessario configurare l'ambiente IBM MQ .

Nota: Questo passo non è necessario quando si utilizza IBM MQ classes for .NET in modalità di bind del server.

L'interfaccia di programmazione .NET consente di utilizzare il valore di personalizzazione NMQ_MQ_LIB ma include anche una classe MQEnvironment. Questa classe consente di specificare i dettagli da utilizzare durante il tentativo di collegamento, come quelli nel seguente elenco:

- Nome canale
- Nome host
- Numero di porta
- Uscite canale
- Parametri SSL
- ID utente e password

Per informazioni complete sulla classe MQEnvironment, consultare MQEnvironment.NET class

Per specificare il nome canale e il nome host, utilizzare il codice seguente:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

Per impostazione predefinita, i client tentano di connettersi a un listener IBM MQ sulla porta 1414. Per specificare una porta differente, utilizzare il codice:

```
MQEnvironment.Port = nnnn;
```

Connessione e disconnessione da un gestore code

Una volta configurato l'ambiente IBM MQ , è possibile connettersi a un gestore code.

Per connettersi a un gestore code, creare una nuova istanza della classe MQQueueManager :

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Per disconnettersi da un gestore code, richiamare il metodo `Disconnect` sul gestore code:

```
queueManager.Disconnect();
```

È necessario disporre dell'autorizzazione di interrogazione (`inq`) sul gestore code durante il tentativo di connessione al gestore code. Senza richiedere l'autorizzazione, il tentativo di connessione non riesce.

Se si richiama il metodo `Disconnect` , tutte le code e i processi aperti a cui è stato effettuato l'accesso tramite tale gestore code vengono chiusi. Tuttavia, è buona pratica di programmazione chiudere esplicitamente queste risorse quando si finisce di utilizzarle. Per chiudere le risorse, utilizzare il metodo `Close` sull'oggetto associato a ciascuna risorsa.

I metodi di `Commit` e `Backout` su un gestore code sostituiscono le chiamate MQCMIT e MQBACK utilizzate con l'interfaccia procedurale.

Accesso a code e argomenti

È possibile accedere alle code e agli argomenti utilizzando i metodi di MQQueueManager o i costruttori appropriati.

Per accedere alle code, utilizzare i metodi della classe MQQueueManager . MQOD (object descriptor structure) è compresso nei parametri di questi metodi. Ad esempio, per aprire una coda su un gestore code rappresentato da un oggetto MQQueueManager denominato queueManager, utilizzare il seguente codice:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

Il parametro *options* è uguale al parametro Options nella chiamata MQOPEN.

Il metodo AccessQueue restituisce un nuovo oggetto della classe MQQueue.

Una volta terminato di utilizzare la coda, utilizzare il metodo Close () per chiuderla, come nel seguente esempio:

```
queue.Close();
```

Con IBM MQ .NET, è anche possibile creare una coda utilizzando il costruttore MQQueue. I parametri sono esattamente gli stessi del metodo accessQueue , con l'aggiunta di un parametro del gestore code che specifica l'oggetto MQQueueManager istanziato da utilizzare. Ad esempio:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

La creazione di un oggetto coda in questo modo consente di scrivere le proprie sottoclassi di MQQueue.

Allo stesso modo, è anche possibile accedere agli argomenti utilizzando i metodi della classe MQQueueManager . Utilizzare un metodo AccessTopic() per aprire un argomento. Viene restituito un nuovo oggetto della classe MQTopic. Una volta terminato di utilizzare l'argomento, utilizzare il metodo Close () di MQTopic per chiuderlo.

È anche possibile creare un argomento utilizzando un costruttore MQTopic. Esistono diversi costruttori per gli argomenti; per ulteriori informazioni, consultare MQTopic.NET class.

Gestione dei messaggi

I messaggi vengono gestiti utilizzando i metodi delle classi coda o argomento. Per creare un nuovo messaggio, creare un nuovo MQMessageobject.

Inserire i messaggi nelle code o negli argomenti utilizzando il metodo Put () della classe MQQueue o MQTopic. Richiamare i messaggi dalle code o dagli argomenti utilizzando il metodo Get () della classe MQQueue o MQTopic. A differenza dell'interfaccia procedurale, dove MQPUT e MQGET immettono e acquisiscono array di byte, le istanze di immissione e di acquisizione IBM MQ classes for .NET della classe MQMessage. La classe MQMessage incapsula il buffer di dati che contiene i dati del messaggio effettivi, insieme a tutti i parametri MQMD (message descriptor) che descrivono quel messaggio.

Per creare un nuovo messaggio, creare una nuova istanza della classe MQMessage e utilizzare i metodi WriteXXX per inserire i dati nel buffer di messaggio.

Quando viene creata la nuova istanza di messaggio, tutti i parametri MQMD vengono impostati automaticamente sui loro valori predefiniti, come definito in Valori iniziali e dichiarazioni della lingua per MQMD. Il metodo Put () di MQQueue utilizza anche un'istanza della classe MQPutMessageOptions come parametro. Questa classe rappresenta la struttura MQPMO. Il seguente esempio crea un messaggio e lo inserisce in una coda:

```
// Build a new message containing my age followed by my name
```

```
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage,pmo);
```

Il metodo Get () di MQQueue restituisce una nuova istanza di MQMessage, che rappresenta il messaggio appena preso dalla coda. Inoltre, utilizza un'istanza della classe di opzioni MQGetMessagecome parametro. Questa classe rappresenta la struttura MQGMO.

Non è necessario specificare una dimensione massima del messaggio, poiché il metodo Get () regola automaticamente la dimensione del relativo buffer interno per adattarla al messaggio in arrivo. Utilizzare i metodi ReadXXX della classe MQMessage per accedere ai dati nel messaggio restituito.

Il seguente esempio mostra come ottenere un messaggio da una coda:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

È possibile modificare il formato numerico utilizzato dai metodi di lettura e scrittura impostando la variabile membro *encoding* .

È possibile modificare la serie di caratteri da utilizzare per la lettura e la scrittura di stringhe impostando la variabile membro *characterSet* .

Per ulteriori dettagli, consultare MQMessage.NET class .

Nota: Il metodo WriteUTF() di MQMessage codifica automaticamente la lunghezza della stringa e i byte Unicode che contiene. Quando il messaggio verrà letto da un altro programma .NET (utilizzando ReadUTF()), questo è il metodo più semplice per inviare informazioni sulla stringa.

Gestione delle proprietà dei messaggi

Le proprietà del messaggio consentono di selezionare i messaggi o di richiamare le informazioni su un messaggio senza accedervi. La classe MQMessage contiene i metodi per richiamare e impostare le proprietà.

È possibile utilizzare le proprietà del messaggio per consentire a un'applicazione di selezionare i messaggi da elaborare o per richiamare le informazioni relative a un messaggio senza accedere alle intestazioni MQMD o MQRFH2 . Inoltre, facilitano la comunicazione tra applicazioni IBM MQ e JMS . Per ulteriori informazioni sulle proprietà dei messaggi in IBM MQ, consultare Proprietà dei messaggi.

La classe MQMessage fornisce una quantità di metodi per richiamare e impostare le proprietà, in base al tipo di dati della proprietà. I metodi get hanno i nomi del formato Get * Property e i metodi set hanno i nomi del formato Set * Property, dove l'asterisco (*) rappresenta una delle seguenti stringhe:

- Booleano
- Byte
- Byte
- Doppia
- Mobile
- Int
- Int2

- Int4
- Int8
- Lungo
- Oggetto
- Breve
- Stringa

Ad esempio, per ottenere la proprietà IBM MQ myproperty (una stringa di caratteri), utilizzare la chiamata `message.GetStringProperty('myproperty')`. È possibile, facoltativamente, passare un descrittore di proprietà, che verrà completato da IBM MQ .

Gestione degli errori

Gestire gli errori derivanti dall' IBM MQ classes for .NET utilizzo dei blocchi `try` e `catch` .

I metodi nell'interfaccia .NET non restituiscono un codice di completamento e un codice motivo. Al contrario, generano un'eccezione ogni volta che il codice di completamento e il codice motivo risultanti da una chiamata IBM MQ non sono entrambi zero. Ciò semplifica la logica del programma in modo che non sia necessario controllare i codici di ritorno dopo ogni chiamata a IBM MQ. È possibile decidere in quali punti del programma si desidera affrontare la possibilità di errore. In questi punti, puoi racchiudere il tuo codice con blocchi `try` e `catch` , come nel seguente esempio:

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                        "CC = " + ex.CompletionCode +
                        "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

Richiamo e impostazione dei valori di attributo

Le classi MQManagedObject, MQQueue e MQQueueManager contengono metodi che consentono di richiamare e impostare i loro valori di attributo. Tenere presente che per MQQueue, i metodi funzionano solo se si specifica l'interrogazione appropriata e si impostano gli indicatori quando si apre la coda.

Per gli attributi comuni, le classi MQQueueManager e MQQueue ereditano da una classe denominata MQManagedObject. Questa classe definisce le interfacce Inquire () e Set ().

Quando si crea un nuovo oggetto gestore code utilizzando il *nuovo* operatore, viene automaticamente aperto per l'interrogazione. Quando si utilizza il metodo AccessQueue() per accedere a un oggetto coda, tale oggetto non viene aperto automaticamente per le operazioni di interrogazione o di impostazione, ciò potrebbe causare problemi con alcuni tipi di code remote. Per utilizzare i metodi Inquire e Set e per impostare le proprietà su una coda, è necessario specificare gli indicatori inquire e set appropriati nel parametro openOptions del metodo AccessQueue().

I metodi inquire e set utilizzano tre parametri:

- array selettori
- array intAttrs
- Array charAttrs

Non sono necessari i parametri SelectorCount, IntAttrCount e CharAttrLength rilevati in MQINQ, poiché la lunghezza di un array è sempre nota. Il seguente esempio mostra come eseguire un'interrogazione su una coda:

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

Tutti gli attributi di questi oggetti possono essere interrogati. Un sottoinsieme di attributi viene esposto come proprietà di un oggetto. Per un elenco di attributi oggetto, consultare Attributi degli oggetti. Per le proprietà dell'oggetto, consultare la descrizione della classe appropriata.

Programmi a più sottoprocessi

L'ambiente di runtime .NET è intrinsecamente multithread. IBM MQ classes for .NET consente a un oggetto gestore code di essere condiviso tra più thread, ma garantisce che tutto l'accesso al gestore code di destinazione sia sincronizzato.

Considerare un programma semplice che si connette a un gestore code e apre una coda all'avvio. Il programma visualizza un singolo pulsante sullo schermo. Quando un utente fa clic su quel pulsante, il programma richiama un messaggio dalla coda. In questa situazione, l'inizializzazione dell'applicazione si verifica in un thread e il codice che viene eseguito in risposta alla pressione del pulsante viene eseguito in un thread separato (il thread dell'interfaccia utente).

L'implementazione di IBM MQ .NET garantisce che, per un particolare collegamento (istanza oggettoMQQueueManager), tutti gli accessi al gestore code di destinazione IBM MQ siano sincronizzati. Il comportamento predefinito è che un thread che desidera emettere una chiamata a un gestore code viene bloccato fino a quando non vengono completate tutte le chiamate in corso per tale connessione. Se si richiede l'accesso simultaneo allo stesso gestore code da più thread all'interno del programma, creare un nuovo oggetto MQQueueManager per ogni thread che richiede l'accesso simultaneo. (Ciò equivale all'emissione di una chiamata MQCONN separata per ciascun thread.)

Se le opzioni di connessione predefinite vengono sovrascritte da MQC.MQCNO_HANDLE_SHARE_NONE o MQC.MQCNO_SHARE_NO_BLOCK il gestore code non è più sincronizzato.

Utilizzo di una tabella di definizione di canale client con .NET

È possibile utilizzare una CCDT (client channel definition table) con IBM MQ classes for .NET. L'ubicazione della CCDT viene specificata in modi diversi, a seconda che si stia utilizzando una connessione gestita o non gestita.

Tipo di connessione client non gestito XA o non XA

Con un tipo di connessione non gestito, è possibile specificare l'ubicazione della CCDT in due modi:

- Utilizzo delle variabili di ambiente MQCHLLIB per specificare la directory in cui si trova la tabella e MQCHLTAB per specificare il nome file della tabella.
- Utilizzo del file di configurazione client. Nella stanza CHANNELS, utilizzare gli attributi **ChannelDefinitionDirectory** per specificare la directory in cui si trova la tabella e **ChannelDefinitionFile** per specificare il nome file.

Se l'ubicazione è specificata sia nel file di configurazione client che utilizzando le variabili di ambiente, le variabili di ambiente hanno la priorità. È possibile utilizzare questa funzione per specificare un'ubicazione standard nel file di configurazione del client e sovrascriverla utilizzando le variabili di ambiente quando necessario.

Tipo di connessione client gestito

Con un tipo di collegamento gestito, è possibile specificare l'ubicazione della CCDT in tre modi:

- Utilizzo del file di configurazione applicazione .NET . Nella sezione CHANNELS, utilizzare le chiavi **ChannelDefinitionDirectory** per specificare la directory in cui si trova la tabella e **ChannelDefinitionFile** per specificare il nome file.
- Utilizzo delle variabili di ambiente MQCHLLIB per specificare la directory in cui si trova la tabella e MQCHLTAB per specificare il nome file della tabella.
- Utilizzo del file di configurazione client. Nella stanza CHANNELS, utilizzare gli attributi **ChannelDefinitionDirectory** per specificare la directory in cui si trova la tabella e **ChannelDefinitionFile** per specificare il nome file.

Se l'ubicazione viene specificata in più di uno di questi metodi, le variabili di ambiente hanno la priorità sul file di configurazione del client e il file di configurazione dell'applicazione .NET ha la priorità su entrambi gli altri metodi. È possibile utilizzare questa funzione per specificare un'ubicazione standard nel file di configurazione del client e sovrascriverla utilizzando le variabili di ambiente o il file di configurazione dell'applicazione quando necessario.

Prima di IBM MQ 9.3.0, c'è una differenza nel comportamento tra i client .NET gestiti e i client IBM MQ Java e C quando si utilizza una CCDT con un raggruppamento di gestori code. Quando un file CCDT contiene un gruppo di gestori code di tre gestori code e tre CLNTCONNs espliciti agli stessi tre gestori code e l'applicazione fornisce "*" come gestore code, i client C e Java restituiscono MQRC_Q_MGR_NAME_ERROR. Tuttavia, il client .NET gestito utilizza il primo CLNTCONN disponibile e, se nessuno è disponibile, utilizza il CLNTCONN raggruppato del gestore code.

V 9.3.0 V 9.3.0 Da IBM MQ 9.3.0, il client .NET si comporta nello stesso modo dei client C e Java e restituisce MQRC_Q_MGR_NAME_ERROR quando si utilizza una CCDT con un raggruppamento di gestori code.

Come un'applicazione .NET determina quale definizione di canale utilizzare

Nell'ambiente client IBM MQ .NET , la definizione di canale da utilizzare può essere specificata in diversi modi. Possono esistere più specifiche della definizione del canale. Un'applicazione deriva la definizione del canale da una o più origini.

Se esiste più di una definizione di canale, quella utilizzata viene selezionata nel seguente ordine di priorità:

1. Proprietà specificate nel costruttore MQQueueManager , esplicitamente o includendo *MQC.CHANNEL_PROPERTY* nella tabella hash delle proprietà
2. Una proprietà *MQC.CHANNEL_PROPERTY* in MQEnvironment.properties hashtable
3. La proprietà *Canale* in MQEnvironment
4. Il file di configurazione dell'applicazione .NET , nome sezione CHANNELS, chiave ServerConnectionParms (si applica solo alle connessioni gestite)
5. La variabile di ambiente *MQSERVER*
6. Il file di configurazione del client, stanza CHANNELS, attributo ServerConnectionParms
7. La tabella CCDT (client channel definition table). L'ubicazione di CCDT è specificata nel file di configurazione dell'applicazione .NET (si applica solo alle connessioni gestite)
8. La tabella CCDT (client channel definition table). L'ubicazione della CCDT viene specificata utilizzando le variabili di ambiente *MQCHLIB* e *MQCHLTAB*
9. La tabella CCDT (client channel definition table). L'ubicazione di CCDT è specificata utilizzando il file di configurazione client

Per gli elementi 1-3, la definizione di canale viene creata campo per campo dai valori forniti dall'applicazione. Questi valori possono essere forniti utilizzando interfacce differenti e possono esistere più valori per ciascuno di essi. I valori dei campi vengono aggiunti alla definizione di canale seguendo l'ordine di priorità fornito:

1. Il valore di *connName* sul costruttore MQQueueManager
2. Valori delle proprietà dalla tabella hash MQQueueManager.properties
3. Valori delle proprietà dalla tabella hash MQEnvironment.properties
4. Valori impostati come campi MQEnvironment (ad esempio, MQEnvironment.Hostname, MQEnvironment.Port)

Per gli elementi 4-6, l'intera definizione di canale viene fornita come valore. I campi non specificati nella definizione di canale assumono i valori predefiniti del sistema. Nessun valore proveniente da altri metodi di definizione dei canali e dei relativi campi viene unito a tali specifiche.

Per gli articoli 7-9, l'intera definizione di canale viene presa dalla CCDT. I campi non specificati esplicitamente quando il canale è stato definito assumono i valori predefiniti del sistema. Nessun valore proveniente da altri metodi di definizione dei canali e dei relativi campi viene unito a tali specifiche.

Utilizzo delle uscite di canale in IBM MQ .NET

Se si utilizzano i collegamenti client, è possibile utilizzare le uscite canale come per qualsiasi altra connessione client. Se si utilizzano i collegamenti gestiti, è necessario scrivere un programma di uscita che implementi un'interfaccia appropriata.

Bind client

Se si utilizzano i collegamenti client, è possibile utilizzare le uscite canale come descritto in Uscite canale. Non è possibile utilizzare le uscite del canale scritte per i bind gestiti.

Bind gestiti

Se si utilizza una connessione gestita, per implementare un'uscita, si definisce una nuova classe .NET che implementa l'interfaccia appropriata. Nel package IBM MQ sono definite tre interfacce di uscita:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Nota: Le uscite utente scritte utilizzando queste interfacce non sono supportate come uscite canale nell'ambiente non gestito.

Il seguente esempio definisce una classe che implementa tutti e tre:

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // complete the body of the send exit here
  }


  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // complete the body of the receive exit here
  }


  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
```

```
                        MQChannelDefinition channelDefParms,
                        byte[]              dataBuffer
                        ref int             dataOffset
                        ref int             dataLength
                        ref int             dataMaxLength)
  {
     // complete the body of the security exit here
  }

}
```

Ad ogni uscita viene passata una MQChannelExit e un'istanza oggetto MQChannelDefinition . Questi oggetti rappresentano le strutture MQCXP e MQCD definite nell'interfaccia procedurale.

I dati che devono essere inviati da un'uscita di invio e i dati ricevuti in un'uscita di sicurezza o di ricezione vengono specificati utilizzando i parametri dell'uscita.

All'immissione, i dati all'offset *dataOffset* con lunghezza *dataLength* nell'array di byte *dataBuffer* sono i dati che verranno inviati da un'uscita di invio e i dati ricevuti in un'uscita di sicurezza o di ricezione. Il parametro *dataMaxLength* fornisce la lunghezza massima (da *dataOffset*) disponibili per l'uscita in *dataBuffer*. Nota: per un'uscita di sicurezza, è possibile che dataBuffer sia null, se questa è la prima volta che l'uscita viene richiamata o se il partner ha deciso di non inviare dati.

Al ritorno, il valore di *dataOffset* e *dataLength* deve essere impostato in modo da puntare all'offset e alla lunghezza all'interno della matrice di byte restituita che le classi .NET devono utilizzare. Per un'uscita di invio, indica i dati che devono essere inviati e, per un'uscita di sicurezza o di ricezione, i dati che devono essere interpretati. L'uscita dovrebbe normalmente restituire una schiera di byte; le eccezioni sono un'uscita di sicurezza che potrebbe scegliere di non inviare dati e qualsiasi uscita richiamata con i motivi INIT o TERM. La forma più semplice di uscita che può essere scritta è quella che non fa altro che restituire dataBuffer:

Il corpo di uscita più semplice possibile è:

```
{
  return dataBuffer;
}
```

Classe MQChannelDefinition

L'id utente e la password specificati con l'applicazione client .NET gestita vengono impostati nella classe IBM MQ .NET MQChannelDefinition passata all'uscita di sicurezza client. L'uscita di sicurezza copia l'id utente e password in MQCD MQCD.RemoteUserIdentifier e MQCD.RemotePassword RemotePassword (consultare “Scrittura di un'uscita di sicurezza” a pagina 976).

Specifica delle uscite canale (client gestito)

Se si specifica un nome canale e un nome connessione quando si crea l'oggetto MQQueueManager (nell'MQEnvironment o nel costruttore MQQueueManager), è possibile specificare le uscite del canale in due modi.

In ordine di precedenza, sono:

1. Passaggio delle proprietà della hashtable MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY o MQC.RECEIVE_EXIT_PROPERTY sul costruttore MQQueueManager .
2. Impostazione delle proprietà MQEnvironment SecurityExit, SendExit o ReceiveExit .

Se non si specifica un nome canale e un nome connessione, le uscite del canale da utilizzare provengono dalla definizione del canale prelevata da una CCDT (client channel definition table). Non è possibile sovrascrivere i valori memorizzati nella definizione del canale. Consultare Tabella di definizione del canale client e “Utilizzo di una tabella di definizione di canale client con .NET” a pagina 591 per ulteriori informazioni sulle tabelle di definizione del canale.

In ogni caso, la specifica assume la forma di una stringa con il seguente formato:

```
Assembly_name(Class_name)
```

Class_name è il nome completo, inclusa la specifica dello spazio dei nomi, di una classe .NET che implementa IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit o IBM.WMQ.MQReceiveExit (come appropriato). *Assembly_name* è l'ubicazione completa, inclusa l'estensione file, dell'assembly che ospita la classe. La lunghezza della stringa è limitata a 999 caratteri se si utilizzano le proprietà di MQEnvironment o MQQueueManager. Tuttavia, se il nome dell'uscita del canale è specificato in CCDT, è limitato a 128 caratteri. Quando necessario, il codice client .NET carica e crea un'istanza della classe specificata analizzando la specifica della stringa.

Specifica dei dati utente dell'uscita canale (client gestito)

Le uscite del canale possono avere dati utente associati. Se si specifica un nome canale e un nome connessione quando si crea l'oggetto MQQueueManager (nell'MQEnvironment o nel costruttore MQQueueManager), è possibile specificare i dati utente in due modi.

In ordine di precedenza, sono:

1. Passaggio delle proprietà della tabella hash MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY o MQC.RECEIVE_USERDATA_PROPERTY sul costruttore MQQueueManager .
2. Impostazione delle proprietà MQEnvironment SecurityUserData, SendUserData o ReceiveUserData.

Se non si specifica un nome canale e un nome connessione, i valori dei dati utente di uscita da utilizzare provengono dalla definizione di canale selezionata dalla CCDT (client channel definition table). Non è possibile sovrascrivere i valori memorizzati nella definizione del canale. Consultare Tabella di definizione del canale client e “Utilizzo di una tabella di definizione di canale client con .NET” a pagina 591 per ulteriori informazioni sulle tabelle di definizione del canale.

In ogni caso, la specifica è una stringa, limitata a 32 caratteri.

Riconnessione client automatica in .NET

È possibile riconnettere automaticamente il proprio client a un gestore code durante un'interruzione di connessione non prevista.

Un client può essere inaspettatamente disconnesso da un gestore code se, ad esempio, il gestore code si arresta o se si verifica un malfunzionamento della rete o del server.

Senza la riconnessione automatica del client, viene generato un errore quando la connessione non riesce. È possibile utilizzare il codice di errore per ristabilire la connessione.

Un client che utilizza la funzione di riconnessione client automatica viene definito un client ricollegabile. Per creare un client ricollegabile, specificare alcune opzioni denominate opzioni di riconnessione durante la connessione al gestore code.

Se l'applicazione client è un client IBM MQ .NET , può scegliere di ottenere la riconnessione automatica del client specificando un valore appropriato per CONNECT_OPTIONS_PROPERTY quando si utilizza la classe `MQQueueManager` per creare un gestore code. Consultare Opzioni di riconnessione per dettagli sui valori di CONNECT_OPTIONS_PROPERTY.

È possibile selezionare se l'applicazione client si connette e si riconnette sempre a un gestore code con lo stesso nome, allo stesso gestore code o a qualsiasi serie di gestori code definiti con lo stesso `QMNAME` nella tabella di connessione client (per i dettagli, consultare Gruppi di gestori code in CCDT).

Supporto TLS (Transport Layer Security) per .NET

Le applicazioni client IBM MQ classes for .NET supportano la codifica TLS (Transport Layer Security). Il protocollo TLS fornisce la sicurezza delle comunicazioni su Internet e consente alle applicazioni client / server di comunicare in modo confidenziale e affidabile.

Concetti correlati

Supporto TLS del client gestito IBM MQ.NET

Supporto TLS per il client .NET non gestito

Il supporto TLS per il client .NET non gestito si basa su C MQI e IBM Global Security Kit (GSKit). C MQI gestisce le operazioni TLS e GSKit implementa i protocolli socket sicuri TLS.

Abilitazione di TLS per il client .NET non gestito

TLS è supportato solo per connessioni client. Per abilitare TLS, è necessario specificare CipherSpec da utilizzare quando si comunica con il gestore code e questo deve corrispondere alla CipherSpec impostata nel canale di destinazione.

Per abilitare TLS, specificare CipherSpec utilizzando la variabile del membro statico SSLCipherSpec di MQEnvironment. Il seguente esempio si collega a un canale SVRCONN denominato SECURE.SVRCONN.CHANNEL, che è stato impostato per richiedere TLS con una CipherSpec di TLS_RSA_WITH_AES_128_CBC_SHA:

V 9.3.0 V 9.3.0

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Consultare Specifica di CipherSpecs per un elenco di CipherSpecs.

La proprietà SSLCipherSpec può essere impostata anche utilizzando MQC.SSL_CIPHER_SPEC_PROPERTY nella tabella hash delle proprietà di connessione.

Per connettersi correttamente utilizzando TLS, il keystore del client deve essere configurato con la catena di certificati root dell'autorità di certificazione da cui è possibile autenticare il certificato presentato dal gestore code. Allo stesso modo, se SSLClientAuth sul canale SVRCONN è stato impostato su MQSSL_CLIENT_AUTH_REQUIRED, il keystore del client deve contenere un certificato personale di identificazione ritenuto attendibile dal gestore code.

Utilizzo del DN (Distinguished Name) del gestore code

Il gestore code si identifica utilizzando un certificato TLS, che contiene il DN (Distinguished Name) .

Un'applicazione client IBM MQ .NET può utilizzare questo DN per garantire la comunicazione con il gestore code corretto. Un modello DN viene specificato utilizzando la variabile nome sslPeerdi MQEnvironment. Ad esempio, l'impostazione:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

consente la riuscita della connessione solo se il gestore code presenta un certificato con un nome comune che inizia con QMGR., e almeno due nomi di unità organizzative, il cui primo deve essere IBM e il secondo WEBSPHERE.

La proprietà SSLPeerName può essere impostata anche utilizzando la MQC.SSL_PEER_NAME_PROPERTY nella tabella hash delle proprietà di connessione. Per ulteriori informazioni sui DN e le regole per l'impostazione dei nomi peer, fare riferimento a Sicurezza IBM MQ.

Se è impostato SSLPeerName , le connessioni hanno esito positivo solo se sono impostate su un modello valido e il gestore code presenta un certificato corrispondente.

Gestione degli errori durante l'utilizzo di TLS

I seguenti codici motivo possono essere emessi da IBM MQ classes for .NET quando ci si connette a un gestore code utilizzando TLS:

MQRC_SSL_NOT_ALLOWED
La proprietà SSLCipherSpec è stata impostata, ma è stata utilizzata la connessione dei bind. Solo la connessione client supporta TLS.

MQRC_SSL_PEER_NAME_MISMATCH
Il modello di DN specificato nella proprietà SSLPeerName non corrisponde al DN presentato dal gestore code.

ERRORE MQRC_SSL_PEER_NAME_ERROR
Il modello DN specificato nella proprietà SSLPeerName non era valido.

V 9.3.0 V 9.3.0 **MQRC_KEY_REPOSITORY_ERROR**
L'ubicazione del repository delle chiavi non è stata specificata, non è valida o non è possibile accedervi.

Supporto TLS per il client .NET gestito

Il client .NET gestito utilizza le librerie di Microsoft .NET Framework per implementare i protocolli socket sicuri TLS. La classe Microsoft System.Net.SecuritySslStream opera come un flusso su socket TCP connessi e invia e riceve dati su tale connessione socket.

Il livello minimo richiesto di .NET Framework è .NET Framework v3.5. Il livello di supporto dell'algoritmo di cifratura è basato sul livello .NET Framework che l'applicazione utilizza:

- Per le applicazioni che si basano sui .NET Framework livelli 3.5 e 4.0, i protocolli socket sicuri sono SSL 3.0 e TSL 1.0.
- Per le applicazioni basate sul livello .NET Framework 4.5, i protocolli socket sicuri disponibili sono SSL 3.0, TLS 1.1 e TLS 1.2.

Potrebbe essere necessario spostare le applicazioni che prevedono un supporto del protocollo TLS superiore in una versione successiva del framework, come definito per il supporto di sicurezza Microsoft in .NET Framework.

Le funzioni principali del supporto TLS per il client .NET gestito sono le seguenti:

Supporto protocollo TLS
Il supporto TLS per il client gestito .NET è definito tramite la classe SSLStream .NET e dipende dal framework .NET utilizzato dall'applicazione. Per ulteriori informazioni, fare riferimento a “Supporto del protocollo TLS per il client .NET gestito” a pagina 599.

Supporto CipherSpec
Le impostazioni TLS per il client gestito .NET sono quelle per i flussi TLS Microsoft.NET . Per ulteriori informazioni, consultare “Supporto CipherSpec per il client .NET gestito” a pagina 599 e “Associazioni CipherSpec per il client .NET gestito” a pagina 601.

Repository chiavi
Il repository delle chiavi sul client è un keystore Windows . Il repository lato server è un tipo di repository CMS (Cryptographic Message Syntax) (CMS). Per ulteriori informazioni, fare riferimento a “Repository di chiavi del client .NET gestito” a pagina 602.

Certificati
È possibile utilizzare certificati TLS autofirmati per implementare l'autenticazione reciproca tra un client e un gestore code. Per ulteriori informazioni, fare riferimento a “Utilizzo di certificati per il client .NET gestito” a pagina 603.

SSLPEERNAME
In .NET, le applicazioni possono utilizzare l'attributo SSLPEERNAME facoltativo per specificare un modello DN (Distinguished Name). Per ulteriori informazioni, fare riferimento a “SSLPEERNAME” a pagina 603.

Conformità FIPS
L'abilitazione di FIPS in modo programmatico non è supportata da Microsoft.NET Security Library. L'abilitazione FIPS è controllata dall'impostazione Criteri di gruppo Windows .

Conformità NSA Suite B
IBM MQ implementa RFC 6460. L'implementazione Microsoft.NET per la suite NSA B è 5430. Ciò è supportato da .NET Framework 3.5 in poi.

Reimpostazione o rinegoziazione della chiave segreta
Sebbene la classe SSLStream non supporti la reimpostazione o la rinegoziazione della chiave segreta, per coerenza con altri client IBM MQ , il client gestito .NET consente alle applicazioni di impostare `SSLKeyResetCount`. Per ulteriori informazioni, fare riferimento a “Reimpostazione o rinegoziazione della chiave segreta per il client .NET gestito” a pagina 604.

Controllo revoca
La classe SSLStream supporta il controllo della revoca dei certificati, che viene eseguito automaticamente dal motore di concatenamento dei certificati. Per ulteriori informazioni, fare riferimento a “Controllo revoca” a pagina 604.

Supporto uscita di sicurezza IBM MQ
La classe `SSLStream` fornisce un supporto limitato per le uscite di sicurezza IBM MQ . La query dei certificati locali e remoti per ottenere `SSLPeerNamePtr`(DN oggetto) e `SSLRemCertIssNamePtr` (DN emittente) è possibile poiché ciò è supportato in Microsoft.NET. Tuttavia, non c'è alcun supporto per ottenere attributi come DNQ, UNSTRUCTUREDNAME e UNSTRUCTUREDADDRESS, quindi questi valori non possono essere richiamati utilizzando le uscite.

Supporto hardware crittografico
L'hardware crittografico non è supportato per il client .NET gestito.

Supporto per TLS1.3 su IBM MQ .NET gestiti e client XMS .NET

V 9.3.2

Da IBM MQ 9.3.2, i clienti IBM MQ .NET e XMS .NET supportano TLS1.3 a condizione che il sistema operativo supporti TLS1.3.

Il client .NET gestito utilizza le librerie di Microsoft .NET Framework per implementare i protocolli socket sicuri TLS. La classe Microsoft `System.Net.SecuritySslStream` funziona come un flusso su socket TCP connessi e invia e riceve dati su tale connessione socket.

Su Windows, .NET utilizza SCHALLEGATO, e su Linux .NET utilizza OpenSSL per la comunicazione SSL.

Windows Per applicazioni client di IBM MQ .NET in esecuzione su Windows

Microsoft ha annunciato che Windows 11 e Windows Server 2022 supportano le cifrature TSL1.3 per impostazione predefinita.

Le suite di cifratura TLS_AES_128_GCM_SHA256 e TLS_AES_256_GCM_SHA384 sono abilitate per impostazione predefinita su entrambe le versioni di Windows.

Attenzione:

- TLS_CHACHA20_POLY1305_SHA256 Il pacchetto di crittografia non è abilitato per impostazione predefinita, ma è supportato.
- Per un client IBM MQ .NET con TLS1.3 abilitato, per connettersi correttamente a un gestore code, IBM Global Security Kit (GSKit) versione 8.0.55.29 è la versione minima richiesta sul lato del gestore code.

Linux Per applicazioni client di IBM MQ .NET in esecuzione su Linux

Poiché .NET utilizza OpenSSL su Linux per la comunicazione SSL, per utilizzare TLS1.3, OpenSSL v1.1.1 è il requisito minimo.

Inoltre, poiché .NET utilizza OpenSSL su Linux, tutte le crittografie supportate da OpenSSL dovrebbero funzionare anche per .NET .

OpenSSL supporta i seguenti CipherSpecs per TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

Concetti correlati

“Associazioni CipherSpec per il client .NET gestito” a pagina 601
L'interfaccia IBM MQ.NET conserva una tabella di associazione da IBM MQ a Microsoft.NET utilizzata per determinare la versione del protocollo TLS che il client gestito deve utilizzare per stabilire una connessione sicura con un gestore code.

Supporto del protocollo TLS per il client .NET gestito

Il supporto TLS di IBM MQ.NET è basato sulla classe SSLStream .NET .

Nota: Il supporto del protocollo TLS per il client .NET gestito dipende dal livello del framework .NET utilizzato dall'applicazione. Per ulteriori informazioni, consultare “Supporto TLS per il client .NET gestito” a pagina 597.

Per la classe SSLStream di Microsoft.NET per inizializzare TLS ed eseguire un handshake con il gestore code, uno dei parametri richiesti che è necessario impostare è **SSLProtocol**, in cui è necessario specificare il numero di versione TLS, che deve essere uno dei seguenti valori:

- SSL3.0
- TLS1.0
- TLS1.2

Il valore di questo parametro è strettamente associato alla famiglia di protocolli a cui appartiene la CipherSpec preferita. Quando SSLStream avvia un handshake TLS con il server (gestore code), utilizza la versione TLS specificata in **SSLProtocol** per identificare l'elenco di CipherSpecs da utilizzare per la negoziazione.

IBM MQ.NET non rende disponibile alcuna proprietà per le applicazioni da utilizzare per impostare questo valore. Invece, IBM MQ utilizza una tabella di associazione per associare internamente la serie CipherSpec alla famiglia Protocol e identifica la versione SSLProtocol da utilizzare. Questa tabella mostra l'associazione di ciascuna CipherSpec supportata tra Microsoft.NET e IBM MQe la versione del protocollo a cui appartengono. Per ulteriori informazioni, consultare “Associazioni CipherSpec per il client .NET gestito” a pagina 601.

Supporto CipherSpec per il client .NET gestito

Le impostazioni CipherSpec per una applicazione vengono utilizzate durante l'handshake con il server.

I client IBM MQ consentono di impostare un valore CipherSpec utilizzato durante l'handshake con il gestore code. i client IBM MQ devono impostare un CipherSpec valido per stabilire una connessione protetta, preferibilmente il CipherSpec specificato nella politica del gruppo Windows . Lasciare questo campo vuoto indica un canale di testo semplice senza alcuna sicurezza sui socket.

Per il client gestito IBM MQ.NET , le impostazioni TLS sono per la classe SSLStream di Microsoft.NET . Per SSLStream, un CipherSpeco un elenco di preferenze di CipherSpecs, può essere impostato solo nella politica del gruppo Windows , che è un'impostazione a livello di computer. SSLStream utilizza quindi la CipherSpec specificata o l'elenco delle preferenze durante l'handshake con il server. Nel caso di altri client IBM MQ , la proprietà CipherSpec può essere impostata nell'applicazione sulla definizione del canale IBM MQ e la stessa impostazione viene utilizzata per la negoziazione TLS. Come risultato di questa limitazione, l'handshake TLS potrebbe negoziare qualsiasi CipherSpec supportato, indipendentemente da quanto specificato nella configurazione del canale IBM MQ . Pertanto, è probabile che ciò provocherà l'errore AMQ9631 sul gestore code. Per evitare questo errore, impostare lo stesso valore CipherSpec impostato nell'applicazione come configurazione TLS nella politica del gruppo Windows .

Il nuovo codice client TLS IBM MQ.NET controlla solo che sia stata negoziata la corretta versione del protocollo. La versione del protocollo TLS deriva dalla CipherSpec impostata dall'applicazione e utilizzata per l'handshake TLS con il server (gestore code). Di conseguenza, è necessario per la progettazione impostare CipherSpec nell'applicazione client gestita da IBM MQ.NET . Se il CipherSpec impostato dal client IBM MQ è diverso da quello dei protocolli SSL 3.0, TLS 1.0 e TLS 1.2 , il client IBM MQ gestito .NET

negozia per impostazione predefinita con una delle cifrature dei protocolli SSL 3.0 o TLS 1.0 e non riporta un errore.

Nota: Se il valore CipherSpec fornito dall'applicazione non è un CipherSpec noto a IBM MQ, il client IBM MQ gestito .NET lo ignora e negozia la connessione in base alla politica del gruppo del sistema Windows .

Impostazione di una CipherSpec

Esistono tre modi per impostare una CipherSpec:

Classe MQEnvironment .NET

Il seguente esempio mostra come impostare un CipherSpec con la classe MQEnvironment.

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

proprietà CipherSpec TLS

Il seguente esempio mostra come impostare un CipherSpec aggiungendo un parametro hashtable nel costruttore MQQueueManager .

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows Politica di gruppo

Quando un elenco di suite di cifratura viene configurato tramite la console di gestione delle politiche del gruppo Windows , la definizione del canale SVRCONN deve specificare una CipherSpeccorrispondente. Un CipherSpec corrispondente potrebbe essere un valore generico come "ANY_TLS12_OR_HIGHER" oppure un valore specifico che si associa alla suite di cifratura più alta che viene negoziata dall'elenco ordinato. L'utilizzo di valori CipherSpec generici è consigliato per l'utilizzo con i client .NET poiché evita la necessità di modificare la configurazione CipherSpec SVRCONN se l'ordine dell'elenco di client cambia.

Utilizzo CCDT

IBM MQ.NET supporta solo tabelle di definizione del canale client (file .TAB) che si trovano su un computer locale. I file CCDT esistenti che hanno un valore CipherSpec impostato possono essere utilizzati per connessioni IBM MQ.NET . Tuttavia, il valore CipherSpec impostato sul canale di connessione client determina la versione del protocollo TLS e deve corrispondere anche alla serie CipherSpec impostata nella politica del gruppo Windows .

Concetti correlati

“Impostazione dell'ambiente IBM MQ” a pagina 587
Prima di utilizzare la connessione client per connettersi a un gestore code, è necessario configurare l'ambiente IBM MQ .

“Supporto TLS per il client .NET gestito” a pagina 597
Il client .NET gestito utilizza le librerie di Microsoft .NET Framework per implementare i protocolli socket sicuri TLS. La classe Microsoft System.Net.SecuritySslStream opera come un flusso su socket TCP connessi e invia e riceve dati su tale connessione socket.

Attività correlate

Specifica di CipherSpecs

Riferimenti correlati

Classe MQEnvironment .NET

Associazioni CipherSpec per il client .NET gestito

L'interfaccia IBM MQ.NET conserva una tabella di associazione da IBM MQ a Microsoft.NET utilizzata per determinare la versione del protocollo TLS che il client gestito deve utilizzare per stabilire una connessione sicura con un gestore code.

Se sul canale SVRCONN viene specificata una CipherSpec , dopo che l'handshake TLS è stato completato, il gestore code tenta di mettere in corrispondenza tale CipherSpec con la CipherSpec negoziata utilizzata dall'applicazione client. Se il gestore code non riesce a trovare una CipherSpeccorrispondente, la comunicazione non riesce con errore AMQ9631.

L'interfaccia IBM MQ.NET gestisce una tabella di associazione IBM MQ a Microsoft.NET CipherSpec . Questa tabella è utilizzata per stabilire la versione del protocollo TLS che il client desidera utilizzare per stabilire una connessione socket protetta con il gestore code. In base al valore SSLCipherSpec , la versione SSLProtocol può essere TLS 1.0o TLS 1.2, a seconda della versione di Microsoft.NET Framework che si sta utilizzando.

Assicurarsi di fornire il valore SSLCipherSpec corretto poiché la specifica di un valore non corretto potrebbe comportare l'uso di protocolli SSL 3.0 o TLS 1.0 .

Tabella 78. Tabella di associazione IBM MQ e Microsoft.NET

IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Versione TLS**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2

Tabella 78. Tabella di associazione IBM MQ e Microsoft.NET (Continua)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Versione TLS**
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.3.2 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
V 9.3.2 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

Note:

1. Deprecated Questa CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleta. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.

Concetti correlati

“Supporto TLS per il client .NET gestito” a pagina 597
Il client .NET gestito utilizza le librerie di Microsoft .NET Framework per implementare i protocolli socket sicuri TLS. La classe Microsoft System.Net.SecuritySslStream opera come un flusso su socket TCP connessi e invia e riceve dati su tale connessione socket.

Repository di chiavi del client .NET gestito
Il repository delle chiavi utilizzato dai client .NET gestiti è il keystore Windows . I certificati e le chiavi private devono essere disponibili nel keystore dell'utente o del sistema per poter essere utilizzati dall'applicazione client sia per l'identit ... che per l'affidabilit ... durante un handshake TLS.

Lato client

Nell'applicazione, è possibile impostare uno dei seguenti valori per il database delle chiavi:

- "*USER": IBM MQ.NET accede alla memorizzazione certificato dell'utente corrente per recuperare i certificati client.
- "*SYSTEM": IBM MQ.NET accede all'account del computer locale per recuperare i certificati.

I certificati del client devono essere memorizzati nell'archivio dei certificati personali dell'utente o dell'account del computer. Tutti i certificati del server (CA) devono essere memorizzati nella directory root dell'archivio certificati.

Nota: È possibile memorizzare più di un certificato in un singolo file nei formati seguenti:

- Scambio di informazioni personali - PKCS #12 (.PFX, .P12)
- Standard di sintassi del messaggio crittografico - Certificati PKCS #7 (.P7BP7B)
- Archivio certificati serializzati Microsoft (.SST)

Utilizzo di certificati per il client .NET gestito
Per i certificati client, il client .NET gestito da IBM MQ accede al keystore Windows e carica tutti i certificati del client che corrispondono all'etichetta del certificato o alla stringa.

Quando si seleziona un certificato da utilizzare, il client IBM MQ gestito .NET utilizza sempre il primo certificato corrispondente per l'handshake SSLStream TLS.

Certificati corrispondenti per etichetta certificato

Se si imposta l'etichetta del certificato, il client IBM MQ gestito .NET ricerca l'archivio certificati Windows con il nome etichetta fornito per identificare il certificato client. Carica tutti i certificati corrispondenti e utilizza il primo certificato nell'elenco. Ci sono due opzioni per impostare l'etichetta del certificato:

- L'etichetta del certificato può essere impostata sulla classe MQEnvironment che accede a MQEnvironment.CertificateLabel.
- L'etichetta del certificato può essere impostata anche in una tabella hash, fornita come parametro di input con il costruttore MQQueueManager , come mostrato nel seguente esempio.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

Il nome ("CertificateLabel") e il valore sono sensibili al maiuscolo / minuscolo.

Corrispondenza dei certificati per stringa

Se l'etichetta del certificato non è impostata, viene ricercato e utilizzato il certificato che corrisponde alla stringa "ibmwebspheremq" e l'utente attualmente collegato (in minuscolo).

Attività correlate
Connessione sicura di un client a un gestore code
Riferimenti correlati
Classe MQEnvironment .NET

SSLPEERNAME
L'attributo SSLPEERNAME viene utilizzato per controllare il DN (Distinguished Name) del certificato dal gestore code peer.

In IBM MQ.NET, le applicazioni possono utilizzare SSLPEERNAME per specificare un modello di nome distinto come mostrato nel seguente esempio.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Come per altri client IBM MQ , SSLPEERNAME è un parametro facoltativo.

Se il valore SSLPEERNAME non è impostato, il client gestito di IBM MQ.NET non esegue alcuna convalida del certificato remoto (server) e il client gestito accetta solo il certificato remoto (/server) così com' è.

Il modo in cui si imposta SSLPEERNAME dipende da quale delle offerte stack IBM MQ si sta utilizzando.

IBM MQ classes for .NET
Ci sono tre opzioni come segue.

1. Impostare MQEnvironment.SSLPeerName nella classe MQEnvironment.
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. Utilizzare il costruttore del gestore code `MQQueueManager (String queueManagerName, Hashtable properties)`. Fornire SSLPEERNAME in `Hashtable properties` come opzione 2.

XMS .NET
Impostare il nome peer SSL nel factory di connessione:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

CF
Includere il nome SslPeercome campo separato da punto e virgola nell'URI.

Riferimenti correlati
Classe MQEnvironment .NET

Reimpostazione o rinegoziazione della chiave segreta per il client .NET gestito
La classe SSLStream non supporta la reimpostazione / rinegoziazione della chiave segreta. Tuttavia, per essere congruente con altri clienti IBM MQ , il cliente IBM MQ gestito .NET consente alle applicazioni di impostare **SSLKeyResetCount**.

Quando viene raggiunto il limite, IBM MQ.NET si disconnette dal gestore code e l'applicazione ne riceve una notifica come eccezione con MQRC_CONNECTION_BROKEN come codice motivo. Le applicazioni possono scegliere di gestire l'eccezione e ristabilire le connessioni o abilitare l'opzione MQCNO_RECONNECT per IBM MQ.NET per riconnettersi automaticamente al gestore code.

L'abilitazione della funzione di riconnessione client automatica significa che, quando viene raggiunto il numero di reimpostazioni della chiave, tutte le connessioni esistenti vengono disattivate e il client IBM MQ.NET ricrea nuovamente tutte le connessioni. Per ulteriori informazioni sulla riconnessione automatica del client, consultare Ricarica automatica del client.

Concetti correlati
Reimpostazione delle chiavi segrete SSL e TLS

Controllo revoca
La classe SSLStream supporta il controllo della revoca dei certificati.

Il controllo della revoca viene eseguito automaticamente dal motore di concatenamento certificati. Ciò si applica sia a OCSP (Online Certificate Status Protocol) che a CRL (Certificate Revocation lists). La classe SSLStream utilizza la revoca del certificato che utilizza solo il server specificato nel certificato, ovvero il server è determinato dal certificato stesso. È possibile che le estensioni CDP HTTP e le richieste OCSP HTTP eseguano il proxy tramite il server proxy HTTP .

Il modo in cui si imposta il controllo della revoca dipende da quale delle offerte stack IBM MQ si sta utilizzando.

IBM MQ.NET
Il controllo revoca può essere impostato accedendo alla proprietà **`MQEnvironment.SSLCertRevocationCheck`** nel file di classe `MQEnvironment.cs` .

XMS .NET
Il controllo di revoca può essere impostato sul contesto della proprietà factory di connessione come mostrato nel seguente esempio.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

CF
Il controllo di revoca può essere impostato sull'URI utilizzando la seguente convenzione di denominazione.

```
"SslCertRevocationCheck=true"
```

Configurazione di TLS per IBM MQ .NET gestito
La configurazione di TLS per IBM MQ .NET gestito consiste nella creazione dei certificati firmatari, quindi nella configurazione del lato server, del lato client e del programma applicativo.

Informazioni su questa attività

Per configurare TLS, è necessario prima creare i certificati del firmatario appropriati. I certificati del firmatario possono essere autofirmati o certificati forniti da un'autorità di certificazione. Sebbene i certificati autofirmati possano essere utilizzati su un sistema di sviluppo, test o pre - produzione, non utilizzarli su un sistema di produzione. Su un sistema di produzione, utilizzare i certificati ottenuti da una CA (certificate authority) esterna attendibile.

Procedura

1. Creare i certificati del firmatario.
   a) Per creare certificati autofirmati, utilizzare uno dei seguenti strumenti forniti con IBM MQ :

      Utilizzare la GUI **strmqikm** oppure utilizzare **runmqckm** o **runmqakm** dalla riga comandi. Per ulteriori informazioni sull'utilizzo di questi strumenti, consultare Utilizzo di **runmqckm**, **runmqakm**e **strmqikm** per gestire i certificati digitali.
   b) Per ottenere i certificati per il gestore code e i client da una CA (Certificate Authority), seguire le istruzioni in Acquisizione di certificati personali da una CA (Certificate Authority).
2. Configurare il lato server.
   a) Configurare TLS sul gestore code, utilizzando IBM Global Security Kit (GSKit), come descritto in Connessione sicura di un client a un gestore code.
   b) Impostare gli attributi TLS del canale SVRCONN:

      - Impostare **SSLCAUTH** su `"REQUIRED/OPTIONAL"`.
      - Impostare **SSLCIPH** su un CipherSpecappropriato.

      Per ulteriori informazioni, consultare “Abilitazione di TLS per il client .NET non gestito” a pagina 596.
3. Configurare il lato client.
   a) Importare i certificati client nell'archivio certificati Windows (sotto l'account Utente / Computer).

      IBM MQ .NET accede ai certificati client dall'archivio certificati Windows , pertanto è necessario importare i propri certificati nell'archivio certificati Windows per stabilire una connessione socket sicura a IBM MQ . Per ulteriori informazioni su come accedere al keystore Windows e importare i certificati lato client, consultare Importazione o esportazione di certificati e chiavi private.
   b) Fornire CertificateLabel come descritto in Connessione di un client a un gestore code in modo sicuro.
   c) Se necessario, modificare la politica di gruppo Windows per impostare la CipherSpec, quindi, affinché gli aggiornamenti della politica di gruppo Windows diventino effettivi, riavviare il computer.
4. Configurare il programma applicativo.
   a) Impostare il valore MQEnvironment o SSLCipherSpec per indicare la connessione come una connessione protetta.

      Il valore specificato viene utilizzato per identificare il protocollo utilizzato (TLS). La serie CipherSpec deve essere una delle CipherSpecs della versione SSLProtocol supportata e può essere preferibilmente uguale a quella specificata in Windows Group Policy. (La versione SSLProtocol

supportata dipende dal framework .NET utilizzato. La versione SSLProtocol può essere TLS 1.0o TLS 1.2, a seconda della versione del framework Microsoft .NET che si sta utilizzando.)

Nota: Se il valore CipherSpec fornito dall'applicazione non è un CipherSpec noto a IBM MQ, il client IBM MQ gestito .NET lo ignora e negozia la connessione in base alla politica del gruppo del sistema Windows .

b) Impostare la proprietà SSLKeyRepository su "*SYSTEM" o "*USER".

c) Opzionale: Impostare SSLPEERNAME sul DN (distinguished name) del certificato del server.

d) Fornire CertificateLabel come descritto in Connessione di un client a un gestore code in modo sicuro.

e) Impostare eventuali ulteriori parametri facoltativi richiesti, ad esempio KeyResetCount, CertificationRevocationCheck e abilitare FIPS.

Esempi di come impostare il protocollo TLS e il repository chiavi TLS

Per .NETdi base, è possibile impostare il protocollo TLS e il repository delle chiavi TLS tramite la classe MQEnvironment come mostrato nel seguente esempio:

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

In alternativa, è possibile impostare il protocollo TLS e il repository chiavi TLS fornendo una hashtable come parte del costruttore MQQueueManager come mostrato nel seguente esempio.

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Operazioni successive

Per ulteriori informazioni sull'introduzione allo sviluppo di applicazioni TLS gestite da IBM MQ .NET , consultare“Scrittura di un'applicazione semplice” a pagina 606.

Riferimenti correlati

Classe MQEnvironment .NET

Conteggio KeyReset(MQLONG)

FIPS (Federal Information Processing Standards) per AIX, Linux, and Windows

Scrittura di un'applicazione semplice

Suggerimenti per la scrittura di una semplice applicazione TLS IBM MQ gestita .NET , inclusi esempi per impostare le proprietà SSL per le factory di connessione, creare una istanza del gestore code, una connessione, una sessione e una destinazione e inviare un messaggio di verifica.

Prima di iniziare

È necessario prima configurare TLS per IBM MQ.NET gestito come descritto in “Configurazione di TLS per IBM MQ .NET gestito” a pagina 605.

Per la configurazione del programma applicativo in .NETdi base, impostare le proprietà SSL utilizzando la classe MQEnvironment o fornendo una tabella hash come parte del costruttore MQQueueManager .

Per la configurazione del programma di applicazione in XMS .NET, impostare le proprietà SSL sul contesto delle proprietà delle factory di connessione.

Procedura

1. Impostare le proprietà SSL per i factory di connessione come mostrato nei seguenti esempi.

Esempio per IBM MQ.NET

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
properties.Add("CertificateLabel", "ibmwebspheremq");
MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
```

Esempio per XMS .NET

```
cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

2. Creare l'istanza, le connessioni, la sessione e la destinazione del gestore code come mostrato nei seguenti esempi.

   Esempio per MQ .NET

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
   MQC.MQOO_FAIL_IF_QUIESCING);
   Console.WriteLine("done");
   ```

   Esempio per XMS .NET

   ```
   connectionWMQ = cf.CreateConnection();
   // Create session
   sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

   // Create destination
   destination = sessionWMQ.CreateQueue(destinationName);

   // Create producer
   producer = sessionWMQ.CreateProducer(destination);
   ```

3. Inviare un messaggio come mostrato nei seguenti esempi.

   Esempio per MQ .NET

   ```
   // creating a message object
   message = new MQMessage();
   message.WriteString(messageString);

   // putting messages continuously
   for (int i = 1; i <= numberOfMsgs; i++)
   {
   Console.Write("Message " + i + " <" + messageString + ">.. ");
   queue.Put(message);
   Console.WriteLine("put");
   }
   ```

   Esempio per XMS .NET

   ```
   textMessage = sessionWMQ.CreateTextMessage();
   textMessage.Text = simpleMessage;
   producer.Send(textMessage);
   ```

4. Verificare la connessione TLS.

   Controllare lo stato del canale per verificare che la connessione TLS sia stata stabilita e stia funzionando correttamente.

Configurazione della traccia per SSLStream
Per catturare gli eventi di traccia e i messaggi relativi alla classe SSLStream, è necessario aggiungere una sezione di configurazione per la diagnostica del sistema al file di configurazione per l'applicazione.

Informazioni su questa attività

Nota:

Questa attività si applica solo a IBM MQ classes for .NET Framework . Il file di configurazione dell'applicazione non è supportato in IBM MQ classes for .NET (librerie.NET Standard e .NET 6).

Se non si aggiunge una sezione di configurazione per la diagnostica del sistema al file di configurazione dell'applicazione, il client IBM MQ gestito .NET non catturerà alcun evento, traccia o punto di debug relativo a TLS e alla classe SSLStream.

Nota: L'avvio della traccia IBM MQ utilizzando **strmqtrc** non cattura tutta la traccia TLS richiesta.

Procedura

1. Creare un file di configurazione dell'applicazione (App.Config) per il progetto dell'applicazione.
2. Aggiungere una sezione di configurazione della diagnostica di sistema come mostrato nel seguente esempio.

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
           </sources>
           <switches>
                 <add name="System.Net" value="Verbose"/>
                 <add name="System.Net.Sockets" value="Verbose"/>
                 <add name="System.Net.Cache" value="Verbose"/>
                 <add name="System.Security" value="Verbose"/>
                 <add name="System.Net.Security" value="Verbose"/>
           </switches>

           <sharedListeners>
                 <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
           </sharedListeners>
         <trace autoflush="true"/>
     </system.diagnostics>
```

Attenzione: Il campo `Version` della voce `add name` deve essere la versione del file .net `amqmdnet.dll` utilizzato.

Attività correlate

Traccia dei client IBM MQ classes for .NET Framework utilizzando un file di configurazione dell'applicazione

Applicazioni di esempio per l'implementazione di TLS in .NET gestito

Le applicazioni di esempio vengono fornite per mostrare l'implementazione di TLS per .NET gestito in IBM MQ classes for .NET, XMS .NET e il canale personalizzato IBM MQ per WCF.

La seguente tabella riporta l'ubicazione delle applicazioni di esempio. *MQ_INSTALLATION_PATH* rappresenta la directory di livello superiore in cui è installato IBM MQ .

Tabella 79. Ubicazione delle applicazioni campione per l'implementazione di TLS in .NET gestito

Offerta stack IBM MQ.NET	**Ubicazione dei campioni**
Di base.NET	*MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs` *MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs`
XMS .NET	*MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs` *MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs`
Canale personalizzato IBM MQ per WCF	*MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs`

Windows Utilizzo del Monitor .NET

.NET Monitor è un'applicazione simile a un controllo trigger IBM MQ .

Importante: Consultare Funzioni che possono essere utilizzate solo con l'installazione primaria su Windows per informazioni importanti.

È possibile creare componenti .NET che vengono istanziati ogni volta che un messaggio viene ricevuto su una coda monitorata e che quindi elaborano tale messaggio. .NET Monitor viene avviato dal comando **runmqdnm** e arrestato dal comando **endmqdnm** . Per dettagli su questi comandi, consultare runmqdnm e endmqdnm.

Per utilizzare .NET Monitor, si scrive un componente che implementa l'interfaccia IMQObjectTrigger , definita in amqmdnm.dll.

I componenti possono essere transazionali o non transazionali. Un componente transazionale deve ereditare da System.EnterpriseServices.ServicedComponent ed essere registrato come RequiresTransaction o SupportsTransaction. Non deve essere registrato come RequiresNew poiché .NET Monitor ha già avviato una transazione.

Il componente riceve oggetti MQQueueManager, MQQueue e MQMessage da **runmqdnm**. Può anche ricevere una stringa parametro utente se ne è stata specificata una, utilizzando l'opzione di riga comandi *-u* , quando è stato avviato `runmqdnm` . Tenere presente che il componente riceve il contenuto di un messaggio arrivato sulla coda monitorata in un oggetto MQMessage. Non deve connettersi al gestore code, aprire la coda o richiamare il messaggio stesso. Il componente deve quindi elaborare il messaggio come appropriato e restituire il controllo a .NET Monitor.

Se il componente è stato scritto come componente transazionale, viene registrato per eseguire il commit o il rollback della transazione utilizzando le funzioni fornite da System.EnterpriseServices.ServicedComponent.

Poiché il componente riceve gli oggetti MQQueueManager e MQQueue e il messaggio, dispone di informazioni di contesto complete per tale messaggio e può, ad esempio, aprire un'altra coda sullo stesso gestore code senza doversi connettere separatamente a IBM MQ.

Windows *Frammenti di codice di esempio*

Questo argomento contiene due esempi di componenti che ottengono un messaggio da .NET Monitor e lo stampano, uno utilizzando l'elaborazione transazionale e l'altro non transazionale. Un terzo esempio mostra le routine di utilità comuni, applicabili a entrambi i primi due esempi. Tutti gli esempi sono in C#.

Esempio 1: elaborazione transazionale

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

Esempio 2: elaborazione non transazionale

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/
```

```
using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

Esempio 3: routine comuni

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ---------------------------------------------------------------- */
    /* Default prefix string of the namespace.                           */
    /* ---------------------------------------------------------------- */
    private string prefixText = "dnmsamp";

    /* ---------------------------------------------------------------- */
    /* Constructor that takes the replacement prefix string to use.     */
    /* ---------------------------------------------------------------- */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ---------------------------------------------------------------- */
    /* Display an arbitrary string to the console.                      */
    /* ---------------------------------------------------------------- */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }
```

```
 /* ------------------------------------------------------------------ */
    /* Display the content of the message passed to the console.          */
    /* ------------------------------------------------------------------ */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

Compilazione dei programmi IBM MQ .NET

Comandi di esempio per compilare applicazioni .NET scritte in vari linguaggi.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Per creare un'applicazione C# utilizzando IBM MQ classes for .NET, utilizzare il seguente comando:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

Per creare un'applicazione Visual Basic utilizzando IBM MQ classes for .NET, utilizzare il seguente comando:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Per creare un'applicazione C++ gestita utilizzando IBM MQ classes for .NET, utilizzare il seguente comando:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

Per altre lingue, consultare la documentazione fornita dal fornitore della lingua.

Utilizzo del client IBM MQ .NET autonomo

Il client IBM MQ .NET offre la possibilità di comprimere e distribuire un assieme IBM MQ .NET senza dover utilizzare l'installazione completa del client IBM MQ su sistemi di produzione per l'esecuzione delle applicazioni.

Prima di iniziare

V 9.3.1 Da IBM MQ 9.3.1, la libreria client `amqmdnetstd.dll` installata nell'ubicazione predefinita si basa su .NET 6. La libreria client `amqmdnetstd.dll` basata su .NET Standard è stata spostata in una nuova ubicazione nel package di installazione client IBM MQ ed è ora disponibile nelle seguenti ubicazioni:

- Windows Su Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`
- Linux Su Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`

LTS Per IBM MQ 9.3.0 Long Term Support, la libreria `amqmdnetstd.dll` è disponibile nelle seguenti ubicazioni:

- Windows Su Windows *MQ_INSTALLATION_PATH*`\bin`. Le applicazioni di esempio sono installate in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux Su Linux *MQ_INSTALLATION_PATH*`/lib64 path`. Gli esempi .NET si trovano in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

Stabilized LTS La libreria `amqmdnet.dll` è ancora fornita, ma questa libreria è stabilizzata; in altre parole, non verranno introdotte nuove funzioni. Per le funzioni più recenti, è necessario migrare alla libreria `amqmdnetstd.dll` . Tuttavia, è possibile continuare a utilizzare la libreria `amqmdnet.dll` sulle release IBM MQ 9.1 Long Term Support o Continuous Delivery .

Informazioni su questa attività

È possibile creare le proprie applicazioni IBM MQ .NET su una macchina su cui è installato il client IBM MQ completo e successivamente comprimere l'assemblaggio di IBM MQ .NET , ossia `amqmdnetstd.dll`, insieme all'applicazione e distribuirlo su sistemi di produzione.

Le applicazioni che crei e distribuisci possono essere le applicazioni .NET tradizionali, i servizi o le applicazioni Microsoft Azure Web / Worker

In tali distribuzioni, il client di IBM MQ .NET supporta solo la modalità gestita di connettività a un gestore code. I collegamenti del server e la connettività della modalità client non gestita non sono disponibili in quanto queste due modalità richiedono un'installazione completa del client IBM MQ . Qualsiasi tentativo di utilizzare queste altre due modalità genera un'eccezione dell'applicazione.

Procedura

Riferimento all'assemblaggio client IBM MQ .NET nelle applicazioni

- Fare riferimento all'assieme `amqmdnetstd.dll` nell'applicazione nello stesso modo in cui è stato fatto per le release precedenti.

  Impostare la proprietà **CopyLocal** dell'assemblaggio `amqmdnetstd.dll` su `True` per assicurarsi che l'assemblaggio `amqmdnetstd.dll` venga copiato nella directory `bin` dell'applicazione. L'impostazione di questa proprietà consente inoltre allo strumento di creazione package dell'applicazione di comprimere i file binari richiesti per la distribuzione sui sistemi di produzione e gli ambienti cloud Microsoft Azure PaaS .

Aggiunta del supporto transazioni globali

- Assicurarsi che l'applicazione distribuisca l'applicazione di monitoraggio WMQDotnetXAMonitor sulla macchina insieme all'applicazione stessa.

Se un'applicazione utilizza la funzione di transazione globale gestita da IBM MQ .NET , deve anche distribuire WMQDotnetXAMonitor sulla macchina insieme all'applicazione stessa. Questo programma di utilità è necessario per il ripristino di tutte le transazioni in dubbio.

Avvio e arresto della traccia

- Solo per IBM MQ classes for .NET Framework , per avviare e arrestare la traccia utilizzando il file di configurazione dell'applicazione e un IBM MQ file di configurazione della traccia specifico, consultare Traccia di un client IBM MQ classes for .NET Framework utilizzando un file di configurazione dell'applicazione.

  È necessario utilizzare il file di configurazione dell'applicazione e un file di configurazione di traccia specifico di IBM MQ poiché, poiché non esiste un'installazione completa del client IBM MQ , gli strumenti standard utilizzati per avviare e arrestare la traccia **strmqtrc** e **endmqtrc**, non sono disponibili.

  Note:

  – Questo modo di generare la traccia si applica al client gestito ridistribuibile .NET e al client .NET autonomo. Vedere .NET application runtime - Windows solo.

  – Il file di configurazione dell'applicazione non è supportato in IBM MQ classes for .NET (librerie.NET Standard e .NET 6). Per abilitare la funzione di traccia per IBM MQ classes for .NET (librerie.NET Standard e .NET 6), utilizzare la variabile di ambiente **MQDOTNET_TRACE_ON** . Consultare Traccia delle applicazioni IBM MQ .NET utilizzando le variabili d'ambiente.

- V 9.3.3

  Avviare e arrestare la traccia utilizzando il file `mqclient.ini` e impostando le proprietà appropriate della stanza Trace.

  Consultare Traccia delle applicazioni IBM MQ .NET con mqclient.ini.

  Da IBM MQ 9.3.3, è possibile configurare la traccia utilizzando il file `mqclient.ini` e impostando le proprietà appropriate della stanza Trace. È inoltre possibile abilitare e disabilitare la traccia in modo dinamico con il file `mqclient.ini` .

Abilitazione del reindirizzamento del collegamento nel file di configurazione dell'applicazione

- Per abilitare il riferimento di collegamento del tempo di compilazione dell'assembly IBM MQ .NET a una versione più recente dell'assembly, aggiungere la proprietà <dependentAssembly> al file di configurazione dell'applicazione.

  Il seguente frammento di esempio nel file `app.config` reindirizza un'applicazione che è stata compilata utilizzando la versione IBM MQ 8.0.0 Fix Pack 2 (`8.0.0.2`) dell'assembly IBM MQ .NET , ma in seguito è stato applicato un fix pack, IBM MQ 8.0.0 Fix Pack 3, che ha aggiornato l'assembly IBM MQ.NET a `8.0.0.3`.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

Concetti correlati

“InstallazioneIBM MQ classes for .NET” a pagina 557
IBM MQ classes for .NET, inclusi gli esempi, sono stati installati con IBM MQ su Windows e Linux

Client ridistribuibili

.NET runtime applicazione - solo Windows

Attività correlate

“Utilizzo dell'applicazione WMQDotnetXAMonitor” a pagina 576
Il client IBM MQ .NET fornisce un'applicazione XA Monitor, WmqDotnetXAMonitor, che è possibile utilizzare per ripristinare tutte le transazioni distribuite incomplete. L'applicazione XAMonitor WmqDotnetstabilisce una connessione al gestore code in cui le transazioni sono in dubbio e quindi risolve la transazione in base ai parametri impostati.

Traccia delle applicazioni IBM MQ .NET

V 9.3.0 OutboundSNI proprietà

È possibile impostare la proprietà **OutboundSNI** in un'applicazione utilizzando una proprietà o una variabile di ambiente.

Da IBM MQ 9.3.0, è possibile impostare MQC.OUTBOUND_SNI_PROPERTY nell'applicazione, utilizzando una tabella hash quando si utilizza la classe `MQQueueManager` per connettersi al gestore code.

MQC.OUTBOUND_SNI_PROPERTY assume i valori seguenti:

- MQC.OUTBOUND_SNI_CHANNEL, che corrisponde a "CHANNEL"
- MQC.OUTBOUND_SNI_HOSTNAME, che corrisponde a "HOSTNAME"
- MQC.OUTBOUND_SNI_ASTERISK, che corrisponde a "*"

Inoltre, è possibile impostare la proprietà **OutboundSNI** utilizzando la variabile di ambiente MQOUTBOUND_SNI, che assume i seguenti valori:

- CHANNEL
- HOSTNAME
- *

e impostare il valore **OutboundSNI** nel file App.`config` , come con qualsiasi altra proprietà `mqclient.ini` .

Nota: Il valore predefinito della proprietà è MQC.OUTBOUND_SNI_CHANNEL se non è impostato alcun valore specifico.

L'ordine di precedenza per l'impostazione della proprietà **OutboundSNI** nel nodo gestito è:

1. proprietà Livello applicazione
2. Variabile di ambiente

Per la proprietà **OutboundSNI** nel nodo non gestito, è supportata solo `mqclient.ini` .

Le proprietà impostate nel file App.`config` sono applicabili solo per le applicazioni .NET Framework .

Se si fornisce un valore non valido a livello dell'applicazione o nel file App.`config` , viene emesso il codice di ritorno MQRC_OUTBOUND_SNI_NOT_VALID.

Se si imposta una variabile di ambiente non valida o si fornisce un valore non valido nel file `mqclient.ini` , viene utilizzato il valore predefinito `CHANNEL` .

OutboundSNI e più certificati

IBM MQ utilizza l'intestazione SNI per fornire più funzionalità di certificati. Se un'applicazione si connette a un canale IBM MQ configurato per utilizzare un diverso certificato tramite il campo CERTLABL, l'applicazione deve connettersi con un'impostazione **OutboundSNI** di CHANNEL.

Se un'applicazione con un'impostazione **OutboundSNI** diversa da CHANNEL si connette ad un canale con un'etichetta di certificato configurata, l'applicazione viene rifiutata con un MQRC_SSL_INITIALIZATION_ERROR e un messaggio AMQ9673 viene stampato nei log degli errori del gestore code.

Per ulteriori informazioni su come IBM MQ fornisce la funzionalità di più certificati, consultare Come IBM MQ fornisce la funzionalità di più certificati .

Sviluppo di applicazioni XMS .NET

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) fornisce un'API (application programming interface) denominata XMS che ha la stessa serie di interfacce di Java Message Service (JMS) API. IBM MQ Message Service Client (XMS) for .NET contiene un'implementazione completamente gestita di XMS, che può essere utilizzata da qualsiasi linguaggio conforme a .NET .

Informazioni su questa attività

XMS supporta:

- Messaggistica point-to-point
- Pubblicazione/sottoscrizione della messaggistica
- Consegna sincrona dei messaggi
- Consegna di messaggi asincroni

Un'applicazione XMS può scambiare messaggi con i seguenti tipi di applicazione:

- Un'applicazione XMS
- Un'applicazione IBM MQ classes for JMS
- Un'applicazione IBM MQ nativa
- Un'applicazione JMS che utilizza il provider di messaggistica predefinito IBM MQ

Un'applicazione XMS può connettersi e utilizzare le risorse di uno dei seguenti server di messaggistica:

IBM MQ Gestore code
L'applicazione può connettersi in modalità bind o client.

WebSphere Application Server service integration bus
L'applicazione può utilizzare una connessione TCP/IP diretta oppure può utilizzare HTTP su TCP/IP.

IBM Integration Bus
I messaggi vengono trasportati tra l'applicazione e il broker utilizzando WebSphere MQ Real-Time Transport. I messaggi possono essere consegnati all'applicazione utilizzando WebSphere MQ Multicast Transport.

Connettendosi a un gestore code IBM MQ , un'applicazione XMS può utilizzare WebSphere MQ Enterprise Transport per comunicare con IBM Integration Bus. In alternativa, un'applicazione XMS può pubblicare e sottoscrivere collegandosi a IBM MQ.

Da IBM MQ 9.1.1, IBM MQ supporta .NET Core per le applicazioni in ambienti Windows . Per ulteriori informazioni, consultare “InstallazioneIBM MQ classes for XMS .NET” a pagina 620.

Da IBM MQ 9.1.2, IBM MQ supporta .NET Core per le applicazioni in ambienti Linux .

Da IBM MQ 9.1.4, XMS .NET le applicazioni gestite sono in grado di bilanciare automaticamente le connessioni tra i gestori code con cluster. Sono supportate sia le librerie .NET Framework che .NET Standard . Per ulteriori informazioni, vedi Informazioni sui cluster uniformi e Bilanciamento automatico dell'applicazione.

Attività correlate
Come contattare il supporto IBM
Risoluzione dei problemi di XMS .NETproblems

Stili di messaggistica supportati da XMS

XMS supporta gli stili di messaggistica point-to-point e di pubblicazione / sottoscrizione.

Gli stili di messaggistica sono anche chiamati domini di messaggistica.

Messaggistica point-to-point

Una forma comune di messaggistica point-to-point utilizza l'accodamento. Nel caso più semplice, un'applicazione invia un messaggio a un'altra applicazione identificando, implicitamente o esplicitamente, una coda di destinazione. Il sistema di messaggistica e accodamento sottostante riceve il messaggio dall'applicazione mittente e instrada il messaggio alla coda di destinazione. L'applicazione ricevente può quindi richiamare il messaggio dalla coda.

Se il sistema di messaggistica e accodamento sottostante contiene IBM Integration Bus, IBM Integration Bus potrebbe replicare un messaggio e instradare le copie del messaggio a code differenti. Di conseguenza, più di un'applicazione può ricevere il messaggio. IBM Integration Bus potrebbe anche trasformare un messaggio e aggiungervi dati.

Una caratteristica fondamentale della messaggistica point - to - point è che un'applicazione inserisce un messaggio in una coda locale quando invia un messaggio. Il sistema di messaggistica e accodamento sottostante determina a quale coda di destinazione viene inviato il messaggio. L'applicazione ricevente richiama il messaggio dalla coda di destinazione.

Pubblicazione/sottoscrizione della messaggistica

Nella messaggistica di pubblicazione / sottoscrizione, esistono due tipi di applicazione: publisher e sottoscrittore.

Un *publisher* fornisce informazioni sotto forma di messaggi di pubblicazione. Quando un publisher pubblica un messaggio, specifica un argomento che identifica l'oggetto delle informazioni all'interno del messaggio.

Un *sottoscrittore* è un consumatore delle informazioni pubblicate. Un sottoscrittore specifica gli argomenti a cui è interessato creando sottoscrizioni.

Il sistema di pubblicazione / sottoscrizione riceve le pubblicazioni dai publisher e le sottoscrizioni dai sottoscrittori. Instrada le pubblicazioni ai sottoscrittori. Un sottoscrittore riceve le pubblicazioni solo sugli argomenti sottoscritti.

Una caratteristica chiave della messaggistica di pubblicazione / sottoscrizione è che un publisher identifica un argomento quando pubblica un messaggio. Non identifica i sottoscrittori. Se un messaggio viene pubblicato su un argomento per cui non sono presenti sottoscrittori, nessuna applicazione riceve il messaggio.

Un'applicazione può essere sia un autore (publisher) che un sottoscrittore (subscriber).

Il modello oggetto XMS

L'API XMS è un'interfaccia orientata agli oggetti. Il modello oggetto XMS è basato sul modello oggetto JMS 1.1 .

Classi XMS principali

Le classi XMS principali o i tipi di oggetto sono i seguenti:

ConnectionFactory
Un oggetto `ConnectionFactory` contiene una serie di parametri per una connessione. Un'applicazione utilizza `ConnectionFactory` per creare una connessione. Un'applicazione può fornire i parametri al runtime e creare un oggetto `ConnectionFactory` . In alternativa, i parametri di connessione possono essere memorizzati in un repository di oggetti gestiti. Un'applicazione può richiamare un oggetto dal repository e creare un oggetto `ConnectionFactory` da esso.

Connessione
Un oggetto `Connection` incapsula una connessione attiva da un'applicazione a un server di messaggistica. Un'applicazione utilizza una connessione per creare sessioni.

Destinazione
Un'applicazione invia o riceve messaggi utilizzando un oggetto `Destination` . Nel dominio di pubblicazione / sottoscrizione, un oggetto `Destination` incapsula un argomento e, nel dominio point-to-point, un oggetto `Destination` incapsula una coda. Un'applicazione può fornire i parametri

per creare un oggetto `Destination` al runtime. In alternativa, è possibile creare un oggetto `Destination` da una definizione di oggetto memorizzata nel repository di oggetti gestiti.

Sessione
Un oggetto `Session` è un contesto a thread singolo per l'invio e la ricezione di messaggi. Un'applicazione utilizza un oggetto `Session` per creare oggetti `Message`, `MessageProducere` `MessageConsumer` .

Messaggio
Un oggetto `Message` incapsula l'oggetto `Message` che un'applicazione invia utilizzando un oggetto `MessageProducer` o che riceve utilizzando un oggetto `MessageConsumer` .

MessageProducer
Un oggetto `MessageProducer` viene utilizzato da un'applicazione per inviare messaggi a una destinazione.

MessageConsumer
Un oggetto `MessageConsumer` viene utilizzato da un'applicazione per ricevere i messaggi inviati ad una destinazione.

Oggetti XMS e relative relazioni

Figura 52 a pagina 618 mostra i principali tipi di oggetto XMS : ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message e Destination. Un'applicazione utilizza una produzione connessioni per creare una connessione e una connessione per creare sessioni. L'applicazione può quindi utilizzare una sessione per creare messaggi, produttori di messaggi e utenti di messaggi. L'applicazione utilizza un producer di messaggi per inviare messaggi a una destinazione e utilizza un consumer di messaggi per ricevere i messaggi inviati a una destinazione.

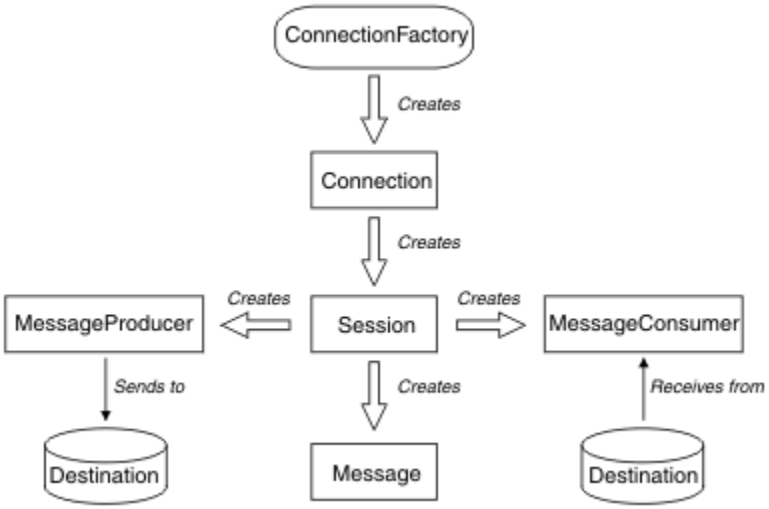


Figura 52. Oggetti XMS e relative relazioni

In XMS .NET, le classi XMS sono definite come una serie di interfacce .NET . Quando si codificano le applicazioni XMS .NET , sono necessarie solo le interfacce dichiarate.

Il modello di oggetto XMS si basa sulle interfacce indipendenti dal dominio descritte in Java Message Service Specifica, Versione 1.1. Le classi specifiche del dominio, come `Topic`, `TopicPublishere` `TopicSubscriber`, non vengono fornite.

Attributi e proprietà degli oggetti XMS

Un oggetto XMS può avere attributi e proprietà, che sono caratteristiche dell'oggetto, implementati in modi diversi:

Attributi
Una caratteristica dell'oggetto che è sempre presente e occupa la memoria, anche se l'attributo non ha un valore. In questo senso, un attributo è simile a un campo in una struttura dati a lunghezza fissa. Una caratteristica distintiva degli attributi è che ogni attributo ha i suoi metodi per impostare e ottenere il suo valore.

Proprietà
Una proprietà di un oggetto è presente e occupa memoria solo dopo che è stato impostato il relativo valore. Una proprietà non può essere eliminata o la sua memoria non può essere ripristinata dopo che il suo valore è stato impostato. È possibile modificarne il valore. XMS fornisce una serie di metodi generici per l'impostazione e il richiamo dei valori delle proprietà.

oggetti gestiti

Utilizzando gli oggetti gestiti, è possibile gestire le impostazioni di connessione utilizzate dalle applicazioni client da gestire da un repository centrale. Un'applicazione richiama le definizioni degli oggetti dal repository centrale e le usa per creare oggetti `ConnectionFactory` e `Destination` . Utilizzando gli oggetti gestiti, è possibile disaccoppiare le applicazioni dalle risorse che utilizzano al runtime.

Ad esempio, le applicazioni XMS possono essere scritte e verificate con oggetti gestiti che fanno riferimento a un insieme di connessioni e destinazioni in un ambiente di test. Quando le applicazioni vengono distribuite, gli oggetti gestiti possono essere modificati per configurare le applicazioni in modo che facciano riferimento a connessioni e destinazioni nell'ambiente di produzione.

XMS supporta due tipi di oggetti gestiti:

- Un oggetto `ConnectionFactory` , utilizzato dalle applicazioni per stabilire la connessione iniziale al server.
- Un oggetto `Destination` , utilizzato dalle applicazioni per specificare la destinazione per i messaggi che vengono inviati e l'origine dei messaggi che vengono ricevuti. Una destinazione è un argomento o una coda sul server a cui si connette un'applicazione.

Lo strumento di amministrazione **`JMSAdmin`** viene fornito con IBM MQ. Viene utilizzato per creare e gestire oggetti gestiti in un repository centrale di oggetti gestiti.

Gli oggetti gestiti nel repository possono essere utilizzati dalle applicazioni IBM MQ classes for JMS e XMS . Le applicazioni XMS possono utilizzare gli oggetti `ConnectionFactory` e `Destination` per connettersi a un IBM MQ gestore code. Un amministratore può modificare le definizioni di oggetto contenute nel repository senza influenzare il codice dell'applicazione.

Il seguente diagramma mostra in che modo un'applicazione XMS generalmente utilizza gli oggetti gestiti. Il lato sinistro del diagramma mostra un repository contenente le definizioni di ConnectionFactory e degli oggetti di destinazione gestiti mediante una console di amministrazione. La parte destra del diagramma mostra un'applicazione XMS che ricerca le definizioni degli oggetti nel repository e utilizza queste definizioni di oggetti durante la connessione a un server di messaggistica.

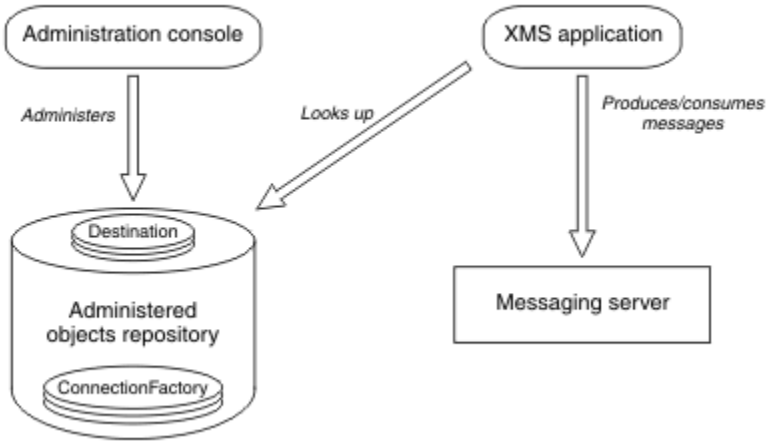


Figura 53. Utilizzo tipico di oggetti gestiti da un'applicazione XMS

Il modello di messaggio XMS

Il modello di messaggio XMS è uguale al modello di messaggio IBM MQ classes for JMS .

In particolare XMS implementa gli stessi campi di intestazione del messaggio e proprietà del messaggio che IBM MQ classes for JMS implementa:

- JMS campi di intestazione. Questi campi hanno nomi che iniziano con il prefisso JMS.
- JMS proprietà definite. Questi campi hanno proprietà i cui nomi iniziano con il prefisso JMSX.
- IBM proprietà definite. Questi campi hanno proprietà i cui nomi iniziano con il prefisso JMS_IBM_.

Di conseguenza, le applicazioni XMS possono scambiare messaggi con le applicazioni IBM MQ classes for JMS . In ogni messaggio, alcuni dei campi di intestazione e delle proprietà vengono impostati dall'applicazione e altri vengono impostati da XMS o IBM MQ classes for JMS. Alcuni dei campi impostati da XMS o IBM MQ classes for JMS vengono impostati quando il messaggio viene inviato e altri quando viene ricevuto. I campi di intestazione e le proprietà vengono propagati con un messaggio tramite un server di messaggistica, dove appropriato. Vengono resi disponibili a tutte le applicazioni che ricevono il messaggio.

Concetti correlati

IBM MQ classes for JMS

Linux Windows InstallazioneIBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET, inclusi gli esempi, sono installati con IBM MQ su Windows e Linux.

Da IBM MQ 9.2.0, Microsoft.NET Core 3.1 è la versione minima richiesta per l'esecuzione di IBM MQ classes for XMS .NET Standard.

V 9.3.0 V 9.3.0 Da IBM MQ 9.3.0, IBM MQ supporta le applicazioni .NET 6 utilizzando IBM MQ classes for XMS .NET Standard. Se si utilizza un'applicazione .NET Core 3.1 , è possibile eseguire questa applicazione con una piccola modifica nel file `csproj` , impostando `targetframeworkversion` su `"net6.0"`, senza alcuna ricompilazione richiesta.

V 9.3.1 IBM MQ 9.3.1 fornisce una libreria client XMS .NET creata rispetto a .NET 6 come framework di destinazione. Da IBM MQ 9.3.1, Microsoft .NET 6.0 è la versione minima richiesta per eseguire le applicazioni utilizzando le librerie IBM MQ create utilizzando .NET 6 come framework di destinazione.

V 9.3.1 Da IBM MQ 9.3.1, la libreria client XMS .NET creata utilizzando .NET Standard è disponibile in una nuova cartella `netstandard2.0` e la libreria client XMS .NET creata utilizzando .NET 6

come framework di destinazione è disponibile in *MQ_INSTALLATION_PATH*/bin su Windows e in *MQ_INSTALLATION_PATH*/lib64 su Linux.

amqmxmsstd.dll libreria

V 9.3.1 Da IBM MQ 9.3.1, la libreria amqmxmsstd.dll è disponibile nelle seguenti ubicazioni

Libreria creata utilizzando .NET Standard 2.0 come framework di destinazione

- Windows Su Windows *MQ_INSTALLATION_PATH*\bin\netstandard2.0.
- Linux Su Linux *MQ_INSTALLATION_PATH*\lib64\netstandard2.0.

Deprecated Queste librerie sono obsolete e IBM intende rimuoverle nelle release future.

Libreria creata utilizzando .NET 6 come framework di destinazione

- Windows Su Windows *MQ_INSTALLATION_PATH*\bin. Le applicazioni di esempio sono installate in *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.
- Linux Su Linux *MQ_INSTALLATION_PATH*\lib64. Gli esempi .NET si trovano in *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.

LTS Per IBM MQ 9.3.0 Long Term Support, la libreria IBM MQ classes for XMS .NET Standard , amqmxmsstd.dll, è disponibile nelle seguenti ubicazioni:

- Windows Su Windows *MQ_INSTALLATION_PATH*\bin. Le applicazioni di esempio sono installate in *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/xms.
- Linux Su Linux *MQ_INSTALLATION_PATH*/lib64 path. Gli esempi .NET si trovano in *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/xms.

Per ulteriori informazioni, consultare “InstallazioneIBM MQ classes for .NET” a pagina 557.

Attenzione: V 9.3.1 Deprecated Da IBM MQ 9.3.1, le IBM MQ .NET librerie client create utilizzando .NET Standard 2.0 come framework di destinazione sono obsolete e le applicazioni che fanno riferimento a tali librerie generano un'avvertenza CS0618 durante il tempo di compilazione.

Stabilized Tutte le librerie IBM.XMS.* sono ancora fornite, ma queste librerie sono stabilizzate; in altre parole, non verranno introdotte nuove funzioni. Per tutte le funzioni più recenti, è necessario migrare alla libreria amqmxmsstd.dll . Tuttavia, è possibile continuare ad utilizzare le librerie esistenti sulle release IBM MQ 9.1 Long Term Support o Continuous Delivery .

V 9.3.1 Se un'applicazione .NET Framework viene compilata utilizzando amqmdnetstd.dll o amqmxmsstd.dll da una versione precedente a IBM MQ 9.3.1 e la stessa applicazione viene eseguita utilizzando librerie client basate su .NET 6 IBM MQ , il seguente tipo di eccezione FileLoad viene generato da .NET:

```
Eccezione rilevata: System.IO.FileLoadException: Impossibile caricare il file o l'assembly
'amqmdnetstd, Version =x.x.x.x, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' o
una delle sue dipendenze. La definizione manifest dell'assembly individuato non corrisponde al
riferimento dell'assieme. (Eccezione da HRESULT: 0x80131040)

Nome file: ' amqmdnetstd, Version =x.x.x.x, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
```

Per risolvere questo errore, le librerie presenti in *MQ_INSTALLATION_PATH*/bin/netstandard2.0 devono essere copiate nella directory da cui è in esecuzione l'applicazione .NET Framework .

Da IBM MQ 9.2.0, i IBM MQ classes for XMS .NET Standard sono disponibili per il download dal repository NuGet . Il pacchetto NuGet contiene sia la libreria amqmxmsstd.dll che la libreria amqmdnetstd.dll . amqmxmsstd.dll dipende da amqmdnetstd.dll e, mentre si impacchettano le applicazioni XMS .NET Core, amqmxmsstd.dll e amqmdnetstd.dll devono essere impacchettati insieme all'applicazione

XMS .NET Core. Per ulteriori informazioni, fare riferimento a "Download di IBM MQ classes for XMS .NET dal repository NuGet" a pagina 623.

dspmqver comando

È possibile utilizzare il comando **dspmqver** per visualizzare le informazioni sulla versione e sulla creazione del componente .NET Core .

Confronto tra IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET (librerie.NET Standard e .NET 6)

La seguente tabella elenca le funzioni per IBM MQ classes for XMS .NET Framework rispetto alle funzioni per IBM MQ classes for XMS .NET (librerie.NET Standard e .NET 6).

Tabella 80. Differenze tra IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET (librerie.NET Standard e .NET 6)

Funzione	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET (librerie.NET Standard e .NET 6)**
Nomi classe (API)	Tutte le classi rimangono le stesse in ogni rete.	Tutte le classi rimangono le stesse in ogni rete.
Sistema operativo	Windows	Windows Contenitori Docker Linux macOS
File `app.config` (file di configurazione per abilitare la traccia nel client ridistribuibile)	Il file `app.config` viene utilizzato per abilitare la traccia per il pacchetto ridistribuibile.	`app.config` non è supportata. Utilizzare le variabili di ambiente.
Traccia	Per tracciare il client XMS .NET , è possibile utilizzare le variabili di ambiente esistenti, come la variabile di ambiente **XMS_TRACE_ON** utilizzata per abilitare la traccia. Per ulteriori informazioni, consultare Configurazione della traccia di XMS .NET utilizzando le variabili di ambiente XMS. Per i client ridistribuibili, il file `app.config` può essere utilizzato per abilitare la traccia.	Per tracciare il client XMS .NET , è possibile utilizzare le variabili di ambiente esistenti, come la variabile di ambiente **XMS_TRACE_ON** utilizzata per abilitare la traccia. Per ulteriori informazioni, consultare Configurazione della traccia di XMS .NET utilizzando le variabili di ambiente XMS.
Modalità di trasporto	Gestito, non gestito e bind	Gestita

Tabella 80. Differenze tra IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET (librerie.NET Standard e .NET 6) (Continua)

Funzione	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET (librerie.NET Standard e .NET 6)
TLS	Il keystore Windows viene utilizzato per memorizzare i certificati.	Su Windows, il keystore deve essere utilizzato per memorizzare i certificati. I valori consentiti sono *USER o *SYSTEM. In base all'input, il client IBM MQ .NET guarda all'archivio chiavi Windows dell'utente corrente o a livello di sistema. Su Linux, si consiglia di utilizzare la classe X509Store per installare i certificati e .NET Core installa i certificati nella seguente ubicazione: "`.dotnet/corefx/cryptography/x509stores`".
CCDT	Supportato	Supportato e le impostazioni del percorso CCDT sono le stesse delle classi .NET Framework.
Riconnessione automatica del client	Supportato	Supportato
Transazioni distribuite	Supportato	Non supportato
Installazione delle DLL (dynamic linked library) nella GAC (global assembly cache)	Le DLL sono installate nella GAC come parte dell'installazione di IBM MQ.	Le DLL non sono installate nella GAC come parte dell'installazione di IBM MQ.
Supporto per tipi di connessione WMQ, WPM e RTT	Supporta tipi di connessione WMQ, WPM e RTT	Supporto solo per WMQ
Oggetti gestiti da JNDI	Supporta LDAP e FileSystem	Supporta solo FileSystem

V 9.3.0 V 9.3.0 Da IBM MQ 9.3.0, per eseguire IBM MQ classes for XMS .NET Framework , è necessario installare Microsoft.NET Framework V4.7.2 o versioni successive.

Attività correlate

“Utilizzo di applicazioni di esempio XMS” a pagina 630
Le applicazioni di esempio XMS .NET forniscono una panoramica delle funzioni comuni di ogni API. È possibile utilizzarli per verificare l'installazione e l'impostazione del server di messaggistica e per creare le proprie applicazioni.

Linux Windows Download di IBM MQ classes for XMS .NET dal repository NuGet

I IBM MQ classes for XMS .NET sono disponibili per il download dal repository NuGet , in modo che possano essere facilmente utilizzati dagli sviluppatori .NET .

Informazioni su questa attività

NuGet è il gestore pacchetti per piattaforme di sviluppo Microsoft incluso .NET. Gli strumenti del client NuGet forniscono la possibilità di produrre e utilizzare pacchetti. Un pacchetto NuGet è un singolo file

compresso con l'estensione `.nupkg` che contiene il codice compilato (DLL), altri file correlati a quel codice e un manifest descrittivo che include informazioni come il numero di versione del pacchetto.

È possibile scaricare il package `IBMXMSDotnetClient` NuGet , che contiene sia la libreria `amqmdnetstd.dll` che la libreria `amqmxmsstd.dll` , dalla Galleria NuGet , che è il repository di package centrale utilizzato da tutti gli autori e i consumer di package.

Nota: V 9.3.1 Da IBM MQ 9.3.1, il pacchetto NuGet contiene le librerie create utilizzando .NET Standard 2.0 e .NET 6 come framework di destinazione. Le librerie per .NET Standard 2.0 sono disponibili nella cartella `netstandard2.0` e le librerie per .NET 6 sono disponibili nella cartella `net6.0` .

Esistono tre metodi per scaricare il pacchetto `IBMXMSDotnetClient` :

- Utilizzando Microsoft Visual Studio. NuGet è distribuito come estensione Microsoft Visual Studio . Da Microsoft Visual Studio 2012, NuGet è preinstallato per default.
- Dalla riga di comando utilizzando NuGet Package Manager o la CLI .NET .
- Utilizzando un browser Web.

Come per il pacchetto ridistribuibile, si abilita la traccia utilizzando la variabile di ambiente **XMS_TRACE_ON**.

Procedura

- Per scaricare il pacchetto `IBMXMSDotnetClient` utilizzando la IU di Package Manager in Microsoft Visual Studio, completa la seguente procedura:
  a) Fare clic con il tasto destro del mouse su un progetto .NET e fare clic su **Gestisci package di nuget**.
  b) Fare clic sulla scheda **Sfoglia** e cercare `"IBMXMSDotnetClient"`.
  c) Selezionare il package e fare clic su **Installa**.

  Durante l'installazione, Package Manager fornisce le informazioni sull'avanzamento sotto forma di istruzioni della console.
- Per scaricare il pacchetto `IBMXMSDotnetClient` dalla riga di comando, scegliere una delle seguenti opzioni:
  - Utilizzando NuGet Package Manager, immettere il seguente comando:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    Durante l'installazione, Package Manager fornisce le informazioni sull'avanzamento sotto forma di istruzioni della console. È possibile reindirizzare l'emissione a un file di log.
  - Utilizzando la CLI .NET , immetti il seguente comando:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- Utilizzando un browser Web, scaricare il package `IBMXMSDotnetClient` da https://www.nuget.org/packages/IBMXMSDotnetClient.

Concetti correlati

“InstallazioneIBM MQ classes for .NET” a pagina 557
IBM MQ classes for .NET, inclusi gli esempi, sono stati installati con IBM MQ su Windows e Linux

Informazioni sulla licenza di IBM MQ client per .NET

Attività correlate

“Download di IBM MQ classes for .NET dal repository NuGet” a pagina 562
I IBM MQ classes for .NET sono disponibili per il download dal repository NuGet , in modo che possano essere facilmente utilizzati da .NET Developers.

Impostazione dell'ambiente del server di messaggistica

Gli argomenti in questa sezione descrivono come impostare l'ambiente del server di messaggistica per consentire alle applicazioni di XMS di connettersi a un server.

Informazioni su questa attività

Per le applicazioni che si connettono a un gestore code IBM MQ , è necessario il client IBM MQ (o il gestore code per la modalità di bind).

Attualmente non esistono prerequisiti per le applicazioni che utilizzano una connessione in tempo reale a un broker.

È necessario configurare l'ambiente del server di messaggistica prima di eseguire qualsiasi applicazione XMS , incluse le applicazioni di esempio fornite con XMS.

Questa sezione contiene i seguenti argomenti:

- "Configurazione del gestore code e del broker per un'applicazione che si connette a un gestore code IBM MQ" a pagina 627
- "InstallazioneIBM MQ classes for XMS .NET" a pagina 620
- "Configurazione di un broker per un'applicazione che utilizza una connessione in tempo reale a un broker" a pagina 628
- "Configurazione del service integration bus per un'applicazione che si connette a WebSphere Application Server" a pagina 629

Listener di messaggi in XMS .NET

Un listener di messaggi viene utilizzato per ricevere messaggi in modo asincrono. A differenza della chiamata `MessageConsumer.receive()` , il listener di messaggi non blocca il thread di chiamata, ma consegna i messaggi a un metodo di callback specificato dall'applicazione, in genere il metodo `onMessage` .

La consegna del messaggio inizia una volta richiamato il metodo `Connection.Start()` . La consegna dei messaggi può essere arrestata e ripresa in qualsiasi momento utilizzando rispettivamente i metodi `Connection.Stop()` e `Connection.Start()` .

Una volta richiamato il metodo `Connection.Start()` dopo aver impostato un listener di messaggi su almeno un consumer in una sessione, tale sessione diventa una sessione asincrona. Una volta che una sessione diventa asincrona, non è possibile richiamare alcun metodo sincrono XMS .NET . ad esempio, `MessageProducer.Send()`. In questo modo si genera un'eccezione con codice motivo IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500).

Chiamate sincrone in una sessione asincrona

`Session.Close` è l'unica chiamata sincrona consentita in una sessione asincrona. Le applicazioni possono anche effettuare chiamate sincrone (tranne `Session.Close`) utilizzando il metodo callback del listener dei messaggi, ossia il metodo `onMessage` .

Oltre a queste due opzioni, è necessario arrestare la connessione utilizzando il metodo `Connection.Stop()` per un'applicazione per effettuare qualsiasi chiamata sincrona. Una volta effettuate le chiamate, è necessario riprendere nuovamente la connessione utilizzando il metodo `Connection.Start()` . che riavvia la consegna del messaggio.

Quanti utenti di messaggi asincroni possono avere una sessione?

Una sessione può avere più utenti di messaggi asincroni. Ma in qualsiasi momento un messaggio viene consegnato a un solo consumatore. Ciò significa, in pratica, che quando arriva un secondo messaggio mentre XMS .NET ha richiamato il metodo `onMessage()` di un utente per consegnare il primo messaggio, il secondo messaggio non verrà consegnato a un utente nella sessione fino a quando non viene restituito il metodo `onMessage()` .

Il secondo messaggio viene recapitato a un consumer nella sessione solo dopo la restituzione del metodo onMessage() . Ciò si verifica perché una sessione gestisce la consegna dei messaggi ai consumer utilizzando un solo thread. Ciò significa che è possibile consegnare un solo messaggio alla volta e che il consumatore può essere qualsiasi.

Se un'applicazione richiede la consegna di messaggi simultanei, ossia, tutti i consumer devono ricevere messaggi contemporaneamente, l'applicazione deve creare più sessioni e ciascuno deve avere un consumer di messaggi asincrono.

I seguenti esempi mostrano questa funzione in modo più chiaro.

Nel primo esempio, ci sono più utenti di messaggi asincroni in una sessione. Una sessione S dispone di tre consumer di messaggi asincroni: AMC1, AMC2 e AMC3 che ricevono messaggi da tre diverse destinazioni Q1, Q2 e Q3.

Poiché esiste una sola sessione S, esiste solo un thread di consegna dei messaggi per consegnare i messaggi ai consumer AMC1, AMC2e AMC3. Quando la sessione consegna il messaggio a AMC1, gli altri due consumer AMC2 e AMC3 attendono, anche se ci sono messaggi in Q2 e Q3 pronti per la consegna.

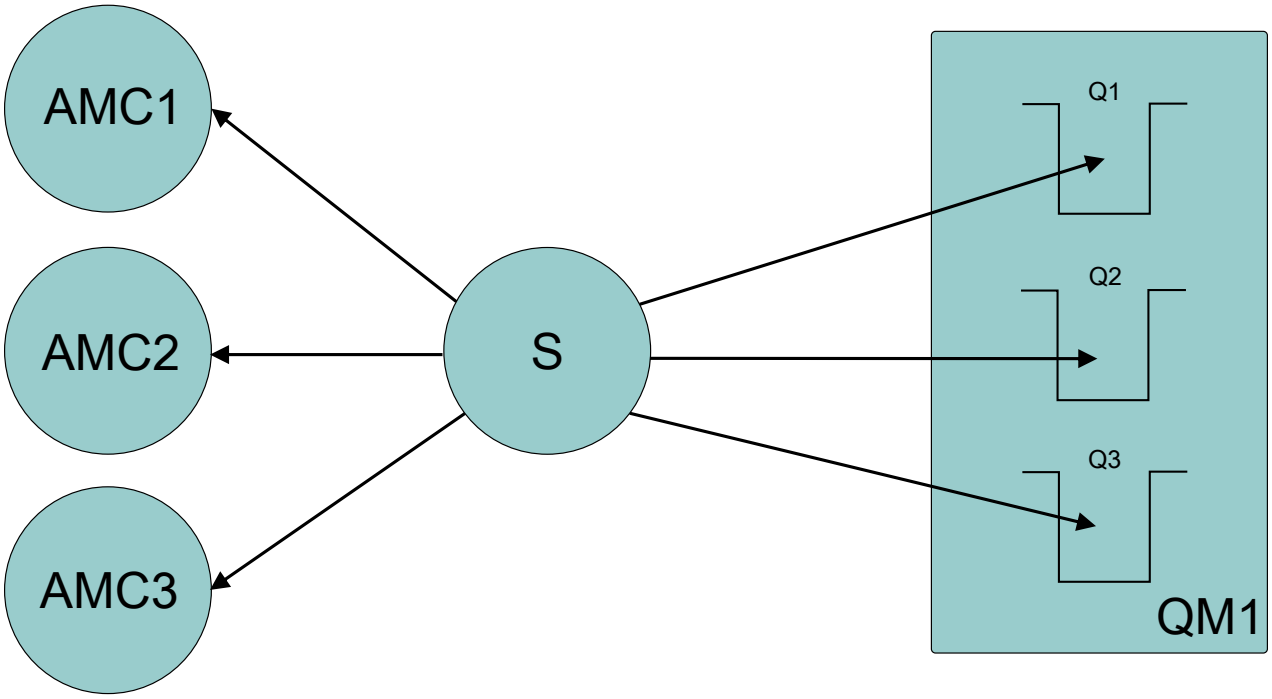


Figura 54. Una sessione con tre utenti di messaggi asincroni

Nel secondo caso ci sono più sessioni S1, S2e S3, ognuna con un utente di messaggi asincrono AMC1, AMC2e AMC3 rispettivamente. Poiché esiste un consumer per ciascuna sessione, i messaggi vengono consegnati ai consumer contemporaneamente.

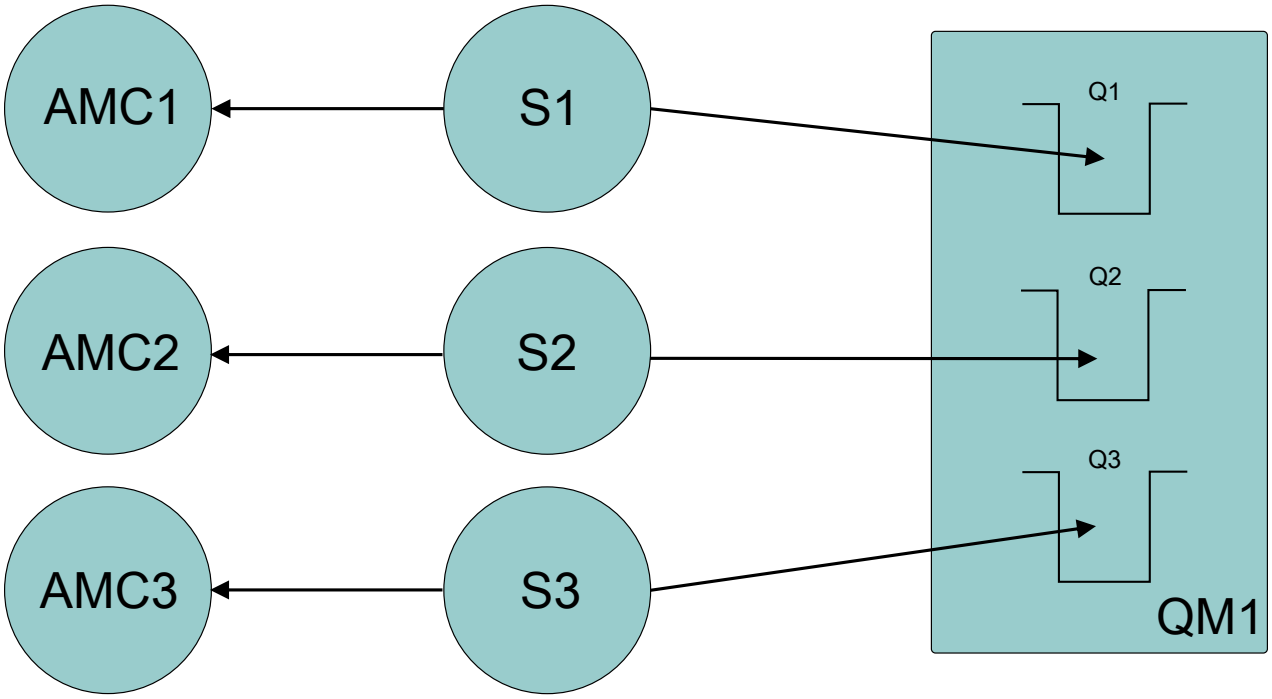


Figura 55. Sessioni multiple, ognuna con un utente di messaggi asincroni

Ciò mostra che se si richiede la consegna simultanea di messaggi, sono necessarie più sessioni.

Configurazione del gestore code e del broker per un'applicazione che si connette a un gestore code IBM MQ

Questa sezione presuppone che si stia utilizzando IBM WebSphere MQ 7.0.1o versioni successive. Prima di poter eseguire un'applicazione che si connette ad un gestore code IBM MQ , è necessario configurarlo. Per un'applicazione di pubblicazione / sottoscrizione, è richiesta una configurazione aggiuntiva se si utilizza l'interfaccia di pubblicazione / sottoscrizione accodata.

Prima di iniziare

XMS funziona con IBM Integration Bus o WebSphere Message Broker 6.1 o versioni successive

Prima di avviare questa attività, effettuare le seguenti operazioni:

- Assicurarsi che l'applicazione abbia accesso a un gestore code in esecuzione.
- Se l'applicazione è un'applicazione di pubblicazione / sottoscrizione e utilizza l'interfaccia di pubblicazione / sottoscrizione accodata, assicurarsi che l'attributo **PSMODE** sia impostato su ENABLED sul gestore code.
- Verificare che l'applicazione utilizzi una factory di connessione le cui proprietà siano impostate in maniera appropriata per connettersi al gestore code. Se l'applicazione è un'applicazione di pubblicazione / sottoscrizione, assicurarsi che le proprietà di factory di connessione appropriate siano impostate per l'utilizzo del broker. Per ulteriori informazioni sulle proprietà di una factory di connessione, vedere Proprietà di ConnectionFactory.

Informazioni su questa attività

Configurare il gestore code e il Broker per eseguire le applicazioni XMS nello stesso modo in cui si configura il Gestore code e l'interfaccia di pubblicazione / sottoscrizione accodata per eseguire le applicazioni IBM MQ JMS . La seguente procedura riepiloga le operazioni da effettuare.

Procedura

1. Sul Gestore code, creare le code necessarie all'applicazione.

   Per una panoramica su come creare le code, consultare Definizione delle code.

   Se l'applicazione è un'applicazione di pubblicazione / sottoscrizione e utilizza l'interfaccia di pubblicazione / sottoscrizione accodata che necessita dell'accesso alle code di sistema IBM MQ classes for JMS , attendere il passo 4a prima di creare le code.
2. Concedere all'ID utente associato alla propria applicazione l'autorità di connettersi al gestore code e l'autorizzazione appropriata per accedere alle code.

   Per una panoramica sull'autorizzazione, consultare Protezione. Se l'applicazione si connette al gestore code in modalità client, consultare anche Client e server.
3. Se l'applicazione si connette al gestore code in modalità client, assicurarsi che un canale di connessione server sia definito nel gestore code e che un listener sia avviato.

   Non è necessario eseguire questa procedura per ciascuna applicazione che si connette al gestore code. Una definizione di canale di connessione server e un listener possono supportare tutte le applicazioni che si collegano in modalità client.
4. Se l'applicazione è un'applicazione di pubblicazione / sottoscrizione e utilizza l'interfaccia di pubblicazione / sottoscrizione accodata, attenersi alla seguente procedura.
   a) Sul gestore code, creare le code del sistema IBM MQ classes for JMS eseguendo lo script dei comandi MQSC forniti con IBM MQ. Assicurarsi che l'ID utente associato a IBM Integration Bus o WebSphere Message Broker disponga dell'autorizzazione per accedere alle code.

      Per informazioni su dove trovare lo script e su come eseguirlo, consultare Utilizzo di IBM MQ classes for Java.

      Eseguire questa operazione solo una volta per il gestore code. La stessa serie di code di sistema IBM MQ classes for JMS può supportare tutte le applicazioni XMS e IBM MQ classes for JMS che si connettono al gestore code.

   b) Concedere all'ID utente associato all'applicazione l'autorità di accedere alle code di sistema IBM MQ classes for JMS .

      Per informazioni sulle autorizzazioni necessarie all'ID utente, consultare Utilizzo IBM MQ classes for JMS.

   c) Per un broker IBM Integration Bus o WebSphere Message Broker, creare e distribuire un flusso di messaggi per servire la coda in cui le applicazioni inviano i messaggi che pubblicano.

      Il flusso di messaggi di base comprende un nodo di elaborazione messaggi MQInput per leggere i messaggi pubblicati e un nodo di elaborazione messaggi Publication per pubblicare i messaggi.

      Per informazioni su come creare e distribuire un flusso di messaggi, consultare la documentazione del prodotto IBM Integration Bus o WebSphere Message Broker disponibile in IBM Integration Bus pagina web della libreria della documentazione del prodotto.

      Non è necessario eseguire questa operazione se un flusso di messaggi adatto è già distribuito nel broker.

Risultati

È ora possibile avviare l'applicazione.

Configurazione di un broker per un'applicazione che utilizza una connessione in tempo reale a un broker

Prima di poter eseguire un'applicazione che utilizza una connessione in tempo reale a un broker, è necessario configurare tale broker.

Prima di iniziare

Prima di avviare questa attività, effettuare le seguenti operazioni:

- Assicurarsi che l'applicazione abbia accesso a un broker in esecuzione.
- Assicurarsi che l'applicazione utilizzi una factory di connessione le cui proprietà siano impostate in maniera appropriata per una connessione in tempo reale a un broker. Per ulteriori informazioni sulle proprietà di una factory di connessione, vedere Proprietà di ConnectionFactory.

Informazioni su questa attività

Configurare un broker per eseguire applicazioni XMS nello stesso modo in cui si configura un broker per eseguire applicazioni IBM MQ classes for JMS . La seguente procedura riepiloga le operazioni da effettuare:

Procedura

1. Creare e distribuire un flusso di messaggi per leggere i messaggi dalla porta TCP/IP su cui un broker è in ascolto e pubblicare i messaggi.

   È possibile effettuare questa operazione in uno dei seguenti modi:

   - Creare un flusso di messaggi che contiene un nodo di elaborazione messaggi **Real-timeOptimizedFlow** .
   - Creare un flusso di messaggi che contenga un nodo di elaborazione messaggi **Real-timeInput** e un nodo di elaborazione messaggi Publication.

   È necessario configurare il nodo **Real-timeOptimizedFlow** o **Real-timeInput** per essere in ascolto sulla porta utilizzata per le connessioni in tempo reale. In XMS, il numero di porta predefinito per le connessioni in tempo reale è 1506.

   Non è necessario eseguire questa operazione se un flusso di messaggi adatto è già distribuito nel broker.
2. Se si desidera che i messaggi vengano consegnati all'applicazione utilizzando IBM MQ classes for JMS, configurare il broker per abilitare il multicast. Configurare gli argomenti che devono essere abilitati al multicast, specificando una QoS (quality of service) affidabile per quegli argomenti che richiedono il multicast affidabile.
3. Se l'applicazione fornisce un ID utente e una password quando si connette al broker e si desidera che il broker autentichi l'applicazione utilizzando queste informazioni, configurare il server dei nomi utente e il broker per l'autenticazione della password di tipo telnet semplice.

Risultati

È ora possibile avviare l'applicazione.

Configurazione del service integration bus per un'applicazione che si connette a WebSphere Application Server

Prima di poter eseguire un'applicazione che si connette a un SIB (Service Integration Bus) WebSphere Application Server service integration technologies , è necessario configurare l'integrazione servizi nello stesso modo in cui si configura il SIB (Service Integration Bus) per eseguire le applicazioni JMS che utilizzano il provider di messaggistica predefinito.

Prima di iniziare

Prima di avviare questa attività, è necessario effettuare le seguenti operazioni:

- Verificare che sia creato un bus di messaggistica e che il proprio server sia stato aggiunto al bus come membro del bus.
- Assicurarsi che l'applicazione abbia accesso a un SIB (Service Integration Bus) che contiene almeno un motore di messaggistica in esecuzione.

- Se è richiesta l'operazione HTTP , è necessario definire un canale di trasporto in entrata del motore di messaggistica HTTP . Per impostazione predefinita, i canali per SSL e TCP vengono definiti durante l'installazione del server.
- Verificare che l'applicazione utilizzi una factory di connessione le cui proprietà siano impostate in maniera appropriata per connettersi al SIB (Service Integration Bus) utilizzando un server di avvio. Le informazioni minime richieste sono:
  - L'endpoint del provider, che descrive l'ubicazione e il protocollo da utilizzare quando si negozia una connessione al server di messaggistica (ossia, tramite il server di avvio). Nella sua forma più semplice, per un server installato con le impostazioni predefinite, l'endpoint di fornitura può essere impostato sul nome host del server.
  - Il nome del bus attraverso cui vengono inviati i messaggi.

  Per ulteriori informazioni sulle proprietà di una factory di connessione, vedere Proprietà di ConnectionFactory.

Informazioni su questa attività

Qualsiasi coda o spazio argomento richiesto deve essere definito. Per impostazione predefinita, uno spazio argomento denominato Default.Topic.Space viene definito durante l'installazione del server ma, se sono necessari ulteriori spazi argomento, è necessario creare tali spazi. Non è necessario predefinire singoli argomenti all'interno di uno spazio argomento, poiché il server crea un'istanza di questi singoli argomenti in modo dinamico come richiesto.

La seguente procedura riepiloga le operazioni da effettuare.

Procedura

1. Creare le code necessarie all'applicazione per la messaggistica point - to - point.
2. Creare eventuali spazi argomenti aggiuntivi necessari all'applicazione per la messaggistica di pubblicazione / sottoscrizione.

Risultati

È ora possibile avviare l'applicazione.

Utilizzo di applicazioni di esempio XMS

Le applicazioni di esempio XMS .NET forniscono una panoramica delle funzioni comuni di ogni API. È possibile utilizzarli per verificare l'installazione e l'impostazione del server di messaggistica e per creare le proprie applicazioni.

Informazioni su questa attività

Se si ha bisogno di aiuto per creare le proprie applicazioni, è possibile utilizzare le applicazioni di esempio come punto di partenza. Sia l'origine che una versione compilata vengono fornite per ciascuna applicazione. Esaminare il codice di origine di esempio e identificare i passi chiave per creare ciascun oggetto richiesto per l'applicazione (ConnectionFactory, Connection, Session, Destination e un Producer o un Consumer o entrambi) e per impostare le proprietà specifiche necessarie per specificare il funzionamento dell'applicazione. Per ulteriori informazioni, consultare “Scrittura di applicazioni XMS” a pagina 633. Gli esempi sono soggetti a modifiche nelle release future di XMS.

La seguente tabella mostra le serie di applicazioni di esempio (una per ogni API) fornite con XMS.

Tabella 81. Applicazioni di esempio per XMS .NET

Nome del campione	Descrizione
SampleConsumerCS	Un'applicazione del consumatore di messaggi che prende i messaggi da una coda o sottoscrive un argomento.

Tabella 81. Applicazioni di esempio per XMS .NET (Continua)	
Nome del campione	**Descrizione**
SampleProducerCS	Un'applicazione del produttore di messaggi che produce messaggi su una coda o su un argomento.
SampleConfigCS	Un'applicazione di configurazione che è possibile utilizzare per creare un repository di oggetti gestiti basato su file. L'applicazione contiene una produzione connessioni e una destinazione per le specifiche impostazioni di connessione. Questo repository di oggetti gestiti può quindi essere utilizzato con ciascuna delle applicazioni consumer e producer di esempio.

Gli esempi che supportano le stesse funzioni nelle varie API presentano differenze sintattiche.

- Le applicazioni consumer e producer di messaggi di esempio supportano entrambe le seguenti funzioni:
  - Le connessioni a IBM MQ, IBM Integration Bus (utilizzando una connessione in tempo reale a un broker) e un WebSphere Application Server service integration bus
  - Ricerche di repository di oggetti amministrati utilizzando l'interfaccia di contesto iniziale
  - Connessioni a code (IBM MQ e WebSphere Application Server service integration bus) e argomenti (IBM MQ, connessione in tempo reale a broker e WebSphere Application Server service integration bus)
  - Messaggi di base, byte, mappa, oggetto, flusso e testo
- L'applicazione consumatore di messaggi di esempio supporta le modalità di ricezione sincrone e asincrona e le istruzioni SQL Selector.
- L'applicazione produttore di messaggi di esempio supporta le modalità di consegna persistenti e non persistenti.

Gli esempi possono funzionare in uno dei due modi:

Modalità semplice
È possibile eseguire gli esempi con l'input utente minimo.

Modalità avanzata
È possibile personalizzare più finemente il modo in cui operano gli esempi.

Tutti gli esempi sono compatibili e possono quindi operare in più lingue.

Windows Da IBM MQ 9.1.1, IBM MQ supporta .NET Core per applicazioni XMS .NET in ambienti Windows . IBM MQ classes for .NET Standard, inclusi gli esempi, vengono installati per impostazione predefinita come parte dell'installazione standard di IBM MQ .

Linux Da IBM MQ 9.1.2, IBM MQ supporta anche .NET Core per le applicazioni in ambienti Linux .

Le applicazioni di esempio per XMS .NET sono installate in `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms`.

Per ulteriori informazioni, consultare “InstallazioneIBM MQ classes for XMS .NET” a pagina 620.

Esecuzione delle applicazioni di esempio .NET

È possibile eseguire le applicazioni di esempio .NET in modo interattivo in modalità semplice o avanzata oppure in modo non interattivo utilizzando file di risposta personalizzati o generati automaticamente.

Prima di iniziare

Prima di eseguire una delle applicazioni di esempio fornite, è necessario impostare l'ambiente del server di messaggistica in modo che le applicazioni possano connettersi a un server. Vedere “Impostazione dell'ambiente del server di messaggistica” a pagina 625.

Procedura

Per eseguire un'applicazione di esempio .NET, completare la seguente procedura:

Suggerimento: Quando si sta eseguendo un'applicazione di esempio, immettere? in qualsiasi momento per ottenere aiuto su cosa fare in seguito.

1. Selezionare la modalità in cui si desidera eseguire l'applicazione di esempio.

   Immettere `Advanced` o `Simple`.
2. Rispondere alle domande.

   Per selezionare il valore predefinito, visualizzato tra parentesi alla fine della domanda, premere Invio. Per selezionare un valore diverso, immettere il valore appropriato e premere Invio.

   Ecco una domanda di esempio:

   ```
   Enter connection type [wpm]:
   ```

   In questo caso, il valore predefinito è `wpm` (connessione a un WebSphere Application Server service integration bus).

Risultati

Quando si eseguono le applicazioni di esempio, i file di risposta vengono generati automaticamente nella directory di lavoro corrente. I nomi dei file di risposta sono nel formato *`connection_type`*-*`sample_type`*`.rsp`; ad esempio, `wpm-producer.rsp`. Se necessario, è possibile utilizzare il file di risposta generato per rieseguire l'applicazione di esempio con le stesse opzioni, in modo da non dover immettere nuovamente le opzioni.

Attività correlate

Creazione delle applicazioni di esempio .NET
Quando si crea un'applicazione .NET di esempio, viene creata una versione eseguibile dell'esempio scelto.

Creazione di applicazioni personalizzate
Si creano le proprie applicazioni come si creano le applicazioni di esempio.

Creazione delle applicazioni di esempio .NET

Quando si crea un'applicazione .NET di esempio, viene creata una versione eseguibile dell'esempio scelto.

Prima di iniziare

Installare il compilatore appropriato. Questa attività presuppone che Microsoft Visual Studio 2012 sia installato e che si abbia familiarità con l'utilizzo.

Procedura

Per creare un'applicazione di esempio .NET , completare la seguente procedura:

1. Fare clic sul file della soluzione `Samples.sln` fornito con gli esempi .NET .
2. Fare clic con il tasto destro del mouse sulla soluzione `Esempi` nella finestra Esplora soluzioni e selezionare **Crea soluzione**.

Risultati

Un programma eseguibile viene creato nella sottocartella appropriata dell'esempio, `bin/Debug` o `bin/Release`, a seconda della configurazione scelta. Questo programma ha lo stesso nome della cartella, con suffisso CS. Ad esempio, se si sta creando la versione C# dell'applicazione di esempio del produttore del messaggio, `SampleProducerCS.exe` viene creato nella cartella `SampleProducer` .

Attività correlate

Esecuzione delle applicazioni di esempio .NET

È possibile eseguire le applicazioni di esempio .NET in modo interattivo in modalità semplice o avanzata oppure in modo non interattivo utilizzando file di risposta personalizzati o generati automaticamente.

Creazione di applicazioni personalizzate
Si creano le proprie applicazioni come si creano le applicazioni di esempio.

“Creazione di applicazioni personalizzate” a pagina 633
Si creano le proprie applicazioni come si creano le applicazioni di esempio.

Creazione di applicazioni personalizzate

Si creano le proprie applicazioni come si creano le applicazioni di esempio.

Prima di iniziare

Installare il compilatore appropriato. Questa attività presuppone che Microsoft Visual Studio 2012 sia installato e che si abbia familiarità con l'utilizzo.

Procedura

- Creare l'applicazione .NET , come descritto in “Creazione delle applicazioni di esempio .NET” a pagina 632.

  Per ulteriori informazioni su come creare le applicazioni, utilizzare i makefile forniti per ciascuna applicazione di esempio.

  Suggerimento: Per facilitare la diagnosi dei problemi in caso di errore, potrebbe essere utile compilare le applicazioni con i simboli inclusi.

Attività correlate

Esecuzione delle applicazioni di esempio .NET
È possibile eseguire le applicazioni di esempio .NET in modo interattivo in modalità semplice o avanzata oppure in modo non interattivo utilizzando file di risposta personalizzati o generati automaticamente.

Creazione delle applicazioni di esempio .NET
Quando si crea un'applicazione .NET di esempio, viene creata una versione eseguibile dell'esempio scelto.

Scrittura di applicazioni XMS

Gli argomenti contenuti in questa sezione forniscono informazioni utili per la scrittura di applicazioni XMS in generale.

Informazioni su questa attività

Questa sezione contiene concetti generali per la scrittura di applicazioni XMS . Consultare anche “Scrittura di applicazioni XMS .NET” a pagina 652 per informazioni specifiche sulla creazione di applicazioni XMS .NET .

Da IBM MQ 9.2.0, il numero di XMS.NET sono state significativamente ridotte, per un totale di cinque. Le cinque librerie di collegamento dinamico sono:

- `IBM.XMS.dll` - include tutti i messaggi in lingua nazionale
- `IBM.XMS.Comms.RMM.dll`
- Tre librerie di collegamento dinamico della politica:
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

V 9.3.0 V 9.3.0 Removed XMS .NETLa messaggistica multicast (tramite RMM) era diventata obsoleta per IBM MQ 9.2 ed è stata rimossa in IBM MQ 9.3.

Questa sezione contiene i seguenti argomenti:

- “Il modello di threading” a pagina 635
- “ConnectionFactories e oggetti Connection” a pagina 635
- “Sessioni” a pagina 637
- “Destinazioni” a pagina 640
- “Produttori di messaggi” a pagina 642
- “Utenti del messaggio” a pagina 643
- “Browser coda” a pagina 646
- “Richiedenti” a pagina 647
- “Eliminazione oggetto” a pagina 647
- “XMS Tipi primitivi” a pagina 648
- “Conversione implicita di un valore di proprietà da un tipo di dati a un altro” a pagina 649
- “Iteratori” a pagina 651
- “CCSID (Coded character set identifier)” a pagina 651
- “Codici di errore e di eccezione XMS” a pagina 652
- “Creazione di applicazioni personalizzate” a pagina 633

Windows Utilizzo del modello di progetto IBM MQ XMS .NET

Il client di IBM MQ XMS .NET offre la possibilità di utilizzare un template di progetto per assistere l'utente nello sviluppo delle applicazioni XMS .NET Core .

Prima di iniziare

È necessario disporre di Microsoft Visual Studio 2017o versione successiva e di .NET Core 2.1 sul sistema.

È necessario copiare il template XMS .NET dal

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip

nella directory

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

dove:

- *& MQ_INSTALL_ROOT* è la directory root dell'installazione
- *& USER_HOME_DIRECTORY* è la tua home directory.

È necessario arrestare e riavviare Microsoft Visual Studio per prelevare il template.

Informazioni su questa attività

Il modello di progetto XMS .NET include del codice comune che è possibile utilizzare per sviluppare le applicazioni. Con il codice integrato, puoi connetterti al gestore code IBM MQ ed eseguire un'operazione put o get semplicemente modificando le proprietà nel codice integrato.

Procedura

1. Aprire Microsoft Visual Studio.
2. Fare clic su **File**, seguito da **Nuovo** e quindi da **Progetto**.
3. Nella finestra *Crea un nuovo progetto*, selezionare `IBM XMS .NET Client App (.NET Core)` e fare clic su **Avanti**.

4. Nella finestra *Configura il nuovo progetto* , modificare il *Nome progetto* del progetto e fare clic su **Crea** per creare il progetto XMS .NET .

   `XMSDotnetApp.cs` è il file creato insieme al file del progetto. Questo file contiene il codice che si connette al gestore code ed esegue un'operazione di invio e ricezione.

   Le proprietà di connessione sono impostate su valori predefiniti:

   - WMQ_CONNECTION_NAME_LIST è impostato su *localhost (1414)*
   - XMSC.WMQ_CHANNEL è impostato su *DOTNET.SVRCONN*

   La coda è impostata su *Q1*ed è possibile modificare queste proprietà di conseguenza.
5. Compilare ed eseguire l'applicazione.

Concetti correlati

Componenti e funzioni di IBM MQ

.NET runtime applicazione - solo Windows

Il modello di threading

Le regole generali gestiscono il modo in cui un'applicazione a più thread può utilizzare gli oggetti XMS .

- Solo gli oggetti dei seguenti tipi possono essere utilizzati contemporaneamente su thread differenti:
  - ConnectionFactory
  - Connessione
  - Dati ConnectionMeta
  - Destinazione
- Un oggetto Session può essere utilizzato solo su un singolo thread alla volta.

Le eccezioni a tali regole vengono indicate da voci etichettate "Thread context" nelle definizioni di interfaccia dei metodi in IBM Message Service Client for .NET reference.

ConnectionFactories e oggetti Connection

Un oggetto ConnectionFactory fornisce un template che un'applicazione utilizza per creare un oggetto Connection. L'applicazione utilizza l'oggetto Connection per creare un oggetto Session.

Per .NET, l'applicazione XMS utilizza innanzitutto un oggetto XMSFactoryFactory per ottenere un riferimento a un oggetto ConnectionFactory appropriato al tipo di protocollo richiesto. Questo ConnectionFactory può quindi produrre connessioni solo per quel tipo di protocollo.

Un'applicazione XMS può creare più connessioni e un'applicazione a più thread può utilizzare un singolo oggetto Connection contemporaneamente su più thread. Un oggetto Connection incapsula una connessione di comunicazione tra un'applicazione e un server di messaggistica.

Una connessione ha diversi scopi:

- Quando un'applicazione crea una connessione, è possibile autenticarla.
- Un'applicazione può associare un identificativo client univoco ad una connessione. L'identificativo client viene utilizzato per supportare sottoscrizioni durevoli nel dominio di pubblicazione / sottoscrizione. L'identificativo client può essere impostato in due modi:

  Il modo preferito per assegnare un identificatore client di connessioni è quello di configurare un oggetto ConnectionFactory specifico del client utilizzando le proprietà e assegnarlo in modo trasparente alla connessione che crea.

  Un modo alternativo di assegnare un identificatore client consiste nell'utilizzare un valore specifico del provider impostato sull'oggetto Connection. Questo valore non sovrascrive l'identificativo che è stato configurato amministrativamente. Viene fornito per il caso in cui non esiste alcun identificativo specificato amministrativamente. Se un identificativo specificato amministrativamente esiste, un tentativo di sovrascriverlo con un valore specifico del provider causa l'emissione di un'eccezione. Se un'applicazione imposta esplicitamente un identificativo, deve farlo immediatamente dopo la creazione

della connessione e prima che venga eseguita qualsiasi altra operazione sulla connessione; altrimenti, viene generata un'eccezione.

Un'applicazione XMS in genere crea una connessione, una o più sessioni e un numero di produttori di messaggi e di utenti di messaggi.

La creazione di una connessione è relativamente dispendiosa in termini di risorse di sistema poiché implica la creazione di una connessione di comunicazione e potrebbe anche comportare l'autenticazione dell'applicazione.

Modalità di connessione avviata e arrestata

Una connessione può operare in modalità avviata o arrestata.

Quando un'applicazione crea una connessione, la connessione è in modalità arrestata. Quando la connessione è in modalità arrestata, l'applicazione può inizializzare le sessioni e può inviare messaggi ma non può riceverli, in modo sincrono o asincrono.

Un'applicazione può avviare una connessione richiamando il metodo `Start Connection` . Quando la connessione è in modalità avviata, l'applicazione può inviare e ricevere messaggi. L'applicazione può quindi arrestare e riavviare la connessione richiamando i metodi Arresta connessione e `Start Connection` .

Chiusura connessione

Un'applicazione chiude una connessione richiamando il metodo Chiudi connessione. Quando un'applicazione chiude una connessione, XMS esegue le seguenti operazioni:

- Chiude tutte le sessioni associate alla connessione ed elimina alcuni oggetti associati a queste sessioni. Per ulteriori informazioni su quali oggetti vengono eliminati, consultare “Eliminazione oggetto” a pagina 647. Allo stesso tempo, XMS esegue il rollback di tutte le transazioni attualmente in corso all'interno delle sessioni.
- Termina la connessione di comunicazione con il server di messaggi.
- Rilascia la memoria e le altre risorse interne utilizzate dalla connessione.

XMS non riconosce la ricezione di alcun messaggio che non è stato possibile confermare durante una sessione, prima di chiudere la connessione. Per ulteriori informazioni sul riconoscimento della ricezione dei messaggi, consultare “Riconoscimento messaggio” a pagina 638.

Gestione eccezione

Le eccezioni XMS .NET sono tutte derivate da System.Exception. Per ulteriori informazioni, consultare “Gestione degli errori in .NET” a pagina 656.

Connessione a un SIB (Service Integration Bus)

Un'applicazione XMS può connettersi a un SIB WebSphere Application Server utilizzando una connessione TCP/IP diretta o utilizzando HTTP su TCP/IP.

Il protocollo HTTP può essere utilizzato in situazioni in cui non è possibile una connessione TCP/IP diretta. Una situazione comune è quando si comunica attraverso un firewall, ad esempio quando due aziende si scambiano messaggi. L'utilizzo di HTTP per comunicare tramite firewall viene spesso definito *HTTP tunneling*. Il tunneling HTTP , tuttavia, è intrinsecamente più lento rispetto all'utilizzo di una connessione TCP/IP diretta poiché le intestazioni HTTP aggiungono in modo significativo alla quantità di dati trasferiti e poiché il protocollo HTTP richiede più flussi di comunicazione rispetto a TCP/IP.

Per creare una connessione TCP/IP, è possibile che un'applicazione utilizzi un factory di connessione la cui proprietà XMSC_WPM_TARGET_TRANSPORT_CHAIN è impostata su XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC. Questo è il valore predefinito della proprietà. Se la connessione viene creata correttamente, la proprietà XMSC_WPM_CONNECTION_PROTOCOL della connessione è impostata su XMSC_WPM_CP_TCP.

Per creare una connessione che utilizzi HTTP, è necessario che un'applicazione utilizzi una factory di connessione la cui proprietà XMSC_WPM_TARGET_TRANSPORT_CHAIN è impostata sul nome di una catena di trasporto in entrata, configurata per utilizzare un canale di trasporto HTTP . Se la connessione è stata creata correttamente, la proprietà XMSC_WPM_CONNECTION_PROTOCOL della connessione è impostata su XMSC_WPM_CP_HTTP. Per informazioni su come configurare le catene di trasporto, consultare Configurazione delle catene di trasporto nella documentazione di WebSphere Application Server .

Un'applicazione ha una scelta simile di protocolli di comunicazione quando si connette a un server di avvio. La proprietà XMSC_WPM_PROVIDER_ENDPOINTS di una factory di connessione è una sequenza di uno o più indirizzi endpoint dei server bootstrap. Il componente della catena di trasporto bootstrap di ogni indirizzo endpoint può essere XMSC_WPM_BOOTSTRAP_TCP, per una connessione TCP/IP a un server bootstrap o XMSC_WPM_BOOTSTRAP_HTTP, per una connessione che utilizza HTTP.

Sessioni

Una sessione è un contesto a thread singolo per inviare e ricevere messaggi.

Un'applicazione può utilizzare una sessione per creare messaggi, produttori di messaggi, utenti di messaggi, browser di code e destinazioni temporanee. Un'applicazione può anche utilizzare una sessione per eseguire transazioni locali.

Un'applicazione può creare più sessioni, in cui ogni sessione produce e consuma messaggi indipendentemente dalle altre sessioni. Se due utenti di messaggi in sessioni separate (o anche nella stessa sessione) sottoscrivono lo stesso argomento, ciascuno riceve una copia di qualsiasi messaggio pubblicato su tale argomento.

A differenza di un oggetto Connection, un oggetto Session non può essere utilizzato contemporaneamente su thread differenti. Solo il metodo di chiusura sessione di un oggetto Session può essere richiamato da un thread diverso da quello che l'oggetto Session sta utilizzando al momento. Il metodo Chiudi sessione termina una sessione e rilascia tutte le risorse di sistema assegnate alla sessione.

Se un'applicazione deve elaborare i messaggi contemporaneamente su più di un thread, l'applicazione deve creare una sessione su ciascun thread e quindi utilizzare tale sessione per qualsiasi operazione di invio o ricezione all'interno di tale thread.

Sessioni transazionali

Le applicazioni XMS possono eseguire transazioni locali. Una *transazione locale* è una transazione che implica modifiche solo alle risorse del gestore code o del service integration bus a cui è connessa l'applicazione.

Le informazioni contenute in questo argomento sono rilevanti solo se un'applicazione si connette a un gestore code IBM MQ o a un SIB (Service Integration Bus) WebSphere Application Server . Le informazioni non sono rilevanti per una connessione in tempo reale a un broker.

Per eseguire transazioni locali, un'applicazione deve prima creare una sessione sottoposta a transazione richiamando il metodo di creazione sessione di un oggetto Connection, specificando come parametro che la sessione viene sottoposta a transazione. Successivamente, tutti i messaggi inviati e ricevuti all'interno della sessione vengono raggruppati in una sequenza di transazioni. Una transazione termina quando l'applicazione esegue il commit o il rollback dei messaggi inviati e ricevuti dall'inizio della transazione.

Per eseguire il commit di una transazione, un'applicazione richiama il metodo Commit dell'oggetto Session. Quando viene eseguito il commit di una transazione, tutti i messaggi inviati all'interno della transazione diventano disponibili per il recapito ad altre applicazioni e tutti i messaggi ricevuti all'interno della transazione vengono riconosciuti in modo che il server di messaggistica non tenti di consegnarli nuovamente all'applicazione. Nel dominio point - to - point, il server di messaggistica rimuove anche i messaggi ricevuti dalle code.

Per eseguire il rollback di una transazione, l'applicazione richiama il metodo di rollback dell'oggetto Session. Quando viene eseguito il rollback di una transazione, tutti i messaggi inviati all'interno della transazione vengono eliminati dal server di messaggistica e tutti i messaggi ricevuti all'interno della

transazione diventano nuovamente disponibili per la consegna. Nel dominio point - to - point, i messaggi ricevuti vengono reinseriti nelle loro code e diventano nuovamente visibili ad altre applicazioni.

Una nuova transazione viene avviata automaticamente quando un'applicazione crea una sessione sottoposta a transazione o richiama il metodo Commit o Rollback. Pertanto, una sessione transazionale ha sempre una transazione attiva.

Quando un'applicazione chiude una sessione transatta, si verifica un rollback implicito. Quando un'applicazione chiude una connessione, si verifica un rollback implicito per tutte le sessioni transazionali della connessione.

Una transazione è interamente contenuta all'interno di una sessione transatta. Una transazione non può estendere le sessioni. Ciò significa che non è possibile per un'applicazione inviare e ricevere messaggi in due o più sessioni transazionali e quindi eseguire il commit o il rollback di tutte queste azioni come una singola transazione.

Concetti correlati

Riconoscimento messaggio
Ogni sessione non sottoposta a transazione dispone di una modalità di riconoscimento che determina il modo in cui vengono riconosciuti i messaggi ricevuti dall'applicazione. Sono disponibili tre modalità di riconoscimento e la scelta della modalità di riconoscimento influisce sulla progettazione dell'applicazione.

Distribuzione del messaggio
XMS supporta le modalità persistenti e non persistenti di consegna dei messaggi e la consegna asincrona e sincrona dei messaggi.

Riconoscimento messaggio

Ogni sessione non sottoposta a transazione dispone di una modalità di riconoscimento che determina il modo in cui vengono riconosciuti i messaggi ricevuti dall'applicazione. Sono disponibili tre modalità di riconoscimento e la scelta della modalità di riconoscimento influisce sulla progettazione dell'applicazione.

Le informazioni contenute in questo argomento sono rilevanti solo se un'applicazione si connette a un gestore code IBM MQ o a un SIB (Service Integration Bus) WebSphere Application Server . Le informazioni non sono rilevanti per una connessione in tempo reale a un broker.

XMS utilizza lo stesso meccanismo per confermare la ricezione dei messaggi utilizzati da JMS.

Se non viene eseguita la transazione di una sessione, il modo in cui vengono riconosciuti i messaggi ricevuti dall'applicazione è determinato dalla modalità di riconoscimento della sessione. Le tre modalità di riconoscimento sono descritte nei seguenti paragrafi:

RICONOSCIUTO_AUTO_XMSC_
La sessione riconosce automaticamente ogni messaggio ricevuto dall'applicazione.

Se i messaggi vengono consegnati in modo sincrono all'applicazione, la sessione conferma la ricezione di un messaggio ogni volta che una chiamata di ricezione viene completata correttamente.

Se l'applicazione riceve correttamente un messaggio, ma un errore impedisce il verificarsi del riconoscimento, il messaggio diventa nuovamente disponibile per la consegna. L'applicazione deve quindi essere in grado di gestire un messaggio che viene riconsegnato.

XMSC_DUPS_OK_RICONOSCIMENTO
La sessione riconosce i messaggi ricevuti dall'applicazione nel momento in cui viene selezionata.

L'utilizzo di questa modalità di riconoscimento riduce la quantità di lavoro che la sessione deve eseguire, ma un malfunzionamento che impedisce la conferma di ricezione del messaggio potrebbe causare la ridisponibilità di più di un messaggio per la consegna. L'applicazione deve quindi essere in grado di gestire i messaggi riconsegnati.

CONFERMA_CLIENT_XMSC_XX_ENCODE_CASE_CAPS_LOCK_OFF
L'applicazione riconosce i messaggi ricevuti richiamando il metodo Acknowledge della classe Message.

L'applicazione può confermare la ricezione di ciascun messaggio singolarmente oppure può ricevere un batch di messaggi e richiamare il metodo Acknowledge solo per l'ultimo messaggio ricevuto.

Quando il metodo di riconoscimento viene richiamato, vengono riconosciuti tutti i messaggi ricevuti dall'ultima volta che il metodo è stato richiamato.

Insieme a una di queste modalità di riconoscimento, un'applicazione può arrestare e riavviare la consegna dei messaggi in una sessione richiamando il metodo Recover della classe Session. I messaggi la cui ricezione non era stata precedentemente confermata vengono riconsegnati. Tuttavia, potrebbero non essere consegnati nella stessa sequenza in cui sono stati precedentemente consegnati. Nel frattempo, potrebbero essere arrivati messaggi con priorità più elevata e alcuni dei messaggi originali potrebbero essere scaduti. Nel dominio point - to - point, alcuni dei messaggi originali potrebbero essere stati utilizzati da un'altra applicazione.

Un'applicazione può determinare se un messaggio viene riconsegnato esaminando il contenuto del campo di intestazione JMSRefornita del messaggio. L'applicazione esegue questa operazione richiamando il metodo Get JMSRenei della classe Message.

Concetti correlati

Sessioni transazionali
Le applicazioni XMS possono eseguire transazioni locali. Una *transazione locale* è una transazione che implica modifiche solo alle risorse del gestore code o del service integration bus a cui è connessa l'applicazione.

Distribuzione del messaggio
XMS supporta le modalità persistenti e non persistenti di consegna dei messaggi e la consegna asincrona e sincrona dei messaggi.

Distribuzione del messaggio

XMS supporta le modalità persistenti e non persistenti di consegna dei messaggi e la consegna asincrona e sincrona dei messaggi.

Modalità consegna messaggi

XMS supporta due modalità di consegna dei messaggi:

permanente
I messaggi permanenti vengono consegnati una sola volta. Un server di messaggistica prende precauzioni speciali, come la registrazione dei messaggi, per garantire che i messaggi persistenti non vadano persi in transito, anche in caso di errore.

Non permanente
I messaggi non persistenti vengono consegnati non più di una volta. I messaggi non persistenti sono meno affidabili dei messaggi persistenti perché possono essere persi in transito in caso di errore.

La scelta della modalità di consegna è un compromesso tra affidabilità e prestazioni. I messaggi non persistenti vengono generalmente trasportati più rapidamente rispetto ai messaggi persistenti.

Consegna di messaggi asincroni

XMS utilizza un thread per gestire tutte le consegne di messaggi asincroni per una sessione. Ciò significa che è possibile eseguire solo una funzione listener dei messaggi o un metodo `onMessage()` alla volta.

Se più di un consumatore di messaggi in una sessione sta ricevendo messaggi in modo asincrono e una funzione del listener di messaggi o un metodo `onMessage()` sta consegnando un messaggio a un consumatore di messaggi, qualsiasi altro consumatore di messaggi che è in attesa dello stesso messaggio deve continuare ad attendere. Anche altri messaggi in attesa di essere consegnati alla sessione devono continuare ad attendere.

Se un'applicazione richiede la consegna simultanea dei messaggi, creare più di una sessione in modo che XMS utilizzi più di un thread per gestire la consegna asincrona dei messaggi. In questo modo, è possibile eseguire contemporaneamente più di una funzione listener di messaggi o di un metodo `onMessage()`.

Una sessione non viene resa asincrona assegnando un listener di messaggi a un consumer. Una sessione diventa asincrona solo quando viene chiamato il metodo `Connection.Start`. Tutte le chiamate

sincrone sono consentite fino a quando non viene richiamato il metodo `Connection.Start` . La consegna del messaggio ai consumer inizia quando viene richiamato il `Connection.Start` .

Se le chiamate sincrone, come la creazione di un consumer o di un producer, devono essere effettuate su una sessione asincrona, è necessario richiamare `Connection.Stop` . È possibile riprendere una sessione richiamando il metodo `Connection.Start` per avviare la consegna dei messaggi. L'unica eccezione è il thread di consegna dei messaggi di sessione, che è il thread che consegna i messaggi alla funzione di callback. Questo thread può effettuare qualsiasi chiamata sulla sessione (tranne una chiamata di chiusura) nella funzione di callback del messaggio.

Nota: In modalità non gestita, la chiamata MQDISC all'interno di una funzione di call-back non è supportata dal client IBM MQ .NET . Quindi, l'applicazione client non può creare o chiudere le sessioni all'interno del callback MessageListener in modalità di ricezione asincrona. Creare ed eliminare la sessione all'esterno del metodo `MessageListener` .

Consegna sincrona dei messaggi

I messaggi vengono consegnati in modo sincrono a un'applicazione se l'applicazione utilizza i metodi di ricezione degli oggetti MessageConsumer .

Utilizzando i metodi di ricezione, un'applicazione può attendere un periodo di tempo specificato per un messaggio oppure può attendere indefinitamente. In alternativa, se un'applicazione non desidera attendere un messaggio, può utilizzare il metodo Ricevi senza attesa.

Concetti correlati

Sessioni transazionali
Le applicazioni XMS possono eseguire transazioni locali. Una *transazione locale* è una transazione che implica modifiche solo alle risorse del gestore code o del service integration bus a cui è connessa l'applicazione.

Riconoscimento messaggio
Ogni sessione non sottoposta a transazione dispone di una modalità di riconoscimento che determina il modo in cui vengono riconosciuti i messaggi ricevuti dall'applicazione. Sono disponibili tre modalità di riconoscimento e la scelta della modalità di riconoscimento influisce sulla progettazione dell'applicazione.

Destinazioni

Un'applicazione di XMS utilizza un oggetto Destinazione per specificare la destinazione dei messaggi inviati e l'origine dei messaggi ricevuti.

Un'applicazione XMS può creare un oggetto Destinazione al runtime o ottenere una destinazione predefinita dal repository di oggetti gestiti.

Come con un ConnectionFactory, il modo più flessibile per un'applicazione XMS per specificare una destinazione è definirlo come un oggetto gestito. Utilizzando questo approccio, le applicazioni scritte in linguaggio C, C + + e .NET e Java, possono condividere le definizioni della destinazione. Le proprietà degli oggetti di destinazione gestiti possono essere modificate senza modificare alcun codice.

Per le applicazioni .NET , creare una destinazione utilizzando il metodo CreateTopic o CreateQueue . Questi due metodi sono disponibili in entrambi gli oggetti ISession e XMSFactoryFactory nell'API .NET . Per ulteriori informazioni, consultare “Destinazioni in .NET” a pagina 654 e ../refdev/sapidest.dita#sapidest.

Identificativi risorsa uniforme argomento

L'URI (uniform resource identifier) dell'argomento specifica il nome dell'argomento; può anche specificare una o più proprietà per esso.

L'URI per un argomento inizia con l'argomento di sequenza: //, seguito dal nome dell'argomento e (facoltativo) un elenco di coppie nome - valore che impostano le restanti proprietà dell'argomento. Il nome di un argomento non può essere vuoto.

Di seguito è riportato un esempio in un frammenti di codice .NET :

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

Per ulteriori informazioni sulle proprietà di un argomento, incluso il nome e i valori validi che è possibile utilizzare in un URI, consultare Proprietà della destinazione.

Quando si specifica un URI di argomento da utilizzare in una sottoscrizione, è possibile utilizzare i caratteri jolly. La sintassi per questi caratteri jolly dipende dal tipo di connessione e dalla versione broker; è disponibile la seguente opzione:

- SIB (Service Integration Bus) WebSphere Application Server

SIB (Service Integration Bus) WebSphere Application Server

Il SIB (service integration bus) WebSphere Application Server utilizza i seguenti caratteri jolly:

* per far corrispondere qualsiasi carattere ad un livello nella gerarchia
// per mettere in corrispondenza 0 o più livelli
//. per far corrispondere 0 o più livelli (alla fine di un'espressione Argomento)

Tabella 82 a pagina 641 fornisce alcuni esempi su come utilizzare questo schema di caratteri jolly.

Tabella 82. URI di esempio che utilizzano lo schema jolly per il SIB (Service Integration Bus) WebSphere Application Server

URI (Uniform Resource Identifier)	Corrisponde a	Esempi
"topic://Sport/ * ball/Results"	Tutti gli argomenti con un unico nome di livello gerarchico che termina in "palla" tra Sport e Risultati	"topic://Sport/Football/Results" e "topic://Sport/Netball/Results"
"argomento://Sport//Risultati"	Tutti gli argomenti che iniziano con "Sport/" e finiscono in "/Risultati"	"topic://Sport/Football/Results" e "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	Tutti gli argomenti che iniziano con "Sport/Football/"	"topic://Sport/Football/Results" e "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/ * ball//Results//."	Argomenti	"topic://Sport/Football/Results" e "topic://Sport/Netball/National/Div3/Results/2002/November"

Concetti correlati

Identificativi risorsa uniforme coda
L'URI di una coda specifica il nome della coda; può anche specificare una o più proprietà della coda.

Destinazioni temporanee
Le applicazioni XMS possono creare e utilizzare destinazioni temporanee.

Identificativi risorsa uniforme coda

L'URI di una coda specifica il nome della coda; può anche specificare una o più proprietà della coda.

L'URI per una coda inizia con la sequenza `queue://`, seguita dal nome della coda; potrebbe includere anche un elenco di coppie nome - valore che impostano le rimanenti proprietà della coda.

Per le code IBM MQ (ma non per le code del provider di messaggistica predefinito WebSphere Application Server), il gestore code su cui si trova la coda può essere specificato prima della coda, con una / che separa il nome del gestore code dal nome della coda.

Se viene specificato un gestore code, questo deve essere quello a cui XMS è direttamente connesso per la connessione che utilizza questa coda oppure deve essere accessibile da questa coda. I gestori code

remoti sono supportati solo per richiamare i messaggi dalle code e non per inserire i messaggi nelle code. Per i dettagli completi, fare riferimento alla documentazione del gestore code IBM MQ .

Se non viene specificato alcun gestore code, il separatore / extra è facoltativo e la sua presenza o assenza non fa alcuna differenza nella definizione della coda.

Le definizioni di coda riportate di seguito sono tutte equivalenti per una coda IBM MQ denominata QB su un gestore code denominato QM_A, a cui XMS è direttamente connesso:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

Concetti correlati

Identificativi risorsa uniforme argomento
L'URI (uniform resource identifier) dell'argomento specifica il nome dell'argomento; può anche specificare una o più proprietà per esso.

Destinazioni temporanee
Le applicazioni XMS possono creare e utilizzare destinazioni temporanee.

Destinazioni temporanee

Le applicazioni XMS possono creare e utilizzare destinazioni temporanee.

Un'applicazione generalmente utilizza una destinazione temporanea per ricevere risposte ai messaggi di richiesta. Per specificare la destinazione in cui deve essere inviata una risposta a un messaggio di richiesta, un'applicazione richiama il metodo Set JMSReplyTo dell'oggetto Message che rappresenta il messaggio di richiesta. La destinazione specificata nella chiamata può essere una destinazione temporanea.

Anche se una sessione viene utilizzata per creare una destinazione temporanea, l'ambito di una destinazione temporanea è in realtà la connessione utilizzata per creare la sessione. Qualsiasi sessione della connessione può creare produttori di messaggi e consumer di messaggi per la destinazione temporanea. La destinazione temporanea rimane fino a quando non viene esplicitamente eliminata o fino al termine della connessione, a seconda di quale delle due si verifica per prima.

Quando un'applicazione crea una coda temporanea, viene creata una coda nel server di messaggistica a cui è connessa l'applicazione. Se l'applicazione è connessa ad un gestore code, viene creata una coda dinamica dalla coda modello il cui nome è specificato dalla proprietà XMSC_WMQ_TEMPORARY_MODEL e il prefisso utilizzato per formare il nome della coda dinamica è specificato dalla proprietà XMSC_WMQ_TEMP_Q_PREFIX . Se l'applicazione è connessa ad un SIB (Service Integration Bus), viene creata una coda temporanea nel bus e il prefisso utilizzato per formare il nome della coda temporanea viene specificato dalla proprietà XMSC_WPM_TEMP_Q_PREFIX .

Quando un'applicazione connessa a un service integration bus crea un argomento temporaneo, il prefisso utilizzato per formare il nome dell'argomento temporaneo viene specificato dalla proprietà XMSC_WPM_TEMP_TOPIC_PREFIX .

Concetti correlati

Identificativi risorsa uniforme argomento
L'URI (uniform resource identifier) dell'argomento specifica il nome dell'argomento; può anche specificare una o più proprietà per esso.

Identificativi risorsa uniforme coda
L'URI di una coda specifica il nome della coda; può anche specificare una o più proprietà della coda.

Produttori di messaggi

In XMS, è possibile creare un produttore di messaggi con una destinazione valida o senza alcuna destinazione associata. Quando si crea un produttore di messaggi con una destinazione null, è necessario specificare una destinazione valida quando si invia un messaggio.

Produttori di messaggi con destinazione associata

In questo scenario, il produttore del messaggio viene creato utilizzando una destinazione valida. Durante l'operazione di invio, non è necessario specificare la destinazione.

Produttori di messaggi senza destinazione associata

In XMS .NET, è possibile creare un produttore di messaggi con destinazione null.

Per creare un produttore di messaggi senza destinazione associata quando si utilizza l'API .NET , è necessario passare NULL come parametro nel metodo `CreateProducer()` dell'oggetto ISession (ad esempio, `session.CreateProducer(null)`). Tuttavia, è necessario specificare una destinazione valida quando viene inviato il messaggio.

Utenti del messaggio

I consumer di messaggi possono essere classificati come sottoscrittori durevoli e non durevoli e consumer di messaggi sincroni e asincroni.

Sottoscrittori permanenti

Un sottoscrittore durevole è un utente di messaggi che riceve tutti i messaggi pubblicati su un argomento, inclusi i messaggi pubblicati mentre il sottoscrittore non è attivo.

Le informazioni contenute in questo argomento sono rilevanti solo se un'applicazione si connette a un gestore code IBM MQ o a un SIB (Service Integration Bus) WebSphere Application Server . Le informazioni non sono rilevanti per una connessione in tempo reale a un broker.

Per creare un sottoscrittore durevole per un argomento, un'applicazione richiama il metodo Crea sottoscrittore durevole di un oggetto Session, specificando come parametri un nome che identifica la sottoscrizione durevole e un oggetto Destinazione che rappresenta l'argomento. L'applicazione può creare un sottoscrittore durevole con o senza un selettore di messaggi e può specificare se il sottoscrittore durevole deve ricevere i messaggi pubblicati dalla propria connessione.

La sessione utilizzata per creare un sottoscrittore durevole deve avere un identificativo client associato. L'identificativo client è lo stesso associato alla connessione utilizzata per creare la sessione; è specificato come descritto in “ConnectionFactories e oggetti Connection” a pagina 635.

Il nome che identifica la sottoscrizione durevole deve essere univoco all'interno dell'identificativo client e quindi l'identificativo client fa parte dell'identificativo univoco completo della sottoscrizione durevole. Il server di messaggistica conserva un record della sottoscrizione durevole e garantisce che tutti i messaggi pubblicati sull'argomento vengano conservati fino a quando non vengono riconosciuti dal sottoscrittore durevole o fino a quando non scadono.

Il server di messaggistica continua a conservare il record della sottoscrizione durevole anche dopo la chiusura del sottoscrittore durevole. Per riutilizzare una sottoscrizione durevole creata in precedenza, un'applicazione deve creare un sottoscrittore durevole specificando lo stesso nome di sottoscrizione e utilizzando una sessione con lo stesso identificativo client, come quelli associati alla sottoscrizione durevole. Solo una sessione alla volta può avere un sottoscrittore durevole per una particolare sottoscrizione durevole.

L'ambito di una sottoscrizione durevole è il server di messaggistica che conserva un record della sottoscrizione. Se due applicazioni connesse a server di messaggistica differenti creano un sottoscrittore durevole utilizzando lo stesso nome sottoscrizione e lo stesso identificativo client, vengono create due sottoscrizioni durevoli completamente indipendenti.

Per eliminare una sottoscrizione durevole, un'applicazione richiama il metodo Unsubscribe di un oggetto Session, specificando come parametro il nome che identifica la sottoscrizione durevole. L'identificativo client associato alla sessione deve essere uguale a quello associato alla sottoscrizione durevole. Il server di messaggistica elimina il record della sottoscrizione durevole che sta gestendo e non invia ulteriori messaggi al sottoscrittore durevole.

Per modificare una sottoscrizione esistente, un'applicazione può creare un sottoscrittore durevole utilizzando lo stesso nome sottoscrizione e lo stesso identificativo client, ma specificando un argomento

o un selettore di messaggi differenti (o entrambi). La modifica di una sottoscrizione durevole equivale ad eliminare la sottoscrizione e a crearne una nuova.

Per un'applicazione che si connette a un gestore code IBM MQ , XMS gestisce le code del sottoscrittore. Pertanto, l'applicazione non è richiesta per specificare una coda del sottoscrittore. XMS ignorerà la coda del sottoscrittore, se specificata.

Non è possibile modificare la coda del sottoscrittore per una sottoscrizione durevole. L'unico modo per modificare la coda del sottoscrittore è eliminare la sottoscrizione e crearne una nuova.

Per un'applicazione che si connette a un SIB (Service Integration Bus), ogni sottoscrittore durevole deve avere una home di sottoscrizione durevole designata. Per specificare la home della sottoscrizione durevole per tutti i sottoscrittori durevoli che utilizzano la stessa connessione, impostare la proprietà XMSC_WPM_DUR_SUB_HOME dell'oggetto ConnectionFactory utilizzato per creare la connessione. Per specificare la home di sottoscrizione durevole per un singolo argomento, impostare la proprietà XMSC_WPM_DUR_SUB_HOME dell'oggetto Destinazione che rappresenta l'argomento. È necessario specificare una home di sottoscrizione durevole per una connessione prima che un'applicazione possa creare un sottoscrittore durevole che utilizza la connessione. Qualsiasi valore specificato per una destinazione sovrascrive il valore specificato per la connessione.

Consumer di messaggi sincroni e asincroni

Il destinatario del messaggio sincrono riceve i messaggi da una coda in modo sincrono e il destinatario del messaggio asincrono riceve il messaggio da una coda in modo asincrono.

Utenti di messaggi sincroni

Un utente di messaggi sincrono riceve un messaggio alla volta. Quando viene utilizzato il metodo `Receive(wait interval)` ; la chiamata attende solo un periodo di tempo specificato, in millisecondi, per un messaggio o fino a quando il destinatario del messaggio non viene chiuso.

Se viene utilizzato il metodo ReceiveNoWait (), il destinatario del messaggio sincrono riceve i messaggi senza alcun ritardo; se il messaggio successivo è disponibile, viene ricevuto immediatamente, altrimenti viene restituito un puntatore a un oggetto Messaggio null.

Consumer messaggi asincroni

Il listener dei messaggi registrato dall'applicazione viene richiamato ogni volta che un nuovo messaggio è disponibile sulla coda.

Messaggi non elaborabili in XMS

Un messaggio non elaborabile è un messaggio che non può essere elaborato da un'applicazione MDB ricevente. Se viene rilevato un messaggio non elaborabile, l'oggetto MessageConsumer XMS può riaccodare il messaggio in base a due proprietà della coda, BOQUEUE e BOTHRESH.

In alcune circostanze, un messaggio consegnato a un MDB potrebbe essere sottoposto a rollback su una coda IBM MQ . Ciò può verificarsi, ad esempio, se un messaggio viene consegnato all'interno di un'unità di lavoro di cui viene successivamente eseguito il rollback. Un messaggio di cui viene eseguito il rollback viene generalmente consegnato di nuovo, ma un messaggio formattato in modo errato potrebbe causare ripetutamente un errore MDB e quindi non può essere consegnato. Tale messaggio è chiamato un messaggio di veleno. È possibile configurare IBM MQ in modo che il messaggio non elaborabile venga automaticamente trasferito a un'altra coda per ulteriori indagini o venga eliminato. Per informazioni su come configurare IBM MQ in questo modo, consultare Gestione dei messaggi non elaborabili in ASF.

A volte, un messaggio formattato in modo errato arriva su una coda. In questo contesto, un formato non corretto indica che l'applicazione ricevente non può elaborare correttamente il messaggio. Un messaggio di questo tipo può causare il malfunzionamento dell'applicazione ricevente e il backout di questo messaggio formattato in modo errato. Il messaggio può essere inviato ripetutamente alla coda di input e ripetutamente ripristinato dall'applicazione. Questi messaggi sono noti come messaggi dannosi. L'oggetto XMS MessageConsumer rileva i messaggi non elaborabili e li reinvia a una destinazione alternativa.

Il gestore code IBM MQ conserva un record del numero di volte in cui è stato eseguito il backout di ciascun messaggio. Quando questo valore raggiunge un valore di soglia configurabile, l'utente del messaggio riaccoda il messaggio ad una coda di backout denominata. Se questa riaccodamento non riesce per qualsiasi motivo, il messaggio viene rimosso dalla coda di input e riaccodato alla coda di messaggi non recapitabili o eliminato.

Gli oggetti XMS ConnectionConsumer gestiscono i messaggi non elaborabili nello stesso modo e utilizzando le stesse proprietà della coda. Se più utenti della connessione stanno monitorando la stessa coda, è possibile che il messaggio non elaborabile possa essere consegnato a un'applicazione più volte del valore di soglia prima che si verifichi la riaccodamento. Questo comportamento è dovuto al modo in cui i singoli utenti della connessione monitorano le code e riaccodano i messaggi dannosi.

Il valore di soglia ed il nome della coda di backout sono attributi di una coda IBM MQ . I nomi degli attributi sono BackoutThreshold e BackoutRequeueQName. La coda a cui si applicano è la seguente:

- Per la messaggistica point - to - point, questa è la coda locale sottostante. Ciò è importante quando gli utenti dei messaggi e dei collegamenti utilizzano gli alias della coda.
- Per la messaggistica di pubblicazione / sottoscrizione in modalità normale del fornitore di messaggistica di IBM MQ , è la coda modello da cui viene creata la coda gestita dell'argomento.
- Per la messaggistica di pubblicazione / sottoscrizione in modalità di migrazione del provider di messaggistica IBM MQ , è la coda CCSUB definita nell'oggetto factory TopicConnectiono la coda CCDSUB definita nell'oggetto argomento.

Per impostare gli attributi QName BackoutThreshold e BackoutRequeue, immettere il seguente comando MQSC:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Per la messaggistica di pubblicazione / sottoscrizione, se il sistema crea una coda dinamica per ciascuna sottoscrizione, questi valori di attributo vengono ottenuti dalla coda modello IBM MQ classes for JMS , SYSTEM.JMS.MODEL.QUEUE. Per modificare queste impostazioni, utilizzare:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Se il valore della soglia di backout è zero, la gestione dei messaggi non elaborabili è disabilitata e i messaggi non elaborabili rimangono sulla coda di input. Altrimenti, quando il conteggio di backout raggiunge il valore di soglia, il messaggio viene inviato alla coda di backout indicata.

Se il conteggio di backout raggiunge il valore di soglia, ma il messaggio non può andare alla coda di backout, il messaggio viene inviato alla coda di messaggi non recapitabili o, se il messaggio è non persistente, viene eliminato.

Questa situazione si verifica se la coda di backout non è definita o se l'oggetto MessageConsumer non può inviare il messaggio alla coda di backout.

Configurazione del sistema per eseguire la gestione dei messaggi non elaborabili

La coda utilizzata da XMS .NET durante l'interrogazione degli attributi **BOTHRESH** e **BOQNAME** dipende dallo stile di messaggistica eseguito:

- Per la messaggistica point - to - point, questa è la coda locale sottostante. Ciò è importante quando un'applicazione XMS .NET utilizza i messaggi dalle code alias o dalle code cluster.
- Per la messaggistica di pubblicazione / sottoscrizione, viene creata una coda gestita per contenere i messaggi per una applicazione. XMS .NET interroga la coda gestita per determinare i valori per gli attributi **BOTHRESH** e **BOQNAME** .

  La coda gestita viene creata da una coda modello associata all'oggetto Argomento a cui l'applicazione ha eseguito la sottoscrizione ed eredita i valori degli attributi **BOTHRESH** e **BOQNAME** dalla coda

modello. La coda modello utilizzata dipende dal fatto che l'applicazione ricevente abbia eliminato una sottoscrizione durevole o non durevole:

- La coda modello utilizzata per le sottoscrizioni durevoli è specificata dall'attributo **MDURMDL** dell'argomento. Il valore predefinito di questo attributo è SYSTEM.DURABLE.MODEL.QUEUE.
- Per sottoscrizioni non durevoli, la coda modello utilizzata è specificata dall'attributo **MNDURMDL** . Il valore predefinito dell'attributo **MNDURMDL** è SYSTEM.NDURABLE.MODEL.QUEUE.

Quando si interrogano gli attributi **BOTHRESH** e **BOQNAME** , XMS .NET:

- Apre la coda locale o la coda di destinazione per una coda alias.
- Interroga gli attributi **BOTHRESH** e **BOQNAME** .
- Chiude la coda locale o la coda di destinazione per una coda alias.

Le opzioni di apertura utilizzate quando si apre una coda locale o la coda di destinazione per una coda alias, dipendono dalla versione di IBM MQ utilizzata:

- Per IBM MQ 9.1.0 Fix Pack 4 Long Term Support e versioni precedenti, e IBM MQ 9.1.4 Continuous Delivery e versioni precedenti: se la coda locale o la coda di destinazione per una coda alias è una coda cluster, XMS .NET apre la coda con le opzioni MQOO_INPUT_AS_Q_DEF, MQOO_INQUIRE e MQOO_FAIL_IF_QUIESCING . Ciò significa che l'utente che esegue l'applicazione ricevente deve disporre dell'interrogazione e dell'accesso all'istanza locale della coda cluster.

  XMS .NET apre tutti gli altri tipi di coda locale con le opzioni di apertura MQOO_INQUIRE e MQOO_FAIL_IF_QUIESCING. Affinché XMS .NET esegua una query dei valori degli attributi, l'utente che esegue l'applicazione ricevente deve avere accesso alla coda locale.
- Quando si utilizza XMS .NET da IBM MQ 9.1.5 e IBM MQ 9.1.0 Fix Pack 5, l'utente che esegue l'applicazione ricevente deve avere accesso alla coda locale, indipendentemente dal tipo di coda.

Per spostare i messaggi non elaborabili in una coda di riaccodamento di backout o in una coda di messaggi non recapitabili del gestore code, è necessario concedere all'utente che esegue l'applicazione le autorizzazioni put e passall .

Gestione dei messaggi non elaborabili in ASF
Quando si utilizzano le ASF (Application Server Facilities), ConnectionConsumer, invece di MessageConsumer, elabora messaggi non elaborabili. Il ConnectionConsumer riaccoda i messaggi in base alle proprietà QName BackoutThreshold e BackoutRequeuedella coda.

Quando un'applicazione utilizza ConnectionConsumers, le circostanze in cui viene eseguito il backout di un messaggio dipendono dalla sessione fornita dal server delle applicazioni:

- Quando la sessione non è sottoposta a transazione, con AUTO_RICONOSCIUTO_AUTOMATICO o DUPS_OK_RICONOSCIMENTO, viene eseguito il backout di un messaggio solo dopo un errore di sistema o se l'applicazione termina in modo imprevisto.
- Quando la sessione non è sottoposta a transazioni con CLIENT_ACKNOWLEDGED, i messaggi non riconosciuti possono essere ripristinati dal server delle applicazioni che richiama `Session.recover()`.

  Di solito, l'implementazione client di MessageListener o del server delle applicazioni richiama `Message.acknowledge()`. `Message.acknowledge()` riconosce tutti i messaggi consegnati nella sessione fino ad ora.
- Quando la sessione viene eseguita, i messaggi non riconosciuti possono essere ripristinati dal server delle applicazioni che richiama `Session.rollback()`.

Browser coda

Un'applicazione utilizza un browser della coda per esaminare i messaggi su una coda senza rimuoverli.

Per creare un browser delle code, un'applicazione richiama il metodo Crea browser delle code di un oggetto ISession, specificando come parametro un oggetto Destinazione che identifica la coda da esaminare. L'applicazione può creare un browser della coda con o senza un selettore di messaggi.

Dopo aver creato un browser di code, l'applicazione può chiamare il metodo GetEnumerator dell'oggetto IQueueBrowser per ottenere un elenco dei messaggi sulla coda. Il metodo restituisce un enumeratore che incapsula un elenco di oggetti Messaggio. L'ordine degli oggetti Messaggio nell'elenco è lo stesso dell'ordine in cui i messaggi vengono richiamati dalla coda. L'applicazione può quindi utilizzare l'enumeratore per esaminare ogni messaggio a turno.

L'enumeratore viene aggiornato dinamicamente quando i messaggi vengono inseriti nella coda e rimossi dalla coda. Ogni volta che l'applicazione richiama IEnumerator.MoveNext() per sfogliare il messaggio successivo nella coda, il messaggio riflette il contenuto corrente della coda.

Un'applicazione può richiamare il metodo GetEnumerator più di una volta per un determinato browser della coda. Ciascuna chiamata restituisce un nuovo enumeratore. L'applicazione può quindi utilizzare più di un enumeratore per esaminare i messaggi su una coda e mantenere più posizioni all'interno della coda.

Un'applicazione può utilizzare un browser di code per cercare un messaggio adatto da eliminare da una coda e quindi utilizzare un consumatore di messaggi con un selettore di messaggi per rimuovere il messaggio. Il selettore del messaggio può selezionare il messaggio in base al valore del campo di intestazione JMSMessageID . Per informazioni su questo e altri campi di intestazione del messaggio JMS, consultare “Campi di intestazione in un messaggio XMS” a pagina 669.

Richiedenti

Un'applicazione utilizza un richiedente per inviare un messaggio di richiesta e quindi attendere e ricevere la risposta.

Molte applicazioni di messaggistica si basano su algoritmi che inviano un messaggio di richiesta e quindi attendono una risposta. XMS fornisce una classe denominata Requestor per facilitare lo sviluppo di questo stile di applicazione.

Per creare un richiedente, un'applicazione richiama il costruttore Create Requestor della classe Requestor, specificando come parametri un oggetto Session e un oggetto Destination che identifica dove devono essere inviati i messaggi di richiesta. La sessione non deve essere sottoposta a transazione né deve avere una modalità di riconoscimento XMSC_CLIENT_ACKNOWLEDGEMENT. Il costruttore crea automaticamente una coda temporanea o un argomento in cui devono essere inviati i messaggi di risposta.

Una volta creato un richiedente, l'applicazione può richiamare il metodo di richiesta dell'oggetto Requestor per inviare un messaggio di richiesta e quindi attendere e ricevere una risposta dall'applicazione che riceve il messaggio di richiesta. La chiamata attende la ricezione della risposta o la fine della sessione, a seconda di quale si verifica per prima. Solo una risposta è richiesta dal richiedente per ogni messaggio di richiesta.

Quando l'applicazione chiude il richiedente, la coda temporanea o l'argomento vengono eliminati. La sessione associata, tuttavia, non viene chiusa.

Eliminazione oggetto

Quando un'applicazione elimina un oggetto XMS creato, XMS rilascia le risorse interne assegnate all'oggetto.

Quando un'applicazione crea un oggetto XMS , XMS assegna memoria e altre risorse interne all'oggetto. XMS conserva queste risorse interne finché l'applicazione non elimina esplicitamente l'oggetto richiamando il metodo close o delete dell'oggetto, a quel punto XMS rilascia le risorse interne. Se un'applicazione tenta di eliminare un oggetto già eliminato, la chiamata viene ignorata.

Quando un'applicazione elimina un oggetto Connection o Session, XMS elimina automaticamente alcuni oggetti associati e rilascia le relative risorse interne. Si tratta di oggetti creati dall'oggetto Connection o Session e che non hanno alcuna funzione indipendente dall'oggetto. Questi oggetti sono mostrati in Tabella 83 a pagina 648.

Nota: se un'applicazione chiude una connessione con sessioni dipendenti, vengono eliminati anche tutti gli oggetti dipendenti da tali sessioni. Solo un oggetto Connection o Session può avere oggetti dipendenti.

Tabella 83. Oggetti eliminati automaticamente		
Oggetto eliminato	**Metodo**	**Oggetti dipendenti che vengono eliminati automaticamente**
Connessione	Chiudi connessione	Oggetti ConnectionMetaData e Session
Sessione	Chiudi sessione	MessageConsumer, MessageProducer, QueueBrowsere oggetti Requestor

Transazioni IBM MQ XA gestite tramite XMS

Le transazioni IBM MQ XA gestite possono essere utilizzate tramite XMS.

Per utilizzare le transazioni XA tramite XMS, è necessario creare una sessione transattiva. Quando la transazione XA è in uso, il controllo della transazione avviene tramite le transazioni globali DTC (Distributed Transaction Coordinator) e non attraverso le sessioni XMS . Quando si utilizzano le transazioni XA, `Session.commit` o `Session.rollback` non possono essere emessi sulla sessione XMS . Utilizzare invece i metodi `Transscope.Commit` o `Transscope.Rollback` DTC per eseguire il commit o il rollback delle transazioni. Se una sessione viene utilizzata per la transazione XA, il producer o il consumer creato utilizzando la sessione deve essere una parte della transazione XA. Non possono essere utilizzati per operazioni al di fuori dell'ambito della transazione XA. Non possono essere utilizzati per operazioni come `Producer.send` o `Consumer.receive` all'esterno della transazione XA.

Un oggetto eccezione `IllegalStateException` viene generato se:

- La sessione di transazione XA viene utilizzata per `Session.commit` o `Session.rollback`.
- Gli oggetti producer o consumer una volta utilizzati nella sessione di transazione XA vengono utilizzati al di fuori dell'ambito della transazione XA.

Le transazioni XA non sono supportate nei consumer asincroni.

Nota:

1. Una chiusura potrebbe essere emessa sull'oggetto `Producer`, `Consumer`, `Sessiono Connection` prima del commit della transazione XA. In quali casi viene eseguito il rollback dei messaggi nella transazione. Allo stesso modo, se la connessione viene interrotta prima del commit della transazione XA, viene eseguito il rollback di tutti i messaggi nella transazione. Per un oggetto `Producer` , un rollback indica che i messaggi non vengono inseriti sulla coda. Per un oggetto `Consumer` , un rollback indica che i messaggi rimangono nella coda.
2. Se un oggetto `Producer` inserisce un messaggio con `TimeToLive` in `TransactionScope` e un `commit` viene emesso una volta trascorso il tempo, il messaggio può scadere prima dell'emissione di `commit` . In questo caso, il messaggio non viene reso disponibile agli oggetti `Consumer` .
3. Gli oggetti `Session` non sono supportati tra i thread. L'utilizzo di transazioni con oggetti `Session` condivisi tra thread non è supportato.

XMS Tipi primitivi

XMS fornisce equivalenti degli otto tipi primitivi Java (byte, short, int, long, float, double, char e boolean). Ciò consente lo scambio di messaggi tra XMS e JMS senza che i dati vadano persi o danneggiati.

Tabella 84 a pagina 648 elenca il tipo di dati Java equivalente, la dimensione e il valore minimo e massimo di ciascun tipo primitivo XMS .

Tabella 84. Tipi di dati XMS e relativi Java equivalenti				
Tipo di dati XMS	**Tipo di datiJava compatibile**	**Dimensione**	**Valore minimo**	**Valore massimo**
System.Boolean	booleano	32 bit	No	vero, true

Tabella 84. Tipi di dati XMS e relativi Java equivalenti (Continua)				
Tipo di dati XMS	**Tipo di datiJava compatibile**	**Dimensione**	**Valore minimo**	**Valore massimo**
System.SBYTE	byte	8 bit	-2^7 (-128)	2^7-1 (127)
System.BYTE	byte	8 bit	-2^7 (-128)	2^7-1 (127)
System.CHAR	byte	8 bit	-2^7 (-128)	2^7-1 (127)
System.Int16	breve	16 bit	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 bit	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	lungo	64 bit	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	mobile	32 bit	-3.402823E-38 (precisione a 7 cifre)	3.402823E+38 (con precisione a 7 cifre)
System.Double	doppio	64 bit	-1.79769313486231E-308 (precisione a 15 cifre)	1.79769313486231E+308 (precisione a 15 cifre)

Conversione implicita di un valore di proprietà da un tipo di dati a un altro

Quando un'applicazione acquisisce il valore di una proprietà, il valore può essere convertito da XMS in un altro tipo di dati. Molte regole regolano quali conversioni sono supportate e come XMS esegue le conversioni.

Una proprietà di un oggetto ha un nome e un valore; il valore ha un tipo di dati associato, dove il valore di una proprietà è indicato anche come *tipo di proprietà*.

Un'applicazione utilizza i metodi della classe PropertyContext per richiamare e impostare le proprietà degli oggetti. Per ottenere il valore di una proprietà, un'applicazione richiama il metodo appropriato per il tipo di proprietà. Ad esempio, per ottenere il valore di una proprietà integer, un'applicazione in genere richiama il metodo di proprietà GetInt.

Tuttavia, quando un'applicazione ottiene il valore di una proprietà, il valore può essere convertito da XMS in un altro tipo di dati. Ad esempio, per ottenere il valore di una proprietà integer, un'applicazione può chiamare il metodo di proprietà GetString, che restituisce il valore della proprietà come stringa. Le conversioni supportate da XMS sono mostrate in Tabella 85 a pagina 649.

Tabella 85. Conversioni supportate da un tipo di proprietà ad altri tipi di dati	
Tipo di proprietà	**Tipi di dati di destinazione supportati**
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
array System.SByte	System.String

Tabella 85. Conversioni supportate da un tipo di proprietà ad altri tipi di dati (Continua)	
Tipo di proprietà	**Tipi di dati di destinazione supportati**
System.Int16	System.String, System.Int32, System.Int64

Le seguenti regole generali regolano le conversioni supportate:

- I valori delle proprietà numeriche possono essere convertiti da un tipo di dati a un altro, purché non si perda alcun dato durante la conversione. Ad esempio, il valore di una proprietà con tipo dati System.Int32 può essere convertito in un valore con tipo dati System.Int64, ma non può essere convertito in un valore con tipo dati System.Int16.
- Un valore di proprietà di qualsiasi tipo di dati può essere convertito in una stringa.
- Un valore della proprietà stringa può essere convertito in qualsiasi altro tipo di dati, a condizione che la stringa sia formattata correttamente per la conversione. Se un'applicazione tenta di convertire un valore della proprietà stringa che non è formattato correttamente, XMS potrebbe restituire errori.
- Se un'applicazione tenta una conversione non supportata, XMS potrebbe restituire un errore.

Le seguenti regole si applicano quando un valore di proprietà viene convertito da un tipo di dati a un altro:

- Quando si converte un valore della proprietà booleana in una stringa, il valore true viene convertito nella stringa "true"e il valore false viene convertito nella stringa "false".
- Quando si converte un valore della proprietà booleana in un tipo di dati numerico, incluso System.SByte, il valore true viene convertito in 1 e il valore false viene convertito in 0.
- Quando si converte un valore di proprietà stringa in un valore booleano, la stringa "true" (non sensibile al maiuscolo / minuscolo) o "1" viene convertita in true e la stringa "false" (non sensibile al maiuscolo / minuscolo) o "0" viene convertita in false. Tutte le altre stringhe non possono essere convertite.
- Quando si converte un valore di proprietà stringa in un valore con tipo di dati System.Int32, System.Int64, System.SByteo System.Int16, la stringa deve avere il seguente formato:

  [*spazi*] [*segno*]*cifre*

  I componenti stringa sono definiti come segue:

  spazi vuoti
  Caratteri vuoti iniziali facoltativi.

  firmare
  Un segno più (+) o segno meno (-) facoltativo.

  cifre
  Una sequenza contigua di caratteri cifra (0-9). Deve essere presente almeno un carattere cifra.

  Dopo la sequenza di caratteri cifra, la stringa può contenere altri caratteri che non sono caratteri cifra, ma la conversione si interrompe non appena viene raggiunto il primo di questi caratteri. Si presuppone che la stringa rappresenti un numero intero decimale.

  XMS potrebbe restituire un errore se la stringa non è formattata correttamente.
- Quando si converte un valore di proprietà della stringa in un valore con tipo di dati System.Double o System.Float, la stringa deve avere il seguente formato:

  [*blank*] [*segno*] [*cifre*] [*punto*[*d_cifre*]] [*e_char*[*e_segno*]*e_cifre*]

  I componenti stringa sono definiti come segue:

  spazi vuoti
  (Facoltativo) Caratteri vuoti iniziali.

  firmare
  (Facoltativo) Il segno più (+) o il segno meno (-).

  cifre
  Una sequenza contigua di caratteri cifra (0-9). Almeno un carattere cifra deve essere presente in *cifre* o *d_digits*.

punto
(Facoltativo) Punto decimale (.).

d_cifre
Una sequenza contigua di caratteri cifra (0-9). Almeno un carattere cifra deve essere presente in *cifre* o *d_digits*.

e_car
Un carattere esponente, che è *E* o *e*.

e_firma
(Facoltativo) Segno più (+) o segno meno (-) per l'esponente.

e_cifre
Una sequenza contigua di caratteri cifra (0-9) per l'esponente. Deve essere presente almeno un carattere cifra se la stringa contiene un carattere esponente.

Dopo la sequenza di caratteri cifra, o i caratteri facoltativi che rappresentano un esponente, la stringa può contenere altri caratteri che non sono caratteri cifra, ma la conversione si interrompe non appena viene raggiunto il primo di questi caratteri. Si presuppone che la stringa rappresenti un numero a virgola mobile decimale con un esponente che sia una potenza di 10.

XMS potrebbe restituire un errore se la stringa non è formattata correttamente.

- Quando si converte un valore di proprietà numerico in una stringa, incluso un valore di proprietà con tipo di dati System.SByte, il valore viene convertito nella rappresentazione di stringa del valore come un numero decimale, non la stringa contenente il carattere ASCII per tale valore. Ad esempio, il numero intero 65 viene convertito nella stringa "65", non nella stringa "A".
- Quando si converte un valore della proprietà della schiera di byte in una stringa, ogni byte viene convertito in 2 caratteri esadecimali che rappresentano il byte. Ad esempio, l'array di byte {0xF1, 0x12, 0x00, 0xFF} viene convertito nella stringa "F11200FF".

Le conversioni da un tipo di proprietà ad altri tipi di dati sono supportate dai metodi delle classi Proprietà e PropertyContext .

Iteratori

Un iteratore incapsula un elenco di oggetti e un cursore che mantiene la posizione corrente nell'elenco. Il concetto di un iteratore, come disponibile in IBM MQ Message Service Client (XMS) for C/C++, viene implementato utilizzando l'interfaccia IEnumerator in IBM MQ Message Service Client (XMS) for .NET.

Quando viene creato un iteratore, la posizione del cursore si trova prima del primo oggetto. Un'applicazione utilizza un iteratore per richiamare ciascun oggetto.

La classe Iteratore di IBM MQ Message Service Client (XMS) for C/C++ è equivalente alla classe Enumeratore in Java. IBM MQ Message Service Client (XMS) for .NET è simile a Java e utilizza un'interfaccia IEnumerator.

Un'applicazione può utilizzare un IEnumerator per eseguire le seguenti attività:

- Per ottenere le proprietà di un messaggio
- Per ottenere le coppie nome - valore nel corpo di un messaggio di associazione
- Per sfogliare i messaggi su una coda
- Per ottenere i nomi delle proprietà del messaggio definite da JMS supportate da un collegamento

CCSID (Coded character set identifier)

In XMS .NET, tutte le stringhe vengono trasmesse utilizzando la stringa .NET nativa. Poiché questa ha una codifica fissa, non sono richieste ulteriori informazioni per interpretarla. Quindi, la proprietà XMSC_CLIENT_CCSID non è richiesta per le applicazioni XMS .NET .

Codici di errore e di eccezione XMS

XMS utilizza un intervallo di codici di errore per indicare errori. Questi codici di errore non sono elencati esplicitamente in questa documentazione perché possono variare da release a release. Solo i codici di eccezione XMS (nel formato XMS_X_ ...) vengono documentati perché rimangono gli stessi tra le release di XMS.

Riconnessione automatica del client IBM MQ tramite XMS

Configurare il proprio client XMS per riconnettersi automaticamente in seguito a un errore di rete, gestore code o server durante l'utilizzo del client IBM WebSphere MQ 7.1 e versioni successive, come provider di messaggi.

Utilizzare le proprietà `WMQ_CONNECTION_NAME_LIST` e `WMQ_CLIENT_RECONNECT_OPTIONS` della classe `MQConnectionFactory` per configurare una connessione client per la riconnessione automatica. La riconnessione client automatica riconnette un client dopo un errore di connessione o come opzione dopo l'arresto del gestore code. La progettazione di alcune applicazioni client le rende inadatte per la riconnessione automatica.

Le connessioni client ricollegabili automaticamente diventano ricollegabili una volta stabilita la connessione.

Nota: Le proprietà `Opzioni di riconnessione client`, `Timeout riconnessione cliente` `Elenco nomi connessione` possono essere impostate anche mediante CCDT (Client Channel Definitions Table) oppure abilitando la riconnessione client tramite il file `mqclient.ini` .

Nota: Se le proprietà di riconnessione sono impostate sull'oggetto `ConnectionFactory` e anche in CCDT, la regola di precedenza è la seguente. Se il valore predefinito della proprietà dell'elenco dei nomi di connessione è impostato nell'oggetto `ConnectionFactory` , la CCDT ha la precedenza. Se l'elenco dei nomi di connessione non è impostato sul valore predefinito, i valori delle proprietà impostati nell'oggetto `ConnectionFactory` hanno la precedenza. Il valore predefinito dell'elenco nomi di connessione è `localhost(1414)`.

Scrittura di applicazioni XMS .NET

Gli argomenti contenuti in questa sezione forniscono informazioni utili per la scrittura di applicazioni XMS .NET .

Informazioni su questa attività

Questa sezione fornisce informazioni specifiche per la scrittura di applicazioni XMS .NET . Per informazioni generali sulla scrittura di applicazioni XMS , consultare “Scrittura di applicazioni XMS” a pagina 633.

Questa sezione contiene i seguenti argomenti:

- “Tipi di dati per .NET” a pagina 652
- “Operazioni gestite e non gestite in .NET” a pagina 653
- “Destinazioni in .NET” a pagina 654
- “Proprietà in .NET” a pagina 655
- “Gestione di proprietà non esistenti in .NET” a pagina 655
- “Gestione degli errori in .NET” a pagina 656
- “Utilizzo dei listener di messaggi ed eccezioni in .NET” a pagina 656

Tipi di dati per .NET

XMS .NET supporta System.Boolean, System.Byte, System.SByte, System.Char, System.String, System.Single, System.Double, System.Decimal, System.Int16, System.Int32, System.Int64, System.UInt16, System.UInt32, System.UInt64, e System.Object. I tipi di dati per il XMS .NET sono diversi dai tipi di dati per XMS C/C + +. È possibile utilizzare questo argomento per identificare i corrispondenti tipi di dati.

La seguente tabella mostra i tipi di dati XMS .NET e XMS C/C++ corrispondenti e li descrive brevemente.

Tabella 86. Tipi di dati per XMS .NET e XMS C/C++

Tipo XMS .NET	**tipo C/C++ XMS**	**Descrizione**
System.SByte	xmsSBYTE xmsINT8	Valore a 8 bit con segno
System.Byte	xmsBYTE xmsUINT8	Valore a 8 bit senza segno
System.Int16	xmsINT16 xmsSHORT	Valore a 16 bit con segno
System.UInt16	xmsUINT16 xmsUSHORT	Valore a 16 bit senza segno
System.Int32	xmsINT32 xmsINT	Valore a 32 bit con segno
System.UInt32	xmsUINT32 xmsUINT	Valore a 32 bit senza segno
System.Int64	xmsLONG xmsINT64	Valore a 64 bit con segno
System.UInt64	xmsULONG xmsUINT64	Valore a 64 bit senza segno
System.Char	xmsCHAR16	Carattere a 16 bit senza segno (Unicode per .NET)
System.Single	xmsFLOAT	Float a 32 bit IEEE
System.Double	xmsDOUBLE	Mobile IEEE a 64 bit
System.Boolean	xmsBOOL	Un valore True / False
Non applicabile	xmsCHAR	Valore a 8 bit con o senza segno (il valore con segno o senza segno dipende dalla piattaforma)
System.Decimal	Non applicabile	Numero intero con segno a 96 bit da 10^0 a 10^{28}
System.Object	Non applicabile	Base di tutti i tipi
System.String	Non applicabile	Tipo di stringa

Operazioni gestite e non gestite in .NET

Il codice gestito viene eseguito esclusivamente nell'ambiente di runtime del linguaggio comune .NET ed è interamente dipendente dai servizi forniti da tale runtime. Un'applicazione viene classificata come non gestita se una parte dell'applicazione viene eseguita o richiama i servizi all'esterno dell'ambiente di runtime del linguaggio comune .NET .

Alcune funzionalità avanzate non possono essere attualmente supportate nell'ambiente .NET gestito.

Se l'applicazione richiede alcune funzionalità che non sono attualmente supportate nell'ambiente completamente gestito, è possibile modificare l'applicazione per utilizzare l'ambiente non gestito senza richiedere modifiche sostanziali all'applicazione. Tuttavia, si noti che lo stack XMS utilizza il codice non gestito quando viene eseguita questa selezione.

Connessioni a un gestore code IBM MQ

Le connessioni gestite a WMQ_CM_CLIENT non supporteranno le comunicazioni non TCP e la compressione del canale. Tuttavia, queste connessioni potrebbero essere supportate utilizzando una connessione non gestita (WMQ_CM_CLIENT_UNMANAGED). Per ulteriori informazioni, consultare “Sviluppo di applicazioni .NET” a pagina 556.

Se si crea una factory di connessione da un oggetto gestito in un ambiente non gestito, è necessario modificare manualmente il valore per la modalità di connessione in XMSC_WMQ_CM_CLIENT_UNMANAGED.

Connessioni a un motore di messaggistica SIB (service integration bus) WebSphere Application Server

Le connessioni a un motore di messaggistica SIB (Service Integration Bus) WebSphere Application Server che richiedono l'utilizzo del protocollo SSL (incluso HTTPS) non sono attualmente supportate come codice gestito.

Destinazioni in .NET

In .NET, le destinazioni vengono create in base al tipo di protocollo e possono essere utilizzate solo sul tipo di protocollo per cui sono create.

Sono fornite due funzioni per la creazione di destinazioni, una per gli argomenti e una per le code:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

Queste funzioni sono disponibili sui seguenti due oggetti nell'API:

- ISessione
- XMSFactoryFactory

In entrambi i casi questi metodi possono accettare una stringa di stile URI, che può includere parametri, nel seguente formato:

```
"topic://some/topic/name?priority=5"
```

In alternativa, questi metodi possono accettare solo un nome di destinazione, ovvero un nome senza un argomento: // o coda: // prefisso e senza parametri.

Ciò significa che la seguente stringa di stile URI:

```
CreateTopic("topic://some/topic/name");
```

produce lo stesso risultato del seguente nome destinazione:

```
CreateTopic("some/topic/name");
```

Come per WebSphere Application Server service integration bus JMS, gli argomenti possono essere specificati anche in un formato abbreviato, che include sia *topicname* che *topicspace* ma non può includere parametri:

```
CreateTopic("topicspace:topicname");
```

Proprietà in .NET

Un'applicazione .NET utilizza i metodi nell'interfaccia PropertyContext per richiamare e impostare le proprietà degli oggetti.

L'interfaccia PropertyContext incapsula i metodi che acquisisce e imposta le proprietà. Questi metodi vengono ereditati, direttamente o indirettamente, dalle seguenti classi:

- BytesMessage
- Connessione
- ConnectionFactory
- ConnectionMetaDati
- Destinazione
- MapMessage
- Messaggio
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- sessione
- StreamMessage
- TextMessage

Se un'applicazione imposta il valore di una proprietà, il nuovo valore sostituisce qualsiasi valore precedente della proprietà.

Per ulteriori informazioni sulle proprietà XMS , consultare Proprietà degli oggetti XMS.

Per facilità d'uso, i valori e i nomi delle proprietà XMS in XMS sono predefiniti come costanti pubbliche in una struttura denominata XMSC. I nomi di queste costanti sono nel formato XMSC.*costante*; ad esempio, XMSC.USERID (una costante del nome proprietà) e XMSC.DELIVERY_AS_APP (una costante valore).

Inoltre, è possibile accedere alle costanti IBM MQ utilizzando IBM.XMS.MQC . Se IBM.XMS è già importato, è possibile accedere ai valori per queste proprietà nel modulo MQC.*costante*. Ad esempio, MQC.MQRO_COA_WITH_FULL_DATA.

Inoltre, se si dispone di un'applicazione ibrida che utilizza sia le classi XMS .NET che IBM MQ per .NET e che importa entrambi IBM.XMS e IBM.WMQ WMQ, quindi, è necessario qualificare completamente lo spazio dei nomi struct MQC per garantire che ogni ricorrenza sia univoca.

Alcune funzionalità avanzate non sono attualmente supportate nell'ambiente .NET gestito. Fare riferimento a “Operazioni gestite e non gestite in .NET” a pagina 653 per ulteriori dettagli.

Gestione di proprietà non esistenti in .NET

La gestione delle proprietà non esistenti in XMS .NET è sostanzialmente coerente con le specifiche JMS e anche con le implementazioni C e C + + di XMS.

In JMS, l'accesso a una proprietà non esistente può generare un'eccezione di sistema Java quando un metodo tenta di convertire il valore non esistente (null) nel tipo richiesto. Se una proprietà non esiste, si verificano le seguenti eccezioni:

- La proprietà getStringe la proprietà getObjectrestituiscono un valore null
- La proprietà getBooleanrestituisce false perché Boolean.valueOf(null) restituisce false
- getIntProperty etc.throw java.lang.NumberFormatException perché Integer.valueOf(null) genera l'eccezione

Se una proprietà non esiste in XMS .NET , si verificano le seguenti eccezioni:

- La proprietà GetStringe la proprietà GetObject(e la proprietà GetBytes) restituiscono un valore null (che è uguale a Java)
- GetBooleanLa proprietà genera System.NullReferenceException
- GetIntProperty etc. throws System.NullReferenceException

Questa implementazione è diversa da quella di Java, ma è ampiamente coerente con le specifiche JMS e con le interfacce XMS C e C + +. Come l'implementazione Java , XMS .NET propaga tutte le eccezioni dalla chiamata System.Convert al chiamante. A differenza di Java tuttavia, XMS genera esplicitamente le eccezioni NullReferenceExceptions piuttosto che utilizzare semplicemente il comportamento nativo del framework .NET passando null alle routine di conversione del sistema. Se la tua applicazione imposta una proprietà su una stringa come "abc" e richiama la proprietà GetInt, la System.FormatException generata da Convert.ToInt32("abc") viene propagato al chiamante, che è congruente con Java. MessageFormatL'eccezione viene generata solo se i tipi utilizzati per setProperty e getProperty non sono compatibili. Questo comportamento è coerente anche con Java.

Gestione degli errori in .NET

Le eccezioni XMS .NET sono tutte derivate da System.Exception. Le chiamate al metodo XMS possono generare eccezioni XMS specifiche come l'eccezione MessageFormat, XMSExceptions generali o le eccezioni di sistema come l'eccezione NullReference.

Scrivere le applicazioni per rilevare questi errori, in blocchi catch specifici o in blocchi catch `System.Exception` generali, in base ai requisiti delle applicazioni.

Utilizzo dei listener di messaggi ed eccezioni in .NET

Un'applicazione .NET utilizza un listener di messaggi per ricevere messaggi in modo asincrono e utilizza un listener di eccezione per ricevere una notifica asincrona di un problema con una connessione.

Informazioni su questa attività

La funzionalità dei listener di messaggi e di eccezioni è la stessa per .NET e per C + +. Tuttavia, esistono alcune piccole differenze di implementazione.

Procedura

- Per impostare un listener di messaggi in modo da ricevere i messaggi in modo asincrono, completare la seguente procedura:
  a) Definire un metodo che corrisponda alla firma del delegato del listener messaggi.

     Il metodo definito può essere un metodo statico o di istanza e può essere definito in qualsiasi classe accessibile. La firma del delegato è la seguente:

     ```
     public delegate void MessageListener(IMessage msg);
     ```

     e quindi è possibile definire il metodo come:

     ```
     void SomeMethodName(IMessage msg);
     ```

  b) Creare un'istanza di questo metodo come delegato utilizzando qualcosa di simile al seguente esempio:

     ```
     MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
     ```

  c) Registrare il delegato con uno o più consumer impostandolo sulla proprietà MessageListener del consumer:

     ```
     consumer.MessageListener = OnMsgMethod;
     ```

È possibile rimuovere il delegato impostando nuovamente MessageListener su null:

```
consumer.MessageListener = null;
```

- Per impostare un listener di eccezioni, completare la seguente procedura.

  Il listener di eccezioni funziona in modo molto simile al listener di messaggi, ma ha una diversa definizione di delegato e viene assegnato alla connessione piuttosto che al consumer di messaggi. Questo è lo stesso di C + +.

  a) Definire il metodo.

  La firma del delegato è la seguente:

```
public delegate void ExceptionListener(Exception ex);
```

  e quindi il metodo definito potrebbe essere:

```
void SomeMethodName(Exception ex);
```

  b) Creare un'istanza di questo metodo come delegato utilizzando qualcosa di simile al seguente esempio:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

  c) Registrare il delegato con il collegamento impostandone la proprietà ExceptionListener :

```
connection.ExceptionListener = OnExMethod ;
```

  È possibile rimuovere il delegato reimpostando ExceptionListener su:

```
null:  connection.ExceptionListener = null;
```

Utilizzo degli oggetti amministrati XMS .NET

Gli argomenti in questa sezione forniscono informazioni sugli oggetti gestiti. Le applicazioni XMS possono recuperare le definizioni degli oggetti da un repository di oggetti amministrato centrale e utilizzarli per creare factory di connessione e destinazioni.

Informazioni su questa attività

Questa sezione fornisce informazioni utili per la creazione e la gestione di oggetti gestiti, descrivendo i tipi di repository di oggetti gestiti supportati da XMS . La sezione spiega anche come un'applicazione XMS effettua una connessione a un repository di oggetti gestiti per richiamare gli oggetti gestiti richiesti.

Questa sezione contiene i seguenti argomenti:

- “XMS .NET tipi supportati di repository di oggetti gestiti” a pagina 658
- “Associazione delle proprietà XMS .NET per gli oggetti gestiti” a pagina 658
- “Proprietà XMS .NET richieste per gli oggetti ConnectionFactory gestiti” a pagina 660
- “Proprietà XMS .NET richieste per gli oggetti Destinazione gestiti” a pagina 661
- “XMS .NET creazione di oggetti gestiti” a pagina 662
- “XMS .NET creazione di oggetti InitialContext” a pagina 663
- “proprietà XMS .NET InitialContext” a pagina 663
- “Formato URI per contesti iniziali XMS” a pagina 663
- “Servizio Web di ricerca JNDI per XMS .NET” a pagina 664

- “XMS .NET richiamo di oggetti gestiti” a pagina 665

XMS .NET tipi supportati di repository di oggetti gestiti

Gli oggetti gestiti del file system e LDAP possono essere utilizzati per connettersi a IBM MQ e WebSphere Application Server, mentre la denominazione COS può essere utilizzata per connettersi solo a WebSphere Application Server.

Le directory degli oggetti del file system assumono la forma di oggetti Java Naming Directory Interface (JNDI) serializzati. Gli indirizzari oggetto LDAP sono indirizzari che contengono oggetti JNDI . Le directory di oggetti file system e LDAP possono essere gestite da IBM MQ Explorer, fornito con IBM MQ e versioni successive. Sia il file system che le directory di oggetto LDAP possono essere utilizzate per gestire connessioni client centralizzando le destinazioni e i factory di connessione IBM MQ . L'amministratore di rete può distribuire più applicazioni che fanno riferimento allo stesso repository centrale e che vengono aggiornate automaticamente per riflettere le modifiche alle impostazioni di connessione apportate nel repository centrale.

Una directory di denominazione COS contiene destinazioni e factory di connessione WebSphere Application Server service integration bus e può essere gestita utilizzando la console di gestione WebSphere Application Server . Per un'applicazione XMS per richiamare gli oggetti dalla directory di denominazione COS, è necessario distribuire un servizio Web di ricerca JNDI . Questo servizio Web non è disponibile su tutti gli WebSphere Application Server service integration technologies. Fare riferimento alla documentazione del prodotto per maggiori dettagli.

Nota: Riavviare le connessioni dell'applicazione per rendere effettive le modifiche alla directory dell'oggetto.

Associazione delle proprietà XMS .NET per gli oggetti gestiti

Per consentire alle applicazioni XMS .NET di utilizzare IBM MQ JMS e WebSphere Application Server factory di connessione e definizioni di oggetti di destinazione, le proprietà richiamate da tali definizioni devono essere associate alle proprietà XMS corrispondenti che possono essere impostate sulle factory di connessione e destinazioni XMS .

Per creare, ad esempio, una factory di connessione XMS con proprietà richiamate da un factory di connessione JMS di IBM MQ , le proprietà devono essere associate tra i due.

Tutte le associazioni di proprietà vengono eseguite automaticamente.

Tabella 87 a pagina 658 dimostra le associazioni tra alcune delle proprietà più comuni di factory di connessione e destinazioni. Le proprietà visualizzate in questa tabella sono solo una piccola serie di esempi e non tutte le proprietà visualizzate sono rilevanti per tutti i tipi di connessione e server.

Tabella 87. Esempi di mappatura dei nomi per la factory di connessione e le proprietà di destinazione

Nome proprietà IBM MQ JMS	**XMS nome proprietà.**	**WebSphere Application Server service integration bus nome proprietà.**
PERSISTENZA (PER)	MODALITÀ_DISTRIBUZIONE_XMSCD	
SCADENZA (SCAD)	XMSC_TIME_TO_LIVE	
PRIORITÀ (PRI)	PRIORITÀ_XMSC_XX_ENCODE_CASE_CAPS_LOCK_OFF	
	Nome_HOST_WPM_XMSC	serverName
	XMSC_WPM_BUS_NAME	busName
	SPACE XMSC_WPM_TOPIC_XX_ENCODE_CASE_ONE spazio	topicName

Nota: Le proprietà mostrate in Tabella 88 a pagina 659 sono applicabili per JMS e XMS .NET.

Tabella 88. Proprietà di XMS .NET

Proprietà	**Tipo oggetto**				
	CF	**QCF**	**TCF**	**Coda**	**Argomento**
APPLICATIONNAME	Y	Y	Y	N/A	N/A
ECCEZIONE	Y	Y	Y	N/A	N/A
CCDTURL	Y	Y	Y	N/A	N/A
Canale	Y	Y	Y	N/A	N/A
ConnectionNameList	Y	Y	Y	N/A	N/A
CLIENTRECONNECTOPTIONS	Y	Y	Y	N/A	N/A
CLIENTRECONNECTTIMEOUT	Y	Y	Y	N/A	N/A
clientID	N/A	Y	N/A	N/A	N/A
COMPHDR "1" a pagina 660	Y	N/A	Y	N/A	N/A
COMPMSG "1" a pagina 660	Y	Y	Y	N/A	N/A
CONNOPT "1" a pagina 660	Y	Y	Y	N/A	N/A
CONNTAG "1" a pagina 660	Y	Y	Y	N/A	N/A
DESCRIZIONE "1" a pagina 660	N/A	Y	N/A	Y	Y
SCADENZA "1" a pagina 660	N/A	N/A	N/A	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
nomehost	N/A	Y	N/A	N/A	N/A
LOCALADDRESS	N/A	Y	N/A	N/A	N/A
PERSISTENCE	N/A	N/A	N/A	Y	Y
PORTA	N/A	Y	N/A	N/A	N/A
Priorità"1" a pagina 660	N/A	N/A	N/A	Y	Y
VERSIONEPROVIDER "1" a pagina 660	N/A	Y	N/A	N/A	N/A
QMANAGER	Y	Y	Y	Y	N/A
Coda"1" a pagina 660	N/A	N/A	N/A	Y	N/A

Tabella 88. Proprietà di XMS .NET (Continua)					
Proprietà	**Tipo oggetto**				
	CF	**QCF**	**TCF**	**Coda**	**Argomento**
SHARECONVALLOWED	Y	Y	Y	N/A	N/A
ARGOMENTO "1" a pagina 660	N/A	N/A	N/A	N/A	Y
Transport "1" a pagina 660	N/A	Y	N/A	N/A	N/A

Nota:

1. Queste proprietà non dispongono di proprietà a livello di applicazione ma possono essere facoltativamente impostate utilizzando le proprietà gestite.

OutboundSNI proprietà

V 9.3.0

Da IBM MQ 9.3.0, è possibile impostare la proprietà XMSC_WMQ_OUTBOUND_SNI, che imposta la proprietà **OutboundSNI** in una applicazione.

XMSC_WMQ_OUTBOUND_SNI_PROPERTY assume i seguenti valori:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL, che si associa a "CHANNEL"
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME, che corrisponde a "HOSTNAME"
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK, che corrisponde a "*"

Inoltre, è possibile impostare la proprietà **OutboundSNI** utilizzando la variabile di ambiente MQOUTBOUND_SNI, che assume i seguenti valori:

- CHANNEL
- HOSTNAME
- *

Nota: La proprietà viene impostata per default su XMSC_WMQ_OUTBOUND_SNI_CHANNEL se non è impostato alcun valore specifico.

L'ordine di precedenza per l'impostazione della proprietà **OutboundSNI** nel nodo gestito è:

1. proprietà Livello applicazione
2. Variabile di ambiente

Per la proprietà **OutboundSNI** nel nodo non gestito, è supportata solo `mqclient.ini` .

Proprietà XMS .NET richieste per gli oggetti ConnectionFactory gestiti

Quando un'applicazione crea una factory di connessione, è necessario definire un numero di proprietà per creare una connessione a un server di messaggistica.

Le proprietà elencate nelle seguenti tabelle sono il minimo richiesto per un'applicazione da impostare per creare una connessione a un server di messaggistica. Se si desidera personalizzare il modo in cui viene creata una connessione, l'applicazione può impostare qualsiasi proprietà aggiuntiva dell'oggetto ConnectionFactory come necessario. Per ulteriori informazioni, consultare Proprietà di ConnectionFactory. Viene incluso un elenco completo delle proprietà disponibili.

Connessione a un gestore code IBM MQ

Tabella 89. Impostazioni delle proprietà per gli oggetti ConnectionFactory gestiti per le connessioni a un gestore code IBM MQ	
proprietà XMS obbligatoria	**Proprietà IBM MQ JMS equivalente obbligatoria**
XMSC_CONNECTION_TYPE	XMS consente di eseguire questa operazione dal nome della classe factory di connessione e dalla proprietà TRANSPORT (TRAN).
NOME_HOST WMQ_XMSC	NOME HOST (HOST)
XMSC_WMQ_PORT	PORTA
XMSC_WMQ_QUEUE_MANAGER	Il nome del gestore code

Connessione in tempo reale a un broker

Tabella 90. Impostazioni di proprietà per oggetti ConnectionFactory gestiti per connessioni in tempo reale a un broker	
ObbligatorioXMS	**Proprietà IBM MQ JMS equivalente obbligatoria**
XMSC_CONNECTION_TYPE	XMS consente di eseguire questa operazione dal nome della classe factory di connessione e dalla proprietà TRANSPORT (TRAN).
XMSC_RTT_NOME_HOST	NOME HOST (HOST)
PORTA_RTT_XMSC	PORTA

Connessione a un WebSphere Application Server service integration bus

Tabella 91. Impostazioni delle propriet ... per gli oggetti ConnectionFactory gestiti per le connessioni a un WebSphere Application Server service integration bus	
XMS proprietà	**Descrizione**
XMSC_CONNECTION_TYPE	Il tipo di server di messaggistica a cui si connette un'applicazione.. Ciò è determinato dal nome della classe factory di connessione.
XMSC_WPM_BUS_NAME	Per una factory di connessione, il nome del bus di integrazione del servizio a cui si connette l'applicazione oppure, per una destinazione, il nome del bus di integrazione del servizio in cui esiste la destinazione.

Proprietà XMS .NET richieste per gli oggetti Destinazione gestiti

Un'applicazione che sta creando una destinazione deve impostare diverse proprietà dell'applicazione su un oggetto Destinazione gestito.

Tabella 92. Impostazioni di proprietà per gli oggetti di destinazione gestiti		
Tipo di connessione	**Proprietà**	**Descrizione**
IBM MQ Gestore code	CODA (QU)	La coda a cui si desidera connettersi
	ARGOMENTO (SUPERIORE)	L'argomento utilizzato dall'applicazione come destinazione

Tabella 92. Impostazioni di proprietà per gli oggetti di destinazione gestiti (Continua)		
Tipo di connessione	**Proprietà**	**Descrizione**
Connessione in tempo reale a un broker	ARGOMENTO (SUPERIORE)	L'argomento utilizzato dall'applicazione come destinazione
WebSphere Application Server service integration bus	topicName queueName	Se l'applicazione si connette a un argomento Se l'applicazione si connette a una coda

XMS .NET creazione di oggetti gestiti

Le definizioni dell'oggetto ConnectionFactory e Destination richieste dalle applicazioni XMS per stabilire una connessione a un server di messaggistica devono essere create utilizzando gli strumenti di gestione appropriati.

Prima di iniziare

Per ulteriori dettagli sui diversi tipi di repository di oggetti gestiti supportati da XMS , consultare “XMS .NET tipi supportati di repository di oggetti gestiti” a pagina 658.

Informazioni su questa attività

Per creare gli oggetti amministrati per IBM MQ utilizzare lo strumento di gestione JMS (JMSAdmin) IBM MQ Explorer o IBM MQ .

Per creare gli oggetti gestiti per IBM MQ o IBM Integration Bus, utilizzare lo strumento di amministrazione JMS IBM MQ (JMSAdmin).

Per creare oggetti gestiti per WebSphere Application Server service integration bus, utilizzare la console di gestione WebSphere Application Server .

Nella strumentazione amministrativa la proprietà è nota come **APPLICATIONNAME** o **APPNAME** in breve.

Nota: Non è possibile utilizzare JMSAdmin per impostare TRANSPORT (UNMANAGED). Pertanto, per ottenere un client XMS non gestito utilizzando un nome applicazione scelto in modo amministrativo, è necessario immettere il seguente comando:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

La seguente procedura riepiloga le operazioni da eseguire per creare gli oggetti gestiti.

Procedura

1. Creare una factory di connessione e definire le proprietà necessarie per creare una connessione dall'applicazione al server scelto.

   Le proprietà minime richieste da XMS per stabilire una connessione sono definite in “Proprietà XMS .NET richieste per gli oggetti ConnectionFactory gestiti” a pagina 660.
2. Creare la destinazione richiesta sul server di messaggistica, a cui si connette l'applicazione:
   - Per una connessione a un gestore code IBM MQ , creare una coda o un argomento.
   - Per una connessione in tempo reale a un broker, creare un argomento.
   - Per una connessione a un WebSphere Application Server service integration bus, creare una coda o un argomento.

   Le proprietà minime richieste da XMS per stabilire una connessione sono definite in “Proprietà XMS .NET richieste per gli oggetti Destinazione gestiti” a pagina 661.

XMS .NET creazione di oggetti InitialContext

Un'applicazione deve creare un contesto iniziale da utilizzare per stabilire una connessione al repository di oggetti gestiti per richiamare gli oggetti gestiti richiesti.

Informazioni su questa attività

Un oggetto InitialContext incapsula una connessione al repository. L'API XMS fornisce i metodi per eseguire le seguenti attività:

- Crea un oggetto InitialContext
- Ricercare un oggetto gestito nel repository di oggetti amministrato.

Procedura

- Per ulteriori dettagli sulla creazione di un oggetto InitialContext , consultare InitialContext per .NET e Proprietà di InitialContext.

proprietà XMS .NET InitialContext

I parametri del costruttore InitialContext includono l'ubicazione del repository di oggetti gestiti, forniti come URI (uniform resource indicator). Affinché un'applicazione stabilisca una connessione al repository, potrebbe essere necessario fornire più informazioni rispetto a quelle contenute nell'URI.

In JNDI e nell'implementazione .NET di XMS, le informazioni aggiuntive vengono fornite in un ambiente Hashtable al costruttore.

L'ubicazione del repository di oggetti amministrati è definita nella proprietà XMSC_IC_URL . Questa proprietà viene generalmente passata alla chiamata di creazione, ma può essere modificata per connettersi a una directory di denominazione diversa prima della ricerca. Per i contesti FileSystem o LDAP, questa proprietà definisce l'indirizzo della directory. Per la denominazione COS, questo è l'indirizzo del servizio Web che utilizza queste proprietà per connettersi alla directory JNDI.

Le seguenti proprietà vengono inoltrate senza modifiche al servizio Web che le utilizzerà per connettersi alla directory JNDI.

- URL_PROVIDER_IC_XMSC
- CREDENZIALI XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- PROTOCOLLO_SICUREZZA_XMSC_ICA

Formato URI per contesti iniziali XMS

L'ubicazione del repository di oggetti gestiti viene fornita come URI (uniform resource indicator). Il formato dell'URI dipende dal tipo di contesto.

Contesto FileSystem

Per il contesto FileSystem , l' URL fornisce l'ubicazione della directory basata sul file system. La struttura dell'URL (URL) è quella definita da RFC 1738, *Uniform Resource Locators (URL)*: l' URL ha il prefisso `file://` e la sintassi che segue questo prefisso è una definizione valida di un file che può essere aperto nel sistema su cui è in esecuzione XMS .

Questa sintassi può essere specifica della piattaforma e può utilizzare i separatori '/ o' \ '. Se si utilizza '\', è necessario eseguire l'escape di ciascun separatore utilizzando un ulteriore '\'. Ciò impedisce al framework .NET di tentare di interpretare il separatore come un carattere escape per quanto segue.

Questi esempi illustrano questa sintassi:

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

Contesto LDAP

Per il contesto LDAP, la struttura di base dell' URL è quella definita da RFC 2255, *The LDAP URL Format*, con il prefisso non sensibile al maiuscolo / minuscolo `ldap://`

La sintassi precisa viene illustrata nel seguente esempio:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Questa sintassi è quella definita in RFC ma non supporta attributi, ambiti, filtri o estensioni.

Esempi di questa sintassi sono:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

Contesto WSS

Per il contesto WSS, l'URL (URL) è nel formato di endpoint dei servizi Web, con il prefisso `http://`.

In alternativa, è possibile utilizzare il prefisso `cosnaming://` o `wsvc://`,

Questi due prefissi vengono interpretati nel senso che si sta utilizzando un contesto WSS con l' URL a cui si accede tramite http, il che consente al tipo di contesto iniziale di derivare facilmente direttamente dall' URL.

Esempi di questa sintassi sono:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

Servizio Web di ricerca JNDI per XMS .NET

Per accedere a una directory di denominazione COS da XMS, un JNDI servizio Web di ricerca deve essere distribuito su un server WebSphere Application Server service integration bus . Questo servizio web converte le informazioni Java dal servizio di denominazione COS in un modulo che le applicazioni XMS possono leggere.

Il servizio Web viene fornito nel file EAR (Enterprise Archive) SIBXJndiLookupEAR.ear, presente nella directory di installazione. Per la release corrente di IBM MQ Message Service Client (XMS) for .NET, è possibile trovare SIBXJndiLookupEAR.ear nella directory *install_dir*`\java\lib` . Questo può essere installato all'interno di un server WebSphere Application Server service integration bus utilizzando la console di gestione o lo strumento di script wsaadmin. Per ulteriori informazioni sulla distribuzione delle applicazioni del servizio Web, fare riferimento alla documentazione del prodotto.

Per definire il servizio Web nelle applicazioni XMS , è sufficiente impostare la proprietà XMSC_IC_URL dell'oggetto InitialContext sull'endpoint del servizio Web URL. Ad esempio, se il servizio Web viene distribuito su un host server denominato MyServer, un esempio di un endpoint del servizio Web URL:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

L'impostazione della proprietà XMSC_IC_URL consente alle chiamate di ricerca InitialContext di richiamare il servizio Web sull'endpoint definito, che a sua volta ricerca l'oggetto gestito richiesto dal servizio di denominazione COS.

Le applicazioni .NET possono utilizzare il servizio Web. La distribuzione lato server è la stessa per XMS C, /C++ e XMS .NET. XMS .NET richiama il servizio Web direttamente tramite Microsoft .NET Framework.

XMS .NET richiamo di oggetti gestiti

XMS richiama un oggetto gestito dal repository utilizzando l'indirizzo fornito quando viene creato l'oggetto InitialContext o nelle proprietà InitialContext .

Gli oggetti da richiamare possono avere i seguenti tipi di nomi:

- Un nome semplice che descrive l'oggetto Destinazione, ad esempio, una destinazione coda denominata SalesOrders
- Un nome composito, che può essere costituito da SubContexts, separato da '/', e deve terminare con il nome oggetto. Un esempio di nome composto è "Warehouse/PickLists/DispatchQueue2" dove Warehouse e Picklists sono SubContexts nella directory di denominazione e DispatchQueue2 è il nome di un oggetto di destinazione.

Come impedire alle applicazioni di utilizzare una versione XMS più recente

Per impostazione predefinita, quando viene installata una versione XMS più recente, le applicazioni che utilizzano la versione precedente passano automaticamente alla versione più recente senza dover recompile.However, è possibile evitare che le applicazioni utilizzino la versione più recente impostando un attributo nel file di configurazione dell'applicazione.

Informazioni su questa attività

La funzione di coesistenza di più versioni garantisce che l'installazione di una versione XMS più recente non sovrascriva la versione XMS precedente. Invece, più istanze di assembly XMS .NET simili coesistono in GAC (Global Assembly Cache), ma hanno numeri di versione diversi. Internamente, il GAC usa un file della politica per instradare le chiamate dell'applicazione all'ultima versione di XMS. Le applicazioni vengono eseguite senza necessità di ricompilazione e possono utilizzare nuove funzioni disponibili nella nuova versione di XMS .NET .

Procedura

- Se è necessario che un'applicazione utilizzi la vecchia versione di XMS .NET , impostare l'attributo `publisherpolicy` su no nel file di configurazione dell'applicazione.

  Nota: Un file di configurazione dell'applicazione è un file con un nome composto dal nome del programma eseguibile a cui il file si riferisce, con il suffisso .config. Ad esempio, il file di configurazione dell'applicazione per text.exe avrà il nome text.exe.config.

  In qualsiasi momento, tuttavia, tutte le applicazioni di un sistema utilizzano la stessa versione di XMS .NET.

Protezione delle comunicazioni per applicazioni XMS

Questa sezione fornisce informazioni sull'impostazione delle comunicazioni protette per consentire alle applicazioni XMS di connettersi tramite SSL (Secure Sockets Layer) a un motore di messaggistica WebSphere Application Server service integration bus o a un gestore code IBM MQ .

Informazioni su questa attività

La sezione contiene i seguenti argomenti:

- “Connessioni protette a un gestore code IBM MQ” a pagina 666
- “Associazioni di nomi CipherSuite e CipherSpec per le connessioni XMS ad un gestore code IBM MQ” a pagina 667
- “Connessioni sicure a un motore di messaggistica WebSphere Application Server service integration bus” a pagina 667
- “Associazioni di nomi CipherSuite e CipherSpec per le connessioni a WebSphere Application Server service integration bus” a pagina 668

Connessioni protette a un gestore code IBM MQ

Per consentire a un'applicazione di XMS .NET di stabilire connessioni sicure a un gestore code IBM MQ , le proprietà pertinenti devono essere definite nell'oggetto ConnectionFactory .

Il protocollo utilizzato nella negoziazione di codifica può essere SSL (Secure Sockets Layer) o TLS (Transport Layer Security), a seconda della CipherSuite specificata nell'oggetto ConnectionFactory .

Le proprietà di ConnectionFactory per le connessioni che utilizzano SSL a un gestore code IBM MQ , con una breve descrizione, sono riportate nella seguente tabella:

Tabella 93. Proprietà di ConnectionFactory per connessioni a un gestore code IBM MQ tramite SSL

Nome della proprietà	**Descrizione**
XMSC_WMQ_SSL_CERT_STORES	L'ubicazione dei server che contengono i CRL (cettificate revocation list) da utilizzare su una connessione SSL a un gestore code.
XMSC_WMQ_SSL_CIPHER_SPEC	Il nome della CipherSpec da utilizzare su una connessione protetta a un gestore code.
XMSC_WMQ_SSL_CIPHER_SUITE	Il nome della CipherSuite da utilizzare su una connessione TLS a un gestore code. Il protocollo utilizzato nella negoziazione della connessione protetta dipende dalla CipherSuite specificata.
XMSC_WMQ_SSL_CRYPTO_HW	I dettagli di configurazione per l'hardware di crittografia connesso al sistema client.
XMSC_WMQ_SSL_FIPS_REQUIRED	Il valore della proprietà determina se un'applicazione può o meno utilizzare delle suite di crittografia non conformi a FIPS. Se questa proprietà è impostata su true, per la connessione client-server vengono utilizzati solo gli algoritmi FIPS.
XMSC_WMQ_SSL_KEY_REPOSITORY	L'ubicazione del file del database delle chiavi in cui sono memorizzati chiavi e certificati.
XMSC_WMQ_SSL_KEY_RESETCOUNT	Il KeyResetCount rappresenta il numero totale di byte non crittografati inviati e ricevuti in una conversazione SSL prima che venga rinegoziata la chiave segreta.
XMSC_WMQ_SSL_PEER_NAME	Il nome peer da utilizzare su una connessione SSL a un gestore code.

Associazioni di nomi CipherSuite e CipherSpec per le connessioni XMS ad un gestore code IBM MQ

InitialContext converte tra la proprietà del factory di connessione JMSAdmin SSLCIPHERSUITE e il XMS quasi equivalente XMSC_WMQ_SSL_CIPHER_SPEC. Una traduzione simile è necessaria se si specifica un valore per XMSC_WMQ_SSL_CIPHER_SUITE ma si omette il valore per XMSC_WMQ_SSL_CIPHER_SPEC.

Tabella 94 a pagina 667 elenca i CipherSpecs disponibili e i relativi equivalenti JSSE CipherSuite .

Tabella 94. CipherSpecs disponibili e i relativi equivalenti JSSE CipherSuite

CipherSpec	**CipherSuite JSSE equivalente**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

Nota: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleto. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.

Connessioni sicure a un motore di messaggistica WebSphere Application Server service integration bus

Per abilitare un'applicazione XMS .NET per stabilire connessioni sicure a un motore di messaggistica WebSphere Application Server service integration bus , le proprietà pertinenti devono essere definite nell'oggetto ConnectionFactory .

XMS fornisce il supporto SSL e HTTPS per le connessioni a un WebSphere Application Server service integration bus. SSL e HTTPS forniscono connessioni sicure per l'autenticazione e la riservatezza.

Come la sicurezza WebSphere , la sicurezza XMS è configurata in base agli standard di sicurezza JSSE e alle convenzioni di denominazione, che includono l'utilizzo di CipherSuites per specificare gli algoritmi utilizzati quando si negozia una connessione sicura. Il protocollo utilizzato nella negoziazione della codifica può essere SSL o TLS, a seconda della CipherSuite specificata nell'oggetto ConnectionFactory .

Tabella 95 a pagina 667 elenca le proprietà che devono essere definite nell'oggetto ConnectionFactory .

Tabella 95. Proprietà di ConnectionFactory per le connessioni sicure a un motore di messaggistica WebSphere Application Server service integration bus

Nome della proprietà	**Descrizione**
XMSC_WPM_SSL_CIPHER_SUITE	Il nome della CipherSuite da utilizzare su una connessione TLS a un motore di messaggistica WebSphere Application Server service integration bus . Il protocollo utilizzato nella negoziazione della connessione protetta dipende dalla CipherSuite specificata.
XMSC_WPM_SSL_KEYRING_LABEL	Il certificato da utilizzare quando si esegue l'autenticazione presso il server.

Di seguito viene riportato un esempio di proprietà ConnectionFactory per le connessioni sicure a un motore di messaggistica WebSphere Application Server service integration bus :

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
```

```
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

Dove chain_name deve essere impostato su BootstrapTunneledSecureMessaging o BootstrapSecureMessaging e port_number è il numero della porta su cui il server bootstrap è in ascolto per le richieste in entrata.

Di seguito è riportato un esempio di proprietà ConnectionFactory per le connessioni sicure a un motore di messaggistica WebSphere Application Server service integration bus con valori di esempio inseriti:

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

Associazioni di nomi CipherSuite e CipherSpec per le connessioni a WebSphere Application Server service integration bus

Poiché IBM Global Security Kit (GSKit) utilizza CipherSpecs invece di CipherSuites, i nomi CipherSuite in stile JSSE specificati nella proprietà XMSC_WPM_SSL_CIPHER_SUITE devono essere associati ai nomi CipherSpec in stile GSKit.

Tabella 96 a pagina 668 elenca la CipherSpec equivalente per ogni CipherSuitericonosciuto.

Tabella 96. CipherSuites disponibili e relativi CipherSpecs equivalenti

CipherSuite	**CipherSpec equivalente**
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

Nota: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA è obsoleto. Tuttavia, può essere ancora utilizzato per trasferire fino a 32 GB di dati prima che la connessione venga terminata con l'errore AMQ9288. Per evitare questo errore, è necessario evitare di utilizzare il triplo DES o abilitare la reimpostazione della chiave segreta quando si utilizza questo CipherSpec.

XMS messaggi

Questa sezione descrive la struttura e il contenuto dei messaggi XMS e spiega come le applicazioni elaborano i messaggi XMS .

Questa sezione contiene i seguenti argomenti:

- “Parti di un messaggio XMS” a pagina 668
- “Campi di intestazione in un messaggio XMS” a pagina 669
- “Proprietà di un messaggio XMS” a pagina 670
- “Il contenuto di un messaggio XMS” a pagina 673
- “Selettori di messaggi” a pagina 676
- “Mappatura dei messaggi XMS sui messaggi IBM MQ” a pagina 677

Parti di un messaggio XMS

Un messaggio XMS è composto da un'intestazione, una serie di proprietà e un corpo.

Intestazione
L'intestazione di un messaggio contiene campi e tutti i messaggi contengono la stessa serie di campi intestazione. XMS e le applicazioni utilizzano i valori dei campi di intestazione per identificare e instradare i messaggi. Per ulteriori informazioni sui campi di intestazione, consultare “Campi di intestazione in un messaggio XMS” a pagina 669.

Insieme di proprietà
Le proprietà di un messaggio specificano ulteriori informazioni sul messaggio. Sebbene tutti i messaggi abbiano la stessa serie di campi di intestazione, ogni messaggio può avere una serie diversa di proprietà. Per ulteriori informazioni, fare riferimento a “Proprietà di un messaggio XMS” a pagina 670.

Corpo
Il corpo di un messaggio contiene dati dell'applicazione. Per ulteriori informazioni, fare riferimento a “Il contenuto di un messaggio XMS” a pagina 673.

Un'applicazione può selezionare i messaggi che desidera ricevere. Utilizzando i selettori di messaggi, che specifica i criteri di selezione. I criteri possono essere basati sui valori di determinati campi di intestazione e sui valori di qualsiasi proprietà di un messaggio. Per ulteriori informazioni sui selettori di messaggi, consultare “Selettori di messaggi” a pagina 676.

Campi di intestazione in un messaggio XMS

Per consentire a una applicazione XMS di scambiare messaggi con un'applicazione WebSphere JMS , l'intestazione di un messaggio XMS contiene i campi di intestazione del messaggio JMS .

I nomi di questi campi di intestazione iniziano con il prefisso JMS. Per una descrizione dei campi di intestazione del messaggio JMS , consultare *Java Message Service Specifica*.

XMS implementa i campi di intestazione del messaggio JMS come attributi di un oggetto Messaggio. Ogni campo di intestazione ha i propri metodi per impostare e ottenere il proprio valore. Per una descrizione di questi metodi, consultare IMessage. Un campo di intestazione è sempre leggibile e scrivibile.

Tabella 97 a pagina 669 elenca i campi di intestazione del messaggio JMS e indica in che modo il valore di ciascun campo è impostato per un messaggio trasmesso. Alcuni dei campi vengono impostati automaticamente da XMS quando un'applicazione invia un messaggio o, nel caso di JMSRenei, quando un'applicazione riceve un messaggio.

Tabella 97. Campi di intestazione del messaggio JMS.]

Nome del campo di intestazione del messaggio JMS	**Modalità di impostazione del valore per un messaggio trasmesso (nel formato *method* [*class*])**
JMSCorrelationID	Impostare JMSCorrelationID [Messaggio]
JMSDeliveryMode	Invia [MessageProducer]
JMSDestination	Invia [MessageProducer]
JMSExpiration	Invia [MessageProducer]
JMSMessageID	Invia [MessageProducer]
JMSPriority	Invia [MessageProducer]
JMSRedelivered	Ricevi [MessageConsumer]
JMSReplyTo	Impostare JMSReplyTo [Messaggio]
JMSTimestamp	Invia [MessageProducer]
JMSType	Imposta JMSType [Messaggio]

Proprietà di un messaggio XMS

XMS supporta tre tipi di proprietà del messaggio: JMS proprietà definite IBM e proprietà definite dall'applicazione.

Un'applicazione XMS può scambiare messaggi con un'applicazione WebSphere JMS perché XMS supporta le seguenti proprietà predefinite di un oggetto Messaggio:

- Le stesse proprietà definite da JMSsupportate da WebSphere JMS . I nomi di queste proprietà iniziano con il prefisso JMSX.
- Le stesse proprietà definite da IBMsupportate da WebSphere JMS . I nomi di queste proprietà iniziano con il prefisso JMS_IBM_.

Ogni proprietà predefinita ha due nomi:

- Un nome JMS , per una proprietà definita da JMSo un nome WebSphere JMS , per una proprietà definita da IBM.

  Questo è il nome con cui la proprietà è nota in JMS o WebSphere JMSed è anche il nome che viene trasmesso con un messaggio che ha questa proprietà. Un'applicazione XMS utilizza questo nome per identificare la proprietà in un'espressione del selettore messaggi.
- Un nome XMS per identificare la proprietà in tutte le situazioni tranne che in un'espressione del selettore messaggi. Ogni nome XMS è definito come costante con nome nella classe `IBM.XMS.XMSC` . Il valore della costante denominata è il nome JMS o WebSphere JMS corrispondente.

Oltre alle proprietà predefinite, un'applicazione XMS può creare e utilizzare la propria serie di proprietà del messaggio. Queste proprietà sono denominate *proprietà definite dall'applicazione* .

Dopo che un'applicazione crea un messaggio, le proprietà del messaggio sono leggibili e scrivibili. Le proprietà rimangono leggibili e scrivibili dopo che l'applicazione invia il messaggio. Quando un'applicazione riceve un messaggio, le proprietà del messaggio sono di sola lettura. Se un'applicazione richiama il metodo `Clear Properties` della classe Message quando le proprietà di un messaggio sono di sola lettura, le proprietà diventano leggibili e scrivibili. Il metodo cancella anche le proprietà.

Il messaggio ricevuto, quando inoltrato dopo aver cancellato le proprietà del messaggio, si comporterà in modo coerente con il funzionamento dell'inoltro di un BytesMessage standard WMQ XMS per .NET con le proprietà del messaggio ripulite.

Ciò, tuttavia, non è consigliato poiché le seguenti proprietà andranno perse:

- Valore della proprietà JMS_IBM_Encoding che indica che i dati del messaggio non possono essere decodificati in modo significativo.
- Valore della proprietà JMS_IBM_Format, che indica che il concatenamento dell'intestazione tra l'intestazione del messaggio (MQMD o la nuova MQRFH2) e le intestazioni esistenti verrebbe interrotto.

Per determinare i valori di tutte le proprietà di un messaggio, un'applicazione può chiamare il metodo Get Properties della classe Message. Il metodo crea un iteratore che incapsula un elenco di oggetti proprietà, dove ogni oggetto proprietà rappresenta una proprietà del messaggio. L'applicazione può quindi utilizzare i metodi della classe Iterator per richiamare ciascun oggetto Proprietà a turno e può utilizzare i metodi della classe Proprietà per recuperare il nome, il tipo di dati e il valore di ogni proprietà.

Proprietà definite da JMSdi un messaggio

Diverse proprietà definite da JMS di un messaggio sono supportate sia da XMS che WebSphere JMS.

Tabella 98 a pagina 671 elenca le proprietà definite JMS di un messaggio supportate sia da XMS che WebSphere JMS. Per una descrizione delle proprietà definite da JMS, vedere *Java Message Service Specifica*. Le proprietà definite da JMSnon sono valide per una connessione in tempo reale a un broker.

La tabella specifica il tipo di dati di ciascuna proprietà e indica in che modo il valore della proprietà è impostato per un messaggio trasmesso. Alcune delle proprietà vengono impostate automaticamente da XMS quando un'applicazione invia un messaggio o, nel caso di JMSXDeliveryCount, quando un'applicazione riceve un messaggio.

Tabella 98. Proprietà definite da JMSdi un messaggio			
Nome XMS della proprietà definita da JMS	**Nome JMS**	**Tipo dati**	**Modalità di impostazione del valore per un messaggio trasmesso (nel formato *method* [*class*])**
IDAPP_JMSX	JMSXAppID	System.String	Invia [MessageProducer]
CONTEGGIO_DISTRIBUZIONE_JMSX	JMSXDeliveryCount	System.Int32	Ricevi [MessageConsumer]
IDGRUPPO_JMS	JMSXGroupID	System.String	Imposta proprietà stringa [PropertyContext]
GROUPSEQ JMS	JMSXGroupSeq	System.Int32	Imposta proprietà numero intero [PropertyContext]
IDUSER_JMS	JMSXUserID	System.String	Invia [MessageProducer]

Proprietà definite da IBMdi un messaggio

Diverse proprietà IBMdefinite di un messaggio sono supportate da XMS e WebSphere JMS.

Tabella 99 a pagina 671 elenca le IBM proprietà definite di un messaggio supportate sia da XMS che WebSphere JMS. Per ulteriori informazioni sulle proprietà definite da IBM, consultare IBM MQ o la documentazione del prodotto WebSphere Application Server .

La tabella specifica il tipo di dati di ciascuna proprietà e indica in che modo il valore della proprietà è impostato per un messaggio trasmesso. Alcune proprietà vengono impostate automaticamente da XMS quando un'applicazione invia un messaggio.

Tabella 99. IBMdi un messaggio			
XMS nome della IBM proprietà definita	**Nome WebSphere JMS**	**Tipo dati**	**Modalità di impostazione del valore per un messaggio trasmesso (nel formato *method* [*class*])**
JMS_IBM_CHARACTER_SET	Serie di caratteristiche IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
JMS_IBM_Encoding	JMS_IBM_Encoding	System.Int32	Imposta proprietà numero intero [PropertyContext]
MESSAGGIO_IBM_JMS	JMS_IBM_ExceptionMessage	System.String	Ricevi [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Ricevi [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Ricevi [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEM DESTINAZIONE	Destinazione JMS_IBM_ExceptionProblem	System.String	Ricevi [MessageConsumer]
JMS_IBM_FEEDBACK	Feedback IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]

Tabella 99. IBMdi un messaggio (Continua)

XMS nome della IBM proprietà definita	Nome WebSphere JMS	Tipo dati	Modalità di impostazione del valore per un messaggio trasmesso (nel formato *method* [*class*])
JMS_IBM_FORMAT	Formato_IBM_JMS	System.String	Imposta proprietà stringa [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Imposta proprietà numero intero [PropertyContext]
TIPO_IBM_JMS	JMS_IBM_MsgType	System.Int32	Imposta proprietà numero intero [PropertyContext]
TIPO_PUTAPPL_IBM_JMS	JMS_IBM_PutApplTipo	System.Int32	Invia [MessageProducer]
DATA_IBM_JMS	JMS_IBM_PutDate	System.String	Invia [MessageProducer]
PUTORA_IBM_JMS	JMS_IBM_PutTime	System.String	Invia [MessageProducer]
COA IBM_REPORT_JMS	COA report IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
COD IBM REPORT_JMS	COD report IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Imposta proprietà numero intero [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	Eccezione JMS_IBM_Report_Exception	System.Int32	Imposta proprietà numero intero [PropertyContext]
SCADENZA_REPORT_IBM_JMS	Scadenza report IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
NAN_IBM_REPORT_JMS	NAN report IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
PAN IBM REPORT_JMS	PAN report IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ ID	ID correlazione password report IBM JMS	System.Int32	Imposta proprietà numero intero [PropertyContext]
ID_IBM_REPORT_PASS_MSG_JMS	ID_ms_ms_IBM_Report_Pass_Msg_ID	System.Int32	Imposta proprietà numero intero [PropertyContext]
IDMESSAGGIO_IBM_SISTEMA_JMS	JMS_IBM_System_MessageID	System.String	Invia [MessageProducer]

Proprietà definite dall'applicazione di un messaggio

Un'applicazione XMS può creare e utilizzare la propria serie di proprietà del messaggio. Quando un'applicazione invia un messaggio, anche queste proprietà vengono trasmesse con il messaggio. Un'applicazione ricevente, utilizzando i selettori di messaggi, può quindi selezionare quali messaggi desidera ricevere in base ai valori di queste proprietà.

Per consentire a un'applicazione WebSphere JMS di selezionare ed elaborare i messaggi inviati da un'applicazione XMS , è necessario che il nome di una proprietà definita dall'applicazione sia conforme alle regole per la formazione di identificatori nelle espressioni del selettore messaggi. Per ulteriori

informazioni, consultare “Selettori di messaggi in JMS” a pagina 147. Il valore di una proprietà definita dall'applicazione deve avere uno dei seguenti tipi di dati: System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Doubleo System.String.

Il contenuto di un messaggio XMS

Il corpo di un messaggio contiene dati dell'applicazione. Tuttavia, un messaggio non può avere alcun corpo e comprende solo i campi di intestazione e le proprietà.

XMS supporta cinque tipi di messaggio:

Byte
Il corpo contiene un flusso di byte. Un messaggio con questo tipo di corpo viene denominato *messaggio byte*. L'interfaccia IBytesMessage contiene i metodi per elaborare il contenuto di un messaggio di byte.

Associazione
Il corpo contiene una serie di coppie nome - valore, in cui ogni valore ha un tipo di dati associato. Un messaggio con questo tipo di corpo viene denominato *messaggio di associazione*. L'interfaccia IMapMessage contiene i metodi per elaborare il contenuto di un messaggio di associazione.

Oggetto
Il corpo contiene un oggetto Java o .NET serializzato. Un messaggio con questo tipo di corpo viene chiamato *messaggio oggetto*. L'interfaccia IObjectMessage contiene i metodi per elaborare il testo di un messaggio oggetto.

Flusso
Il corpo contiene un flusso di valori, in cui ciascun valore ha un tipo di dati associato. Un messaggio con questo tipo di corpo viene chiamato *messaggio di flusso*. L'interfaccia IStreamMessage contiene i metodi per elaborare il contenuto di un messaggio di flusso.

Testo
Il body contiene una stringa. Un messaggio con questo tipo di corpo viene denominato *messaggio di testo*. L'interfaccia ITextMessage contiene i metodi per elaborare il contenuto di un messaggio di testo.

L'interfaccia IMessage è l'elemento principale di tutti gli oggetti messaggio e può essere utilizzata nelle funzioni di messaggistica per rappresentare qualsiasi tipo di messaggio XMS .

Per informazioni sulla dimensione e sui valori massimo e minimo di ciascuno di tali tipi di dati, consultare Tabella 84 a pagina 648.

Messaggi byte

Il corpo di un messaggio di byte contiene un flusso di byte. Il corpo contiene solo i dati effettivi ed è responsabilità delle applicazioni di invio e di ricezione interpretare tali dati.

I messaggi di byte sono utili se un'applicazione XMS deve scambiare messaggi con applicazioni che non utilizzano l'interfaccia di programmazione dell'applicazione XMS o JMS .

Dopo che un'applicazione crea un messaggio di byte, il corpo del messaggio è di sola scrittura. L'applicazione assembla i dati dell'applicazione nel corpo richiamando i metodi di scrittura appropriati dell'interfaccia IBytesMessage per .NET. Ogni volta che l'applicazione scrive un valore nel flusso di messaggi di byte, il valore viene assemblato immediatamente dopo il precedente valore scritto dall'applicazione. XMS mantiene un cursore interno per ricordare la posizione dell'ultimo byte assemblato.

Quando l'applicazione invia il messaggio, il corpo del messaggio diventa di sola lettura. In questa modalità, l'applicazione può inviare il messaggio ripetutamente.

Quando un'applicazione riceve un messaggio di byte, il corpo del messaggio è di sola lettura. L'applicazione può utilizzare i metodi di lettura appropriati dell'interfaccia IBytesMessage per leggere il contenuto del flusso di messaggi di byte. L'applicazione legge i byte in sequenza e XMS mantiene un cursore interno per ricordare la posizione dell'ultimo byte letto.

Se un'applicazione richiama il metodo Reset dell'interfaccia IBytesMessage quando il corpo di un messaggio di byte è scrivibile, il corpo diventa di sola lettura. Il metodo riposiziona anche il cursore all'inizio del flusso di messaggi di byte.

Se un'applicazione richiama il metodo `Clear Body` dell'interfaccia IMessage per .NET quando il corpo di un messaggio di byte è di sola lettura, il corpo diventa scrivibile. Il metodo cancella anche il corpo.

Associa messaggi

Il corpo di un messaggio di associazione contiene una serie di coppie nome - valore, in cui ogni valore ha un tipo di dati associato.

In ogni coppia nome - valore, il nome è una stringa che identifica il valore e il valore è un elemento dei dati dell'applicazione che ha uno dei tipi di dati XMS elencati in Tabella 100 a pagina 675. L'ordine delle coppie nome - valore non è definito. La classe MapMessage contiene i metodi per impostare e richiamare le coppie nome - valore.

Un'applicazione può accedere a una coppia nome - valore in modo casuale specificandone il nome.

Un'applicazione .NET può utilizzare la proprietà MapNames per ottenere un'enumerazione dei nomi nel contenuto del messaggio della mappa.

Quando un'applicazione ottiene il valore di una coppia nome - valore, il valore può essere convertito da XMS in un altro tipo di dati. Ad esempio, per ottenere un numero intero dal corpo di un messaggio di associazione, un'applicazione può richiamare il metodo GetString della classe MapMessage , che restituisce il valore intero come stringa. Le conversioni supportate sono uguali a quelle supportate quando XMS converte un valore di proprietà da un tipo di dati a un altro. Per ulteriori informazioni sulle conversioni supportate, consultare “Conversione implicita di un valore di proprietà da un tipo di dati a un altro” a pagina 649.

Dopo che un'applicazione crea un messaggio di mappa, il corpo del messaggio è leggibile e scrivibile. Il corpo rimane leggibile e scrivibile dopo che l'applicazione ha inviato il messaggio. Quando un'applicazione riceve un messaggio di mappa, il corpo del messaggio è di sola lettura. Se un'applicazione richiama il metodo Clear Body della classe Message quando il corpo di un messaggio della mappa è di sola lettura, il corpo diventa leggibile e scrivibile. Il metodo cancella anche il corpo.

Messaggi oggetto

Il corpo di un messaggio di oggetto contiene un oggettoJava o .NET serializzato.

Un'applicazione XMS può ricevere un messaggio oggetto, modificarne i campi di intestazione e le proprietà e inviarlo ad un'altra destinazione. Un'applicazione può anche copiare il corpo di un messaggio oggetto e utilizzarlo per formare un altro messaggio oggetto. XMS tratta il testo di un messaggio oggetto come un array di byte.

Dopo che un'applicazione crea un messaggio oggetto, il corpo del messaggio è leggibile e scrivibile. Il corpo rimane leggibile e scrivibile dopo che l'applicazione ha inviato il messaggio. Quando un'applicazione riceve un messaggio oggetto, il corpo del messaggio è di sola lettura. Se un'applicazione richiama il metodo `Clear Body` dell'interfaccia IMessage per .NET quando il contenuto di un messaggio oggetto è di sola lettura, il corpo diventa leggibile e scrivibile. Il metodo cancella anche il corpo.

Il processo di flusso dei messaggi

Il corpo di un messaggio di flusso contiene un flusso di valori, in cui ogni valore ha un tipo di dati associato.

Il tipo di dati di un valore è uno dei tipi di dati XMS elencati in Tabella 100 a pagina 675.

Dopo che un'applicazione ha creato un messaggio di flusso, il corpo del messaggio è scrivibile. L'applicazione assembla i dati dell'applicazione nel corpo richiamando i metodi di scrittura appropriati dell'interfaccia IStreamMessage per .NET. Ogni volta che l'applicazione scrive un valore nel flusso di messaggi, il valore e il tipo di dati vengono assemblati immediatamente dopo il precedente valore

scritto dall'applicazione. XMS conserva un cursore interno per ricordare la posizione dell'ultimo valore assemblato.

Quando l'applicazione invia il messaggio, il corpo del messaggio diventa di sola lettura. In questa modalità, l'applicazione può inviare il messaggio più volte.

Quando un'applicazione riceve un messaggio di flusso, il corpo del messaggio è di sola lettura. L'applicazione può utilizzare i metodi di lettura appropriati dell'interfaccia IStreamMessage per .NET per leggere il contenuto del flusso di messaggi. L'applicazione legge i valori in sequenza e XMS mantiene un cursore interno per ricordare la posizione dell'ultimo valore letto.

Quando un'applicazione legge un valore dal flusso di messaggi, il valore può essere convertito da XMS in un altro tipo di dati. Ad esempio, per leggere un numero intero dal flusso di messaggi, un'applicazione può chiamare il metodo ReadString , che restituisce il numero intero come stringa. Le conversioni supportate sono uguali a quelle supportate quando XMS converte un valore di proprietà da un tipo di dati a un altro. Per ulteriori informazioni sulle conversioni supportate, consultare “Conversione implicita di un valore di proprietà da un tipo di dati a un altro” a pagina 649.

Se si verifica un errore mentre un'applicazione sta tentando di leggere un valore dal flusso di messaggi, il cursore non è avanzato. L'applicazione può recuperare dall'errore tentando di leggere il valore come un altro tipo di dati.

Se un'applicazione richiama il metodo Reset dell'interfaccia IStreamMessage per XMS quando il contenuto di un messaggio di flusso è di sola scrittura, il corpo diventa di sola lettura. Il metodo riposiziona il cursore all'inizio del flusso di messaggi.

Se un'applicazione richiama il metodo `Clear Body` dell'interfaccia IMessage per XMS quando il contenuto di un messaggio di flusso è di sola lettura, il corpo diventa di sola scrittura. Il metodo cancella anche il corpo.

Messaggi di testo

Il corpo di un messaggio di testo contiene una stringa.

Dopo che un'applicazione crea un messaggio di testo, il corpo del messaggio è leggibile e scrivibile. Il corpo rimane leggibile e scrivibile dopo che l'applicazione ha inviato il messaggio. Quando un'applicazione riceve un messaggio di testo, il corpo del messaggio è di sola lettura. Se un'applicazione richiama il metodo Clear Body dell'interfaccia IMessage per .NET quando il contenuto di un messaggio di testo è di sola lettura, il corpo diventa leggibile e scrivibile. Il metodo cancella anche il corpo.

Tipi di dati per elementi di dati dell'applicazione

Per essere certi che un'applicazione XMS possa scambiare messaggi con un'applicazione IBM MQ classes for JMS , entrambe le applicazioni devono essere in grado di interpretare i dati dell'applicazione nel corpo di un messaggio nello stesso modo.

Per questo motivo, ogni elemento dei dati dell'applicazione scritto nel corpo di un messaggio da parte di un'applicazione XMS deve avere uno dei tipi di dati elencati in Tabella 100 a pagina 675. Per ciascun tipo di dati, la tabella mostra il tipo di dati Java compatibile. XMS fornisce i metodi per scrivere elementi di dati dell'applicazione solo con questi tipi di dati.

Tabella 100. Tipi di dati XMS compatibili con i tipi di dati Java

XMS Tipo dati	**Rappresenta**	**Tipo di dati Java compatibile**
System.Boolean	Il valore booleano true o false	booleano
System.Char16	Carattere a doppio byte	char
System.SByte	Numero intero a 8 bit con segno	byte
System.Int16	Numero intero a 16 bit con segno	breve
System.Int32	Numero intero a 32 bit con segno	int

Tabella 100. Tipi di dati XMS compatibili con i tipi di dati Java (Continua)		
XMS Tipo dati	**Rappresenta**	**Tipo di dati Java compatibile**
System.Int64	Numero intero a 64 bit con segno	lungo
System.Float	Numero a virgola mobile con segno	mobile
System.Double	Numero a virgola mobile a doppia precisione con segno	doppio
System.String	Stringa di caratteri	Stringa

Per informazioni sulla dimensione, il valore massimo e il valore minimo di ognuno di questi tipi di dati, consultare "XMS Tipi primitivi" a pagina 648.

Selettori di messaggi

Un'applicazione XMS utilizza selettori di messaggi per selezionare i messaggi che desidera ricevere.

Quando un'applicazione crea un consumatore di messaggi, può associare un'espressione del selettore di messaggi al consumatore. L'espressione del selettore messaggi specifica i criteri di selezione.

Quando un'applicazione si connette al gestore code IBM WebSphere MQ 7.0 , la selezione del messaggio viene effettuata sul lato gestore code. XMS non effettua alcuna selezione e consegna semplicemente il messaggio ricevuto dal gestore code, fornendo così prestazioni migliori.

Un'applicazione può creare più di un consumatore di messaggi, ciascuno con la propria espressione del selettore di messaggi. Se un messaggio in entrata soddisfa i criteri di selezione di più di un consumer di messaggi, XMS consegna il messaggio a ognuno di questi consumer.

Un'espressione del selettore messaggi può fare riferimento alle seguenti proprietà di un messaggio:

- Proprietà definite da JMS
- IBM
- Proprietà definite dall'applicazione

Può anche fare riferimento ai seguenti campi di intestazione del messaggio:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

Un'espressione del selettore messaggi, tuttavia, non può fare riferimento ai dati nel corpo di un messaggio.

Di seguito è riportato un esempio di espressione del selettore di messaggi:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS consegna un messaggio a un consumatore di messaggi con questa espressione del selettore di messaggi solo se il messaggio ha una priorità maggiore di 3; una proprietà definita dall'applicazione, produttore, con un valore di `Jaguar;` e un'altra applicazione definita - proprietà, modello, con un valore di `xj6` o `xj12.`

Le regole di sintassi per la formazione di un'espressione del selettore di messaggi in XMS sono le stesse di quelle in IBM MQ classes for JMS. Per informazioni su come creare un'espressione del selettore messaggi, consultare la documentazione del prodotto IBM MQ . Si noti che, in un'espressione del selettore messaggi, i nomi delle proprietà definite da JMSdevono essere i nomi JMS e i nomi delle

proprietà definite da IBMdevono essere i nomi IBM MQ classes for JMS . Non è possibile utilizzare i nomi XMS in un'espressione del selettore messaggi.

Mappatura dei messaggi XMS sui messaggi IBM MQ

I campi di intestazione JMS e le proprietà di un messaggio XMS sono associati ai campi nelle strutture di intestazione di un messaggio IBM MQ .

Quando un'applicazione XMS è connessa a un gestore code IBM MQ , i messaggi inviati al gestore code vengono associati ai messaggi IBM MQ nello stesso modo in cui i messaggi IBM MQ classes for JMS vengono associati ai messaggi IBM MQ in circostanze simili.

Se la proprietà XMSC_WMQ_TARGET_CLIENT di un oggetto di destinazione è impostata su XMSC_WMQ_TARGET_DEST_JMS, i campi di intestazione JMS e le proprietà di un messaggio inviato alla destinazione vengono associati ai campi nelle strutture di intestazioni MQMD e MQRFH2 del messaggio IBM MQ . Se si imposta la proprietà XMSC_WMQ_TARGET_CLIENT in questo modo, si presuppone che l'applicazione che riceve il messaggio possa gestire un'intestazione MQRFH2 . L'applicazione ricevente potrebbe quindi essere un'altra applicazione XMS , un'applicazione IBM MQ classes for JMS o un'applicazione IBM MQ nativa progettata per gestire un'intestazione di MQRFH2 .

Se invece la proprietà XMSC_WMQ_TARGET_CLIENT di un oggetto di destinazione è impostata su XMSC_WMQ_TARGET_DEST_MQ, i campi di intestazione JMS e le proprietà di un messaggio inviato alla destinazione vengono associati ai campi nella struttura dell'intestazione MQMD del messaggio IBM MQ . Il messaggio non contiene un'intestazione MQRFH2 e tutti i campi di intestazione JMS e le proprietà che non possono essere associati ai campi nella struttura dell'intestazione MQMD vengono ignorati. L'applicazione che riceve il messaggio può quindi essere una IBM MQ nativa che non è progettata per gestire un'intestazione MQRFH2 .

I messaggi IBM MQ ricevuti da un gestore code vengono associati ai messaggi XMS nello stesso modo in cui i messaggi IBM MQ vengono associati ai messaggi IBM MQ classes for JMS in circostanze simili.

Se un messaggio IBM MQ in entrata dispone di un'intestazione MQRFH2 , il messaggio XMS risultante ha un corpo il cui tipo è determinato dal valore della proprietà **Msd** contenuto nella cartella `mcd` dell'intestazione MQRFH2 . Se la proprietà **Msd** non è presente nell'intestazione MQRFH2 o se il messaggio IBM MQ non ha un'intestazione MQRFH2 , il messaggio XMS risultante ha un corpo il cui tipo è determinato dal valore del campo *`Format`* nell'intestazione MQMD. Se il campo *`Format`* è impostato su MQFMT_STRING, il messaggio XMS è un messaggio di testo. Altrimenti, il messaggio XMS è un messaggio di byte. Se il messaggio IBM MQ non ha un'intestazione MQRFH2 , vengono impostati solo i campi di intestazione JMS e le proprietà che possono derivare dai campi nell'intestazione MQMD.

Per ulteriori informazioni sull'associazione dei messaggi IBM MQ classes for JMS ai messaggi IBM MQ , consultare “Mappatura dei messaggi JMS sui messaggi IBM MQ” a pagina 150.

Lettura e scrittura del descrittore del messaggio da una applicazione IBM MQ Message Service Client (XMS) for .NET

È possibile accedere a tutti i campi del descrittore del messaggio (MQMD) di un messaggio IBM MQ ad eccezione di StrucId e Version; BackoutCount può essere letto ma non scritto.

Gli attributi del messaggio forniti da IBM MQ Message Service Client (XMS) for .NET facilitano le applicazioni XMS per impostare i campi MQMD e anche per guidare applicazioni IBM WebSphere MQ .

Alcune limitazioni si applicano quando si utilizza la messaggistica di pubblicazione / sottoscrizione. Ad esempio, i campi MQMD come MsgID e CorrelId, se impostati, vengono ignorati.

La funzione non è disponibile anche quando la proprietà **PROVIDERVERSION** è impostata su 6.

Accesso ai dati del messaggio IBM MQ da una applicazione IBM MQ Message Service Client (XMS) for .NET

È possibile accedere ai dati completi del messaggio IBM MQ , inclusa l'intestazione MQRFH2 (se presente) e qualsiasi altra intestazione IBM MQ (se presente) all'interno di un'applicazione IBM MQ Message Service Client (XMS) for .NET come corpo di un JMSBytesMessage.

La funzione descritta in questo argomento è disponibile solo quando ci si connette a un gestore code IBM WebSphere MQ 7.0 o successivo e il provider di messaggistica IBM MQ è in modalità normale.

Le proprietà dell'oggetto di destinazione determinano in che modo l'applicazione XMS accede all'intero messaggio IBM MQ (inclusa l'intestazione MQRFH2 , se presente) come corpo di un JMSBytesMessage.

ALW Sviluppo di applicazioni client AMQP

Il supporto IBM MQ per le API AMQP, permette a un'amministratore IBM MQ di creare un canale AMQP. Quando viene avviato, questo canale definisce un numero di porta che accetta le connessioni dalle applicazioni client AMQP.

È possibile installare un canale AMQP su sistemi AIX, Linux, and Windows ; non è disponibile su IBM i o z/OS.

Un'applicazione client AMQP 1.0 può connettersi al gestore code con un canale AMQP.

Sviluppo di applicazioni utilizzando la libreria Apache Qpid JMS

Introduzione

La libreria Apache Qpid JMS usa il protocollo AMQP 1.0 per fornire un'implementazione della specifica JMS 2.

Apache Qpid JMS usa alcuni aspetti del protocollo AMQP 1.0 in modo diverso dalle API di messaggistica MQ Light . IBM MQ 9.2 aggiunge il supporto ai canali IBM MQ AMQP, in modo che le applicazioni Apache Qpid JMS siano in grado di connettersi a IBM MQ ed eseguire la messaggistica di pubblicazione / sottoscrizione, incluso l'utilizzo di sottoscrizioni condivise.

V 9.3.0 IBM MQ 9.3 aggiunge ulteriore supporto ai IBM MQ canali AMQP, in modo che le applicazioni Apache Qpid JMS siano in grado di connettersi a IBM MQ ed eseguire la messaggistica point - to - point. Per ulteriori informazioni, consulta “Supporto point - to - point sui canali AMQP” a pagina 683.

V 9.3.0 IBM MQ 9.3.0 aggiunge ulteriore supporto di ricerca coda per IBM MQ canali AMQP, in modo che le applicazioni JMS Apache Qpid possano connettersi a IBM MQ ed eseguire l'esplorazione dei messaggi da una coda. Per ulteriori informazioni, consulta “Supporto point - to - point sui canali AMQP” a pagina 683.

V 9.3.0 IBM MQ 9.3.0 aggiunge due ulteriori attributi di canale per i canali AMQP, TMPMODEL e TMPQPRFX. Questi attributi sono per la coda modello e il prefisso della coda temporanea da utilizzare durante la creazione di una coda temporanea.

Intercomunicazione con altre applicazioni IBM MQ

È possibile inviare messaggi tra le applicazioni Apache Qpid JMS e altre applicazioni IBM MQ . Ad esempio, un'applicazione Qpid Apache può pubblicare messaggi su un argomento e le applicazioni MQ Light possono riceverli creando una sottoscrizione.

Un'applicazione Apache Qpid JMS può anche pubblicare messaggi utilizzati dalle applicazioni IBM MQ tradizionali, ad esempio utilizzando la chiamata API MQSUB per sottoscrivere lo stesso argomento.

Allo stesso modo, le applicazioni JMS Qpid Apache possono sottoscrivere gli argomenti IBM MQ su cui le applicazioni IBM MQ tradizionali pubblicano i messaggi.

È anche possibile che un'applicazione JMS Qpid Apache condivida una sottoscrizione con un'applicazione MQ Light , purché entrambi i client specifichino lo stesso nome di condivisione e lo stesso modello di argomento.

Per fare ciò, l'applicazione Apache Qpid JMS non deve connettersi con un ID client. Ciò garantisce che il nome sottoscrizione IBM MQ utilizzato da entrambe le applicazioni sia lo stesso.

Attenzione: Non è possibile per un'applicazione Apache Qpid JMS condividere una sottoscrizione con un'applicazione IBM MQ JMS .

Limitazioni diApache Qpid JMS

Sono supportate le seguenti funzionalità JMS :

- Riconoscimento client, riconoscimento automatico e modalità di riconoscimento dups ok (DUPS_OK_ACKNOWLEDGE)
  - Connessione con o senza credenziali
  - Creazione di un consumer su una destinazione argomenti
  - Creazione di un consumer durevole su una destinazione argomenti
  - Creazione di un consumer condiviso su una destinazione argomenti
  - Creazione di un consumer durevole condiviso su una destinazione argomenti
  - Modalità di riconoscimento client e di riconoscimento automatico
  - Riconoscimento messaggio e riconoscimento sessione
  - Annullamento della sottoscrizione da una sottoscrizione durevole
  - V 9.3.0 Creazione di una coda temporanea
  - V 9.3.0 Creazione di un consumer su una coda o una destinazione della coda temporanea
  - V 9.3.0 JMS MessageListeners
  - V 9.3.0 Consumer JMS per ricevere il corpo; metodo JMS 2.0 denominato `Consumer.receiveBody()`
  - V 9.3.0 Sono supportati i seguenti tipi di messaggi JMS:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - V 9.3.0 Ricerca dei messaggi da una coda

Le seguenti funzionalità JMS non sono supportate dai client AMQP:

- L'utilizzo di sessioni transazionali e JMSContexts transazionali
  - L'utilizzo dei selettori di messaggi
  - L'utilizzo dell'attributo **nolocal**
  - L'utilizzo delle sessioni transazionali
  - L'uso del ritardo di consegna
  - In IBM MQ 9.3.0, sfogliare i messaggi da una coda.
  - Creazione di più sottoscrizioni o consumer durevoli con lo stesso ID client e argomento
  - V 9.3.0 Argomenti temporanei JMS
  - I filtri AMQP non sono supportati.

V 9.3.3 Da IBM MQ 9.3.3 la seguente nota non è più valida per gli utenti Continuous Delivery .

Attenzione: V 9.3.0 Riconoscimento client: se devono essere utilizzati trasferimenti di messaggi AMQP non stabiliti, vale a dire quando è richiesto un riconoscimento client dei messaggi, i client devono eseguire l'impostazione in modo tempestivo inviando i riconoscimenti dei messaggi in modo tempestivo quando si utilizza la modalità di riconoscimento client oppure considerare l'impostazione della proprietà del gestore code MARKINT (**MsgMarkBrowseInterval**) su un valore superiore.

Il valore predefinito per **MsgMarkBrowseInterval** è cinque secondi. Se un'applicazione non rientra in questo valore predefinito, potrebbero essere visualizzati messaggi duplicati. Per evitare la duplicazione dei messaggi, è necessario aumentare il valore di **MsgMarkBrowseInterval** di conseguenza, impostandolo idealmente su NOLIMIT, per rappresentare un intervallo di tempo illimitato. Se un'applicazione si interrompe o si disconnette prima che venga risolto un messaggio, i messaggi vengono resi disponibili per un'altra applicazione.

Per ulteriori informazioni, consultare **MsgMarkBrowseInterval** . Poiché si tratta di una proprietà del gestore code, il valore impostato verrà applicato a tutte le applicazioni connesse a tale gestore code.

Su AMQP **MsgMarkBrowseInterval** è valido solo per code, non per sottoscrizioni.

Download dei client AMQP di esempio

IBM MQ non fornisce i client AMQP, ma è possibile scaricare i client MQ Light o i client AMQP open-source basati sulle librerie Qpid Apache . Per ulteriori informazioni, vedi IBM MQ Light e Apache Qpid.

È anche possibile scaricare altri client AMQP open-source basati su Apache librerie Qpid. Per ulteriori informazioni, consultare https://qpid.apache.org/index.html.

Attenzione: Il supporto IBM non è in grado di fornire il supporto per la configurazione o i defect per questi pacchetti client e qualsiasi domanda di utilizzo o report di defect del codice deve essere indirizzata ai rispettivi progetti.

Distribuzione di client AMQP a IBM MQ

Quando un'applicazione è pronta per la distribuzione, richiede tutte le funzioni di monitoraggio, affidabilità e sicurezza di altre applicazioni enterprise. Può anche scambiare dati con altre applicazioni enterprise.

Una volta distribuito un client AMQP, è possibile scambiare messaggi con applicazioni IBM MQ . Ad esempio, se si utilizza il client AMQP per inviare un messaggio di stringa JavaScript , l'applicazione IBM MQ riceve un messaggio MQ , dove il campo del formato di MQMD è impostato su MQSTR.

Gestione del canale AMQP

Il canale AMQP può essere gestito come altri canali MQ . È possibile utilizzare comandi MQSC, messaggi di comandi PCF o IBM MQ Explorer per definire, avviare, arrestare e gestire i canali. In Creazione e utilizzo di canali AMQP, vengono forniti dei comandi di esempio per definire e avviare la connessione dei client a un gestore code.

Quando viene avviato un canale AMQP, è possibile verificarlo collegando un client AMQP 1.0 . Ad esempio, MQ Light, Apache Qpid Proton o Apache Qpid JMS.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW MQ Light, Apache Qpid JMSe AMQP (Advanced Message Queuing Protocol)

Il client MQ Light , i client Apache Qpid come Apache Proton e le API Apache Qpid JMS si basano sul protocollo di collegamento OASIS Standard AMQP 1.0 . AMQP specifica il modo in cui i messaggi vengono inviati tra mittenti e destinatari. Un'applicazione agisce come un mittente quando l'applicazione invia un messaggio al broker dei messaggi, ad esempio IBM MQ. IBM MQ agisce come mittente quando invia un messaggio a una applicazione AMQP.

Alcuni dei vantaggi dell'AMQP sono i seguenti:

- Un protocollo standardizzato aperto
- Compatibilità con altri client AMQP 1.0 open source

- Molte implementazioni client open source disponibili

Sebbene qualsiasi client AMQP 1.0 possa connettersi a un canale AMQP, alcune funzioni AMQP non sono supportate, ad esempio transazioni o sessioni multiple.

Per ulteriori informazioni, consultare il sito Web diAMQP.org e il PDF OASIS Standard AMQP 1.0.

Le API MQ Light e Apache Qpid JMS dispongono delle seguenti funzioni di messaggistica:

- Consegna del messaggio al massimo una volta
- Consegna del messaggio almeno una volta
- Indirizzamento destinazione stringa argomento
- Durata messaggio e destinazione
- Destinazioni condivise per consentire a più sottoscrittori di condividere il workload
- Acquisizione del client per una facile risoluzione dei client in sospeso
- Lettura anticipata dei messaggi configurabile
- Riconoscimento configurabile dei messaggi

Per la documentazione completa dell'API Apache Qpid JMS , consultare Qpid JMS.

Attività correlate
Creazione e utilizzo di canali AMQP
Protezione dei client AMQP

ALW supporto AMQP 1.0

I canali AMQP forniscono un livello di supporto per le applicazioni AMQP 1.0-compliant .

I canali AMQP supportano un sottoinsieme del protocollo AMQP 1.0 . È possibile connettere i client compatibili AMQP 1.0 a un canale AMQP IBM MQ . Per utilizzare tutte le funzioni di messaggistica supportate dai canali AMQP, è necessario impostare correttamente il valore di determinati campi AMQP 1.0 .

Queste informazioni delineano il modo in cui i campi AMQP devono essere formattati ed elencare le funzioni della specifica AMQP 1.0 che non sono supportate dai canali AMQP.

Le seguenti funzioni della specifica AMQP 1.0 non sono supportate o sono limitate nel loro utilizzo:

Frame ATTACH

V 9.3.0

I canali AMQP si aspettano che le funzionalità nel frame ATTACH contengano una delle seguenti:

```
topic
temporary queue
queue
shared
```

Le funzioni implicano il tipo di oggetto e, nel caso di più funzioni, l'ordine di priorità della selezione della funzione è argomento, coda temporanea, coda.

Se una funzionalità non contiene un valore previsto, la funzionalità predefinita è argomento. Tutte le altre funzioni vengono ignorate.

Nota: Alcuni client AMQP non impostano queste capacità e otterranno il comportamento predefinito IBM MQ di pubblicazione / sottoscrizione. Ad esempio, il connettore Quarkus Reactive Messaging AMQP 1.0 imposta solo le funzioni dalla versione 2.8.0CR1 in poi.

V 9.3.0 I canali AMQP si aspettano che `distribution-Mode` nel frame ATTACH contenga uno dei seguenti elementi, per un'origine o una destinazione:

- `spostare`
- `copia`

dove `move` implica un richiamo distruttivo e `copy` implica browser.

Nota: Se `distribution-Mode` non è impostato o è impostato su un valore diverso da `copy`, si presuppone `move` .

Nomi link

I canali AMQP si aspettano che il nome di un link AMQP segua uno dei cinque formati:

- Un argomento semplice (per la pubblicazione e la sottoscrizione)
  - Pubblicazione di messaggi: una stringa di argomenti semplice (ad esempio, un nome link `"/sports/football"`) fa sì che un messaggio venga pubblicato sull'argomento `/sports/football` .
  - Sottoscrizione a un argomento per ricevere messaggi: una stringa di argomenti semplice (ad esempio, un nome link di `"/sports/football"` determina la definizione di una sottoscrizione nell'argomento `/sports/football` .
- Un argomento verbose privato (per la sottoscrizione)
  - Una stringa di argomenti dettagliata che descrive una sottoscrizione privata nel formato: `"private:`*`topic string`*`"` (ad esempio: `"private:/sports/football"`). Il comportamento è identico a una stringa di argomenti semplice. La dichiarazione `private` distingue una sottoscrizione specifica per un particolare client AMQP da una sottoscrizione condivisa tra client.
- Un argomento verbose condiviso (per la sottoscrizione)
  - Una stringa di argomento dettagliata che descrive una sottoscrizione condivisa nel formato: `"share:`*`share name`*`:`*`topic string`*`"` (ad esempio: `"share:bbc:/sports/football"`).
- V 9.3.0 Una coda (per la messaggistica punto a punto per il producer e il consumer)
  - Produttore per inviare messaggi; una stringa del nome coda fa sì che un produttore invii un messaggio su una coda.
  - Utente che riceve i messaggi; una stringa del nome coda fa sì che un utente riceva i messaggi da una coda.
- V 9.3.0 Vuoto (per la messaggistica punto a punto su una coda temporanea)
  - Produttore per inviare messaggi su una coda temporanea; il vuoto fa sì che un produttore invii un messaggio su una coda temporanea.
  - Il consumer riceve i messaggi su una coda temporanea; il vuoto fa sì che un consumer riceva i messaggi da una coda temporanea.

Per ulteriori informazioni sul modo in cui i messaggi AMQP vengono associati a e dai messaggi IBM MQ , consultare “Mappatura dei campi AMQP sui campi IBM MQ (messaggi in entrata)” a pagina 687.

Lunghezze massime per stringhe argomento, nomi condivisione e ID client

La stringa di argomenti, il nome condivisione e l'ID client devono essere contenuti entro 10237 byte. Inoltre, la lunghezza massima di un ID client è 256 caratteri.

Queste lunghezze massime indicano che è possibile avere una delle seguenti:

- una stringa di argomenti molto lunga, purché il nome della condivisione sia breve
- un nome di condivisione lungo, ma una stringa di argomento breve

ID contenitore

I canali AMQP prevedono che il container - id di un AMQP Open contenga un ID client AMQP univoco. La lunghezza massima di un ID client AMQP è 256 caratteri e l'ID può contenere caratteri alfanumerici, il segno percentuale (%), la barra (/), il punto (.) e il carattere di sottolineatura (_).

Sessioni

I canali AMQP supportano solo una singola sessione AMQP. Un client AMQP che tenta di creare più di una sessione AMQP riceve un messaggio di errore e viene disconnesso dal canale.

Transazioni

I canali AMQP non supportano le transazioni AMQP. Un frame di collegamento AMQP che tenta di coordinare una nuova transazione o un frame di trasferimento AMQP che tenta di dichiarare una nuova transazione viene respinto con un messaggio di errore.

Stato consegna

I canali AMQP supportano solo uno stato di consegna per i frame di disposizione Accettato, Rilasciato o Modificato. Si noti che, quando viene utilizzato uno stato Modificato, i canali AMQP non supportano l'opzione undeliverable - here.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

V 9.3.0 ALW Supporto point - to - point sui canali AMQP

Il canale AMQP IBM MQ fornisce supporto per l'invio di messaggi alle code e la ricezione di messaggi dalle code.

I client AMQP come una libreria Apache Qpid™ JMS richiedono una funzionalità `queue` o `temporary-queue` quando inviano il frame di collegamento AMQP. Le funzionalità consentono al canale AMQP di identificare l'oggetto come coda, coda temporanea o argomento. In assenza di una capacità di coda o di coda temporanea, o anche di una qualsiasi delle capacità, si presume che la richiesta sia per un argomento.

I canali IBM MQ AMQP forniscono il supporto per il tipo di coda per:

Coda di ricezione e invio
I messaggi possono essere inviati a una coda e utilizzati da una coda. Per l'utilizzo dei messaggi, sono supportate le modalità sincrone e asincrone.

V 9.3.0 Messaggio di ricerca coda
Oltre a inserire i messaggi in una coda e ottenere i messaggi da una coda, i messaggi possono anche essere sfogliati da una coda.

Supporto coda temporanea
I messaggi possono essere inviati a una coda temporanea e utilizzati da una coda temporanea. Si noti che l'eliminazione della coda temporanea è supportata, se lo stesso oggetto della coda temporanea utilizzato per creare la coda temporanea viene utilizzato anche per eliminare la coda temporanea.

Il SISTEMA SYSTEM.DEFAULT.MODEL.QUEUE viene utilizzato durante la creazione di una coda temporanea e il prefisso per la coda temporanea sarà `AMQP.*`.

Il SISTEMA SYSTEM.DEFAULT.MODEL.QUEUE è per impostazione predefinita una coda dinamica temporanea, ma è possibile utilizzare la proprietà **`Definition type`** sul SISTEMA SYSTEM.DEFAULT.MODEL.QUEUE per modificare la coda in una coda dinamica permanente.

coda dinamica permanente
Una coda dinamica permanente viene eliminata quando un client AMQP, come una libreria Apache Qpid JMS , invia una richiesta con un frame `detach` con l'attributo **`closed`** impostato su *true*.

Importante:

Comportamento JMS Qpid:
È necessario richiamare un comando API Qpid JMS , ad esempio, il metodo `javax.jms.TemporaryQueue.delete()` per eliminare la coda dopo l'utilizzo e questo processo cancella anche i messaggi presenti nella coda.

Se non si immette un comando di questo tipo, la coda rimane con i messaggi ancora presenti, quando la connessione viene chiusa.

-

coda dinamica temporanea
Una coda dinamica temporanea viene eliminata quando il client AMQP chiude la connessione.

Importante:

Comportamento JMS Qpid:
Se si richiama un comando API Qpid JMS , ad esempio, il metodo `javax.jms.TemporaryQueue.delete()` , si chiude la connessione JMS o la connessione si interrompe, la coda viene eliminata e tutti i messaggi vengono persi.

La chiusura di una sessione JMS non comporta l'eliminazione della coda temporanea, anche se la coda temporanea potrebbe essere stata creata utilizzando il metodo `javax.jms.Session.createTemporaryQueue()` .

-

Attività correlate
Creazione e utilizzo di canali AMQP
Protezione dei client AMQP

ALW Associazione di campi di messaggi AMQP e IBM MQ

I messaggi AMQP sono composti da un'intestazione, annotazioni di consegna, annotazioni di messaggi, proprietà, proprietà dell'applicazione, corpo e piè di pagina.

I messaggi AMQP sono composti dalle parti seguenti:

Intestazione
L'intestazione facoltativa contiene cinque attributi fissi del messaggio:

- **durevole** - specifica i requisiti di durata
- **priority** - priorità del messaggio relativo
- **ttl** - TTL (time to live) in millesimi di secondo
- **primo acquirente** - se questo è true, il messaggio non è stato acquisito da nessun altro link
- **delivery - count** - il numero di tentativi di recapito non riusciti precedenti.

Consegna - annotazioni
Facoltativo. Specifica gli attributi di intestazione non standard del messaggio per destinatari diversi. Le annotazioni di consegna trasmettono le informazioni dal peer di invio al peer di ricezione.

Annotazioni messaggio
Facoltativo. Specifica gli attributi di intestazione non standard del messaggio per destinatari diversi. La sezione delle annotazioni del messaggio viene utilizzata per le proprietà del messaggio che sono indirizzate all'infrastruttura e devono essere trasmesse in ogni fase di consegna.

Proprietà
Facoltativo. Questa parte è equivalente al descrittore del messaggio MQ . Contiene i seguenti campi fissi:

- **id - messaggio** - identificativo del messaggio dell'applicazione
- **user - id** - ID dell'utente di creazione
- **a** - indirizzo del nodo a cui è destinato il messaggio
- **subject** - l'oggetto del messaggio
- **reply - to** - il nodo a cui inviare le risposte
- **ID correlazione** - identificativo di correlazione dell'applicazione
- **content - type** - Tipo di contenuto MIME

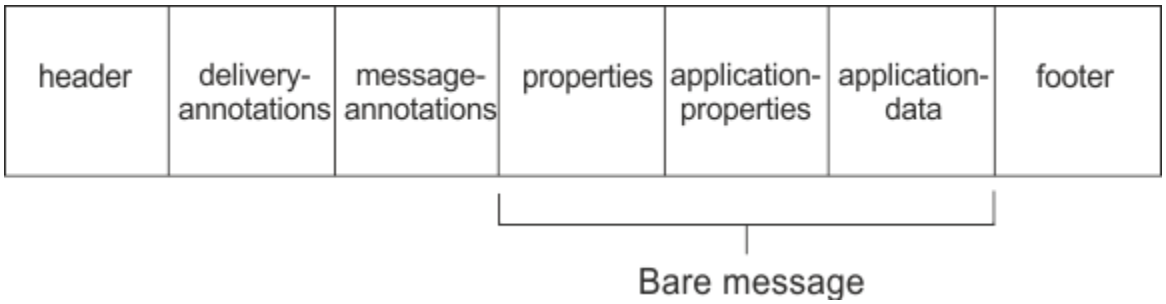
- **content - encoding** - Tipo di contenuto MIME. Utilizzato come modificatore del tipo di contenuto.
- **absolute - expire - time** - l'ora in cui questo messaggio viene considerato scaduto
- **creazione - ora** - l'ora in cui è stato creato questo messaggio
- **group - id** - il gruppo a cui appartiene questo messaggio
- **group - sequence** - il numero di sequenza di questo messaggio all'interno del gruppo
- **reply - to - group - id** - il gruppo a cui appartiene il messaggio di risposta

Applicazioni - proprietà
Equivalente alle proprietà del messaggio MQ .

Corpo
Equivalente al payload dell'utente MQ .

Piè di pagina
Facoltativo. Il piè di pagina viene utilizzato per i dettagli relativi al messaggio o alla consegna che possono essere calcolati o valutati solo dopo che l'intero messaggio è stato creato o visualizzato (ad esempio, hash del messaggio, HMAC, firme e dettagli di crittografia).

Il formato del messaggio AMQP è illustrato nella seguente figura:



Le proprietà, le proprietà dell'applicazione e la parte dati dell'applicazione sono note come "messaggio semplice". Questo è il messaggio inviato dal mittente ed è immutabile. Il destinatario visualizza l'intero messaggio, inclusi l'intestazione, il piè di pagina, le annotazioni di consegna e le annotazioni di messaggio.

Per una descrizione completa del formato del messaggio AMQP 1.0 , consultare lo standard OASIS all'indirizzo https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf

Attività correlate
Creazione e utilizzo di canali AMQP
Protezione dei client AMQP

ALW Mappatura dei campi IBM MQ sui campi AMQP (messaggi in uscita)

Quando un messaggio IBM MQ viene pubblicato e IBM MQ lo invia a un'utente AMQP, propagherà alcuni attributi del messaggio IBM MQ in attributi di messaggio AMQP equivalenti.

intestazione

Un'intestazione è inclusa solo se uno dei cinque campi nell'intestazione contiene un valore non predefinito. Nell'intestazione vengono inclusi solo i campi con un valore non predefinito. I cinque campi di intestazione sono inizialmente derivati dalla proprietà `mq_amqp.Hdr` equivalente, se impostata, e quindi modificati come mostrato nella seguente tabella:

Tabella 101. Associazioni dei campi di intestazione

Campo	Valore predefinito	Valore
durevole	No	True se `MQMD.Persistence` è impostato su `MQPER_PERSISTENT`, altrimenti false.

Tabella 101. Associazioni dei campi di intestazione (Continua)		
Campo	**Valore predefinito**	**Valore**
priorità	4	Da `mq_amqp.Hdr.Pri`, se impostato, o altrimenti da `MQMD.Priority`, se impostato. Se nessuno dei due è impostato, impostare su 4.
ttl	n/a	`MQMD.Expiry` in millisecondi. Se il valore di `MQMD.Expiry` è MQEI_UNLIMITED, impostare il valore massimo per il campo ttl AMQP
primo acquirente	No	Da `mq_amqp.Hdr.Fac`, se impostato o false in caso contrario.
consegna - conteggio	0	Da `mq_amqp.Hdr.Dct`, se impostato, o 0 in caso contrario.

consegna - annotazione

Impostare come necessario dal canale AMQP.

annotazione messaggio

Non incluso.

Proprietà

Le **proprietà** non verranno modificate dalle proprietà `mq_amqp.Prp` equivalenti, se impostate. Se il messaggio non era in origine un messaggio AMQP (ovvero, il tipo PutApplnon è MQAT_AMQP), viene generata una sezione delle proprietà come descritto nella seguente tabella:

Tabella 102. Associazioni dei campi delle proprietà	
Nome	**Valore**
id messaggio	`MQMD.MsgId` è impostato come binario.
id utente	Il formato UTF-8 di `MQMD.UserIdentifier` è impostato come binario in ordine byte di rete.
a	La coda da cui è stato ricevuto il messaggio o, per una pubblicazione, la stringa di argomenti.
soggetto	Non impostato.
a risposta	Il `MQMD.ReplyToQ` se non è vuoto, altrimenti non è impostato.
id - correlazione	`MQMD.CorrelId` è impostato come binario se non è vuoto, altrimenti non è impostato.
tipo - contenuto	Non impostato.
codifica contenuto	Non impostato.
ora di scadenza assoluta	Non impostato.
creazione - ora	I campi `MQMD.PutDate` e `MQMD.PutTime` vengono utilizzati per generare un timestamp.
id gruppo	Non impostato.
sequenza - gruppo	Non impostato.
id - risposta - a - gruppo	Non impostato.

applicazione - proprietà

Tutte le proprietà IBM MQ nel gruppo "usr" vengono aggiunte come **application - properties**.

corpo

Il canale AMQP esegue un get con convert, per convertire il payload IBM MQ in UTF-8.

Se il payload IBM MQ non contiene un messaggio AMQP, il payload IBM MQ viene impostato nel corpo come una singola sezione di dati stringa per il formato MQFMT_STRING (la conversione fornita in UTF-8 ha avuto esito positivo) oppure come una singola sezione di dati binari.

Se viene incluso un messaggio in formato AMQP, questo viene impostato come corpo. Qualsiasi intestazione IBM MQ (senza includere le proprietà dei messaggi restituite in un handle del messaggio) che precede il messaggio AMQP viene anteposta come valore binario se il corpo è una sequenza AMQP. Altrimenti, le intestazioni IBM MQ vengono eliminate.

pié di pagina

Nessun piè di pagina incluso.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

Riferimenti correlati

MQMD - Descrittore messaggi

ALW Mappatura dei campi AMQP sui campi IBM MQ (messaggi in entrata)

Quando il canale AMQP riceve un messaggio e lo inserisce in IBM MQ, trasmette alcuni degli attributi del messaggio AMQP in attributi del messaggio IBM MQ equivalenti.

Le seguenti limitazioni si applicano quando si associa un messaggio AMQP in entrata:

- Se il campo `message-id` o `correlation-id` nella parte delle proprietà è un uuid o un ulong, il messaggio viene rifiutato.
- Qualsiasi `message-annotations` causa il rifiuto del messaggio.
- Le sezioni `delivery-annotations` e `footer` sono consentite, ma non vengono trasmesse al messaggio IBM MQ .

Le seguenti sezioni secondarie mostrano l'espressione IBM MQ di un messaggio AMQP.

Descrittore messaggio

Tabella 103. Descrittore del messaggio per il messaggio AMQP

Campo	Valore
StrucId	ID_STRUC_MQMD
Versione	MQMD_VERSION_1
Prospetto	MQRO_NONE
MsgType	MQMT_DATAGRAM
Scadenza	Valore preso dal campo `ttl` nell'intestazione del messaggio AMQP
Feedback	MQFB_NONE
Codifica	MQEN_NORMAL

Tabella 103. Descrittore del messaggio per il messaggio AMQP (Continua)	
Campo	**Valore**
CodedCharSetId	1208 (UTF-8)
Formato	Vedere Payload
Priorità	Valore preso dal campo `priority` nell'intestazione del messaggio AMQP. Se impostato, è limitato a un massimo di 9. Se non impostato, assume il valore predefinito di 4.
Persistenza	Se il campo `durable` nell'intestazione del messaggio AMQP è impostato su true, impostare su MQPER_PERSISTENT. Altrimenti, impostare su MQPER_NOT_PERSISTENT.
MagId	Il gestore code assegna un MsgIdunivoco a 24 byte.
CorrelId	Valore preso dal campo `correlation-id` nelle proprietà AMQP, se impostato. Impostare su un valore binario a 24 byte. Altrimenti, impostare su MQCI_NONE/.
BackoutCount	0
ReplyToQ	V 9.3.0 Valore preso dal campo `reply-to` nelle proprietà AMQP, se impostato. Altrimenti, impostare su "".
ReplyToQMgr	""
V 9.3.0 Prospetto	Valore derivato da qualsiasi proprietà del report JMS IBM impostata nelle proprietà dell'applicazione AMQP.
UserIdentifier	Impostare sull'identificativo dell'utente autenticato che si è connesso al canale AMQP
AccountingToken	MQACT_NONE
ApplIdentityData	Stringa esadecimale. Impostare gli ultimi 8 byte dell'identificativo di connessione MQ del canale AMQP.
PutApplType	MQAT_AMQP
PutApplName	
PutDate	Valore preso dal campo `creation-time` delle proprietà AMQP, se impostato. Altrimenti, impostare la data corrente.
PutTime	Valore preso dal campo `creation-time` delle proprietà AMQP, se impostato. Altrimenti, impostare l'ora corrente.
ApplOriginData	""

Proprietà dei messaggi

Esistono due ragioni per impostare le proprietà del messaggio:

- Per consentire il flusso di parti del messaggio AMQP attraverso il gestore code senza influire sul payload del messaggio.
- Per consentire la selezione di `application-properties`.

La seguente tabella mostra le proprietà impostate dal messaggio AMQP:

Tabella 104. Proprietà messaggio AMQP

Nome proprietà	Nome MQRFH2	Tipo	Descrizione
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	Una stringa di identificazione per il canale AMQP. Viene utilizzato per generare il messaggio, in modo che le parti interessate possano indicare quale versione ha inserito il messaggio (ad esempio, il team del servizio durante la diagnosi dei problemi). Il valore non è convalidato dal gestore code e non deve essere documentato esternamente.
Versione AMQP	mq_amqp.Ver	MQTYPE_STRING	La versione del messaggio AMQP. Se non è presente, si presuppone "1.0". Il valore non viene convalidato dal gestore code.
AMQPCliente	mq_amqp.Cli	MQTYPE_STRING	Una stringa di identificazione per l'API. Viene utilizzato per inviare il messaggio AMQP al canale, in modo che le parti interessate possano indicare quale versione ha inserito il messaggio (ad esempio, il team di servizio durante la diagnosi dei problemi). Il valore non è convalidato dal gestore code e non deve essere documentato esternamente.
DurataAMQP	mq_amqp.Hdr.Dur	BOOLEAN MQTIPO	Il valore del campo `durable` nell'intestazione del messaggio AMQP, se impostato.
Priorità AMQP	mq_amqp.Hdr.Pri	MQTYPE_INT32	Il valore del campo `priority` nell'intestazione del messaggio AMQP, se impostato.
AMQPTTL	mq_amqp.Hdr.Ttl	MQTYPE_INT64	Il valore del campo `ttl` nell'intestazione del messaggio AMQP, se impostato.
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	BOOLEAN MQTIPO	Il valore del campo `first-acquirer` nell'intestazione del messaggio AMQP, se impostato.
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	Il valore del campo `delivery-count` nell'intestazione del messaggio AMQP, se impostato.
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	Il valore del campo `message-id` nelle proprietà AMQP, se impostato come stringa.
		MQTYPE_BYTE_STRING	Il valore del campo `message-id` nelle proprietà AMQP, se impostato come stringa di byte.

Tabella 104. Proprietà messaggio AMQP (Continua)			
Nome proprietà	**Nome MQRFH2**	**Tipo**	**Descrizione**
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	Il valore del campo `user-id` nelle proprietà AMQP, se impostato.
AMQPA	mq_amqp.Prp.To	MQTYPE_STRING	Il valore del campo `to` nelle proprietà AMQP, se impostato.
Oggetto AMQP	mq_amqp.Prp.Sub	MQTYPE_STRING	Il valore del campo `subject` nelle proprietà AMQP, se impostato.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	Il valore del campo `reply-to` nelle proprietà AMQP, se impostato.
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	Il valore del campo `correlation-id` nelle proprietà AMQP, se impostato come stringa.
		MQTYPE_BYTE_STRING	Il valore del campo `correlation-id` nelle proprietà AMQP, se impostato come stringa di byte.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	Il valore del campo `content-type` nelle proprietà AMQP, se impostato.
AMQPContentEncoding	mq_amqp.Prp.Cne	MQTYPE_STRING	Il valore del campo `content-encoding` nelle proprietà AMQP, se impostato.
Ora AMQPAbsoluteExpiry	mq_amqp.Prp.Aet	MQTYPE_STRING	Il valore del campo `absolute-expiry-time` nelle proprietà AMQP, se impostato.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	Il valore del campo `creation-time` nelle proprietà AMQP, se impostato.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	Il valore del campo `group-id` nelle proprietà AMQP, se impostato.
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	Il valore del campo `group-sequence` nelle proprietà AMQP, se impostato.
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	Il valore del campo `reply-to-group-id` nelle proprietà AMQP, se impostato.

Ogni proprietà dell'applicazione dal messaggio AMQP è impostata come una proprietà del messaggio IBM MQ . La sezione `application-properties` deve essere ricostituita in modo identico byte per byte, pertanto si applicano le seguenti limitazioni:

- Se una proprietà dell'applicazione viene rifiutata dal codice di convalida MQSETMP, il messaggio da rifiutare. Ad esempio:
  - La lunghezza del nome proprietà è limitata a MQ_MAX_PROPERTY_NAME_LENGTH.
  - Il nome della proprietà deve rispettare le regole definite da Java Language Specification for Java Identifiers.
  - Il nome della proprietà non deve iniziare con JMS o `usr.`JMS ad eccezione delle proprietà JMS documentate che possono essere impostate.
  - Il nome della proprietà non deve essere una parola chiave SQL.

- Una proprietà dell'applicazione contenente il carattere Unicode U+002E (".") causa il rifiuto del messaggio. La proprietà deve essere espressiva nel gruppo "usr" di proprietà utilizzato da JMS.
- Sono supportate solo le proprietà null, boolean, byte, short, int, long, float, double, binary e string. Una proprietà dell'applicazione con qualsiasi altro tipo causerà il rifiuto del messaggio.

V 9.3.0 È possibile configurare le seguenti proprietà JMS utilizzando `application-properties`:

- JMS_IBM_REPORT_ECCEZIONE
- JMS_IBM_REPORT_EXPIRATION
- COA IBM REPORT_JMS
- COD REPORT_IBM_JMS
- PAN IBM_REPORT_JMS
- NAN_IBM_REPORT_JMS
- ID_MSG_PASS_IBM_REPORT_JMS
- ID_CORREL_IBM_REPORT_JMS
- MSG_DISCARD_IBM_REPORT_JMS JMS

Notare che i nomi e i valori delle proprietà sono congruenti con i dettagli “Associazione di campi specifici del fornitore JMS” a pagina 162 equivalenti e che i valori non validi vengono ignorati.

payload

- Per un AMQP body con una singola sezione di dati binari, i dati binari (esclusi i bit AMQP) vengono inseriti come payload IBM MQ , con un formato MQFMT_NONE.
- Per un body AMQP con una singola sezione di dati stringa, i dati stringa (esclusi i bit AMQP) vengono inseriti come payload IBM MQ , con un Formato MQFMT_STRING.
- Altrimenti, l'AMQP body forma il payload così com' è, con un formato MQFMT_AMQP.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Affidabilità della consegna dei messaggi

Questa sezione confronta le funzioni di affidabilità per MQ Light API e Apache Qpid JMS.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Affidabilità del messaggio MQ Light

Esistono quattro funzioni dell'API MQ Light che consentono di controllare l'affidabilità della consegna dei messaggi alle e dalle applicazioni AMQP.

Essi sono:

- “QOS (Quality of Service) del messaggio” a pagina 691
- “Conferma automatica sottoscrittore” a pagina 692
- “Durata sottoscrizione” a pagina 693
- “Persistenza messaggio” a pagina 693

QOS (Quality of Service) del messaggio

MQ Light API offre due qualità di servizio:

- Al massimo una volta
- Almeno una volta

È possibile scegliere quale qualità del servizio si desidera che gli editori e i sottoscrittori utilizzino.

Se si utilizza un client MQ Light , impostare l'opzione **qos** client o subscribe su *QOS_AT_MOST_ONCE* o su *QOS_AT_LEAST_ONCE*.

Se si utilizza un client AMQP diverso, impostare l'attributo **settled** del frame di trasferimento (per i publisher) o il frame di disposizione (per i sottoscrittori) su *true* o *false*, in base alla qualità del servizio che si desidera ottenere.

La qualità del servizio determina quando un messaggio viene eliminato dal lato `sending` di una conversazione.

Editoriale

Se un publisher sceglie **QOS** *0* (al massimo una volta), il publisher non attende un riconoscimento dal gestore code, prima di eliminare la copia del messaggio.

Se la connessione al gestore code ha esito negativo prima del completamento dell'invio, il messaggio potrebbe non essere ricevuto dai sottoscrittori.

Se un publisher sceglie **QOS** *1* (almeno una volta), il publisher attende che il gestore code riconosca che il messaggio è stato scritto nelle code del sottoscrittore prima di eliminare la sua copia del messaggio.

Se la connessione al gestore code non riesce durante l'invio, il publisher invia nuovamente il messaggio una volta che si è riconnesso al gestore code.

Sottoscrizione

Se un sottoscrittore sceglie **QOS** *0* , il gestore code non attende un riconoscimento da parte del sottoscrittore prima di eliminare la sua copia del messaggio.

Se la connessione al sottoscrittore ha esito negativo prima che il sottoscrittore abbia ricevuto il messaggio, tale messaggio potrebbe essere perso.

Se un sottoscrittore sceglie **QOS** *1* , il gestore code attende un riconoscimento dal sottoscrittore prima di eliminare la sua copia del messaggio.

Da IBM MQ 9.3.3, i messaggi riconosciuti vengono rimossi in batch per migliorare le prestazioni. Per ulteriori informazioni, consultare “Rimozione dei messaggi AMQP riconosciuti dalla coda nei batch” a pagina 694.

Se la connessione al sottoscrittore ha esito negativo prima che il sottoscrittore abbia ricevuto il messaggio, il messaggio viene conservato dal gestore code. Il gestore code invia nuovamente il messaggio al sottoscrittore quando il gestore code si riconnette o a un altro sottoscrittore se la sottoscrizione è condivisa.

Conferma automatica sottoscrittore

Se un sottoscrittore sceglie **QOS** *1* (almeno una volta), deve confermare la ricezione di ciascun messaggio prima che il gestore code ne elimini la copia. Il sottoscrittore può decidere quando confermare i messaggi.

Con **auto-confirm** impostato su *true*, il client di MQ Light riconosce automaticamente la consegna di ciascun messaggio, una volta che il client ha ricevuto correttamente il messaggio sulla rete.

Ciò garantisce che se si verifica un errore di rete, il messaggio viene riconsegnato all'applicazione. Tuttavia, è ancora possibile che l'applicazione perda il messaggio, se l'applicazione ha esito negativo tra il client MQ Light che riconosce il messaggio e l'applicazione che lo elabora.

Con **auto-confirm** impostato su *false*, il client MQ Light non riconosce automaticamente la consegna del messaggio, ma lo lascia all'applicazione per decidere quando deve essere riconosciuto.

Ciò consente a un'applicazione di effettuare un aggiornamento a una risorsa esterna, ad esempio un database o un file, prima di confermare al gestore code che il messaggio è stato elaborato e può essere eliminato.

Durata sottoscrizione

Quando un'applicazione esegue la sottoscrizione, sceglie se la sottoscrizione e la destinazione in cui sono memorizzati i messaggi per tale sottoscrizione continuano a esistere anche dopo la disconnessione dell'applicazione.

L' MQ Light opzione di sottoscrizione **ttl** viene utilizzata per specificare il tempo (in millisecondi) in cui una sottoscrizione continua a esistere dopo la disconnessione dell'applicazione. Se l'applicazione si riconnette prima di questo periodo di tempo, la sottoscrizione viene ripresa e l'applicazione può continuare a utilizzare i messaggi da tale sottoscrizione.

Se il periodo time - to - live trascorre senza che l'applicazione si riconnette, la sottoscrizione viene rimossa e tutti i messaggi memorizzati nella relativa destinazione vengono persi, anche se si tratta di messaggi persistenti.

Se è importante non perdere messaggi, è necessario specificare un valore TTL (time - to - live) per l'applicazione, che sia sufficientemente elevato per garantire che i messaggi non vengano persi durante un'interruzione.

Persistenza messaggio

La persistenza dei messaggi è controllata dalle applicazioni di pubblicazione e sottoscrizione e dalla configurazione degli oggetti argomento IBM MQ .

Se il sottoscrittore AMQP utilizza **QOS** *0* (al massimo una volta) e crea una sottoscrizione non durevole, il canale AMQP inserisce sempre i messaggi non persistenti nella coda del sottoscrittore, indipendentemente dalle altre opzioni descritte nel testo seguente.

Tenere presente che, se il gestore code viene arrestato, sia la sottoscrizione che i messaggi andranno persi.

Se un programma di pubblicazione AMQP imposta l'intestazione **durable** AMQP su *true*, il canale AMQP inserisce i messaggi persistenti nelle code del sottoscrittore.

Se il gestore code è stato arrestato per qualsiasi motivo, i messaggi sono ancora disponibili per i sottoscrittori quando il gestore code viene riavviato.

Se l'intestazione **durable** non è impostata, il canale AMQP sceglie la persistenza dei messaggi pubblicati in base all'attributo **DEFPSIST** dell'oggetto argomento IBM MQ pertinente.

Per impostazione predefinita, questo è il SISTEMA SYSTEM.BASE.TOPIC, che utilizza un attributo **DEFPSIST** *NO* (non persistente).

Attenzione: Le versioni successive del client di MQ Light non supportano l'impostazione dell'intestazione durevole AMQP.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Affidabilità del messaggio Apache Qpid JMS

Esistono quattro funzionalità della libreria Apache Qpid™ JMS che consentono di controllare l'affidabilità della consegna dei messaggi alle applicazioni AMQP e da esse.

Questi sono per:

- “Editoriale” a pagina 694 V 9.3.0 /Producer per messaggistica punto a punto
  - scadenza del messaggio
  - Persistenza messaggio
- “Sottoscrizione” a pagina 694
  - Durata sottoscrizione

– Modalità di riconoscimento sessione V 9.3.0 (applicabile anche per la messaggistica point to point del consumer)

Editoriale

scadenza del messaggio

L'impostazione del valore TTL (time - to - live) del produttore JMS influenza il tempo di scadenza assegnato ai messaggi pubblicati da tale produttore di messaggi.

Assicurarsi che il valore TTL (time - to - live) per un produttore JMS sia sufficientemente grande da consentire l'utilizzo dei messaggi prima della scadenza.

In alternativa, lasciando il valore time - to - live non impostato, si impedisce al messaggio di scadere dalla coda di sottoscrizione.

Persistenza messaggio

L'impostazione della modalità di consegna del produttore del messaggio JMS imposta la persistenza del messaggio IBM MQ pubblicato nell'argomento specificato.

Assicurarsi di utilizzare **`DeliveryMode`**.PERSISTENTE per i messaggi che devono essere conservati quando un gestore code viene terminato o ha un'interruzione.

Sottoscrizione

Durata sottoscrizione

I canali AMQP supportano la creazione di sottoscrizioni durevoli utilizzando le versioni durevoli dei metodi consumer di creazione JMS :

- **`createDurableConsumer()`**
- **`createSharedDurableConsumer()`**

Modalità riconoscimento sessione

Per garantire che un messaggio utilizzato sia stato completamente elaborato prima di essere rimosso dalla coda di sottoscrizione IBM MQ , creare una sessione JMS utilizzando **`Session`**.Utilizzare il metodo **`message.acknowledge()`** per riconoscere questo messaggio e tutti gli altri precedentemente ricevuti in questa sessione.

Concetti correlati

Sviluppo di applicazioni client AMQP
Il supporto IBM MQ per le API AMQP, permette a un'amministratore IBM MQ di creare un canale AMQP. Quando viene avviato, questo canale definisce un numero di porta che accetta le connessioni dalle applicazioni client AMQP.

V 9.3.3 Rimozione dei messaggi AMQP riconosciuti dalla coda nei batch

Se un'applicazione AMQP sta utilizzando la consegna del messaggio QOS_AT_LEAST_ONCE (1), il servizio AMQP attende un riconoscimento dall'applicazione prima di eliminare la copia di un messaggio che conserva dopo aver inviato tale messaggio all'applicazione. Da IBM MQ 9.3.3, i messaggi che sono stati riconosciuti vengono rimossi dalla coda in batch, invece che singolarmente, con un conseguente miglioramento delle prestazioni.

Informazioni su questa attività

Per Long Term Support e Continuous Delivery prima di IBM MQ 9.3.3, ogni singolo messaggio viene rimosso dalla coda.

Da IBM MQ 9.3.3, è possibile utilizzare le due proprietà di sistema **com.ibm.mq.AMQP.BATCHSZ** e **com.ibm.mq.AMQP.BATCHINT** per ottimizzare l'elaborazione dei riconoscimenti in batch per migliorare le prestazioni:

com.ibm.mq.AMQP.BATCHSZ
Questo attributo definisce il numero massimo di riconoscimenti da ricevere prima che il servizio AMQP rimuova i messaggi. Il numero può essere compreso tra 1 e 9999. Se è impostato un numero non valido o se il numero specificato non è compreso nell'intervallo, viene utilizzato il valore predefinito 50.

La dimensione batch non influisce sul modo in cui vengono trasferiti i messaggi. I messaggi vengono sempre trasferiti singolarmente, ma vengono rimossi in batch dopo che il servizio AMQP riceve i riconoscimenti. La dimensione effettiva di un batch può essere inferiore al valore specificato da **com.ibm.mq.AMQP.BATCHINT**. Ad esempio, un batch viene completato se il periodo impostato dall'attributo **com.ibm.mq.AMQP.BATCHINT** scade.

com.ibm.mq.AMQP.BATCHINT
Questo attributo definisce la quantità di tempo, in millisecondi, per cui il servizio AMQP conserva i messaggi riconosciuti nella coda. Se il batch non è pieno, il batch viene cancellato dopo questa durata. È possibile specificare un numero qualsiasi di millisecondi, compreso tra 1 e 999 999 999. Il valore predefinito è 50. Se non si specifica un valore per questo attributo, viene utilizzato il valore predefinito 50.

Note:

1. Se il servizio AMQP attende un riconoscimento prima di eliminare un messaggio dipende da quale delle seguenti due qualità di servizio un'applicazione sta utilizzando per la consegna del messaggio:
   - QOS per QOS AT_MOST_ONCE (0)

     Se un'applicazione AMQP utilizza questa QoS (quality of service), non riconosce i messaggi, quindi il servizio AMQP elimina i messaggi dopo averli inviati all'applicazione senza attendere un riconoscimento.
   - QOS_AT_LEAST_ONCE (1)

     Se un'applicazione AMQP utilizza questo QoS (quality of service), riconosce i messaggi, quindi il servizio AMQP conserva una copia di ciascun messaggio dopo averlo inviato all'applicazione fino a quando non riceve un riconoscimento dall'applicazione. Se l'applicazione si disconnette o perde la connessione prima di confermare il messaggio, il messaggio viene reso disponibile ad altre applicazioni. Il servizio AMQP non rimuove un messaggio dalla coda fino a quando non viene riconosciuto.
2. MQ Appliance Le proprietà di sistema **com.ibm.mq.AMQP.BATCHSZ** e **com.ibm.mq.AMQP.BATCHINT** non sono applicabili su IBM MQ Appliance. Un valore predefinito di 50 viene utilizzato su IBM MQ Appliance.

.

Procedura

Utilizzare le proprietà di sistema **com.ibm.mq.AMQP.BATCHSZ** e **com.ibm.mq.AMQP.BATCHINT** per ottimizzare l'elaborazione dei riconoscimenti in batch.

Da IBM MQ 9.3.3, quando il gestore code viene creato, il file `amqp_java.properties` contiene i seguenti valori predefiniti per le proprietà di sistema:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

A seconda della frequenza dei messaggi utilizzati, è possibile ottimizzare l'elaborazione dei riconoscimenti in batch per migliorare le prestazioni. Un gestore code migrato non ha queste proprietà nel file `amqp_java.properties` . Quindi, per un gestore code migrato o se le proprietà non sono impostate, vengono utilizzati i valori predefiniti. È possibile aggiungere queste proprietà per ottimizzare i valori delle prestazioni ottimizzate.

I messaggi riconosciuti vengono rimossi in batch quando si verifica una delle seguenti condizioni:

- Il numero di messaggi riconosciuti raggiunge **com.ibm.mq.AMQP.BATCHSZ**.
- **com.ibm.mq.AMQP.BATCHINT** viene superato dopo l'avvio del batch.
- L'applicazione scollega o chiude la coda o l'argomento prima che siano soddisfatte le due precedenti condizioni.

ALW Topologie per i client AMQP con IBM MQ

Topologie di esempio per aiutarti a sviluppare i tuoi client AMQP per lavorare con IBM MQ.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Client AMQP che comunicano su IBM MQ

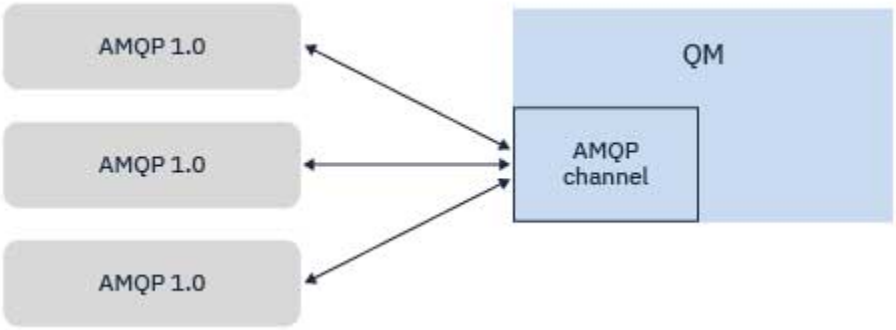
È possibile utilizzare IBM MQ come provider di messaggistica per qualsiasi applicazione conforme a AMQP 1.0. Sebbene qualsiasi client AMQP 1.0 possa connettersi a un canale AMQP, alcune funzioni AMQP non sono supportate, ad esempio transazioni o sessioni multiple.

Definendo uno o più canali AMQP, i client AMQP 1.0 possono connettersi al gestore code e inviare messaggi a una stringa di argomenti. I clienti possono anche sottoscrivere un modello di argomento per ricevere messaggi che corrispondono al modello.

Nel seguente scenario, le sole applicazioni che inviano e ricevono messaggi sono le applicazioni AMQP 1.0 .

Le applicazioni possono scegliere se le destinazioni create sottoscrivendo una stringa di argomenti sono persistenti in modo che i messaggi non vadano persi se l'applicazione perde temporaneamente la connessione al gestore code.

Le applicazioni possono anche scegliere per quanto tempo conservare i messaggi prima di eliminarli dalla destinazione.



Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Client AMQP che scambiano messaggi con applicazioni IBM MQ

Definendo e avviando un canale AMQP, le applicazioni AMQP 1.0 possono pubblicare i messaggi ricevuti dalle applicazioni MQ esistenti. I messaggi pubblicati tramite un canale AMQP vengono tutti inviati agli argomenti MQ , non alle code MQ . Un'applicazione MQ che ha creato una sottoscrizione utilizzando la chiamata API MQSUB riceve i messaggi pubblicati dalle applicazioni AMQP 1.0 , a condizione che la stringa argomento o l'oggetto argomento utilizzato dall'applicazione MQ corrisponda alla stringa argomento pubblicata dal client AMQP.

I dati, gli attributi e le proprietà del messaggio AMQP sono impostati sul messaggio MQ ricevuto dall'applicazione MQ . Per ulteriori informazioni sulle mappature di messaggi da AMQP a MQ , consultare “Mappatura dei campi AMQP sui campi IBM MQ (messaggi in entrata)” a pagina 687.

Se l'applicazione MQ ha creato una sottoscrizione durevole, i messaggi pubblicati dall'applicazione AMQP vengono memorizzati sulla coda che supporta la sottoscrizione. I messaggi vengono quindi ricevuti dall'applicazione MQ quando l'applicazione riprende la sottoscrizione. Se l'applicazione AMQP specifica un TTL (time to live) del messaggio e l'applicazione MQ non si riconnette entro il TTL (time to live), il messaggio è scaduto dalla coda.

Le applicazioni AMQP 1.0 possono anche utilizzare i messaggi pubblicati da applicazioni MQ esistenti. I messaggi pubblicati dalle applicazioni MQ su un argomento o una stringa di argomento MQ vengono ricevuti da un'applicazione AMQP 1.0 a condizione che l'applicazione abbia effettuato la sottoscrizione con un modello di argomento che corrisponde alla stringa di argomento pubblicata.

Se l'applicazione AMQP 1.0 specifica un valore TTL (time - to - live) per la sottoscrizione e l'applicazione AMQP si disconnette per un periodo superiore al TTL (time - to - live), la sottoscrizione scade dal gestore code e tutti i messaggi memorizzati nella coda di sottoscrizione vengono persi.

I campi MQMD, le proprietà del messaggio e i dati dell'applicazione sono impostati sul messaggio AMQP ricevuto dall'applicazione AMQP. Per ulteriori informazioni sulle associazioni di messaggi da MQ a AMQP, consultare “Mappatura dei campi IBM MQ sui campi AMQP (messaggi in uscita)” a pagina 685.

Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Configurazione di client AMQP per interagire direttamente con applicazioni su code IBM MQ

V 9.3.0 L'implementazione IBM MQ AMQP supporta la pubblicazione / sottoscrizione e il punto a punto. Per qualsiasi client AMQP che non supporta point to point, utilizzare la seguente procedura per inviare messaggi a una coda o ricevere messaggi da una coda.

Panoramica

Ad esempio, si supponga che un'applicazione riceva i messaggi da una coda di immissione `IN_QUEUE` e li inserisci in una coda di emissione `OUT_QUEUE`. È possibile per i client AMQP inserire messaggi in `IN_QUEUE`e ricevere messaggi da `OUT_QUEUE`

Nota: Non sono richieste modifiche all'applicazione stessa.

Per un programma di pubblicazione AMQP per inserire un messaggio in una coda, è necessario creare una sottoscrizione di gestione per la stringa di argomenti in cui il client AMQP sta pubblicando, con una destinazione della coda desiderata; consultare “Invio di messaggi all'applicazione:” a pagina 698.

Per un sottoscrittore AMQP per ottenere messaggi da una coda, è necessario sostituire la coda con una coda alias con lo stesso nome, con una destinazione di un oggetto argomento che rappresenta la stringa di argomenti a cui è sottoscritto il client AMQP; consultare “Richiamo dei messaggi dall'applicazione:” a pagina 698 .

Invio di messaggi all'applicazione:

L'applicazione sta già raccogliando i messaggi da IN_QUEUE e si desidera che il client AMQP sia in grado di pubblicare i messaggi, in modo che vadano in questa coda per essere elaborati dall'applicazione.

A tale scopo, si crea una nuova sottoscrizione di gestione, in cui la stringa di argomenti da cui questa sottoscrizione riceve i messaggi, è la stringa di argomenti in cui il client AMQP pubblica. La coda di destinazione di questa sottoscrizione è la coda di immissione per l'applicazione, IN_QUEUE.

Tutti i messaggi pubblicati nella stringa di argomenti definita per tale sottoscrizione di gestione vengono instradati alla destinazione definita, in questo caso IN_QUEUE.

Supponendo che il client AMQP pubblichi in una stringa di argomenti /application/in, è possibile creare una sottoscrizione amministrativa APP_IN, utilizzando il seguente comando MQSC:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

Una volta definito questo oggetto, tutti i messaggi pubblicati in /application/in vengono instradati alla destinazione IN_QUEUE, dove vengono prelevati dall'applicazione allo stesso modo di qualsiasi altro messaggio inserito in questa coda da altre applicazioni.

Richiamo dei messaggi dall'applicazione:

L'applicazione sta inserendo i messaggi in OUT_QUEUE, dove possono essere prelevati ed elaborati da altri client.

Tuttavia, in questo caso, si desidera che i messaggi vengano consegnati a un client AMQP, ma i client AMQP utilizzano solo la pubblicazione / sottoscrizione e non possono raccogliere i messaggi direttamente dalla coda.

Per sostituire i client che precedentemente ricevevano il messaggio con il client AMQP di sottoscrizione, è necessario creare un oggetto argomento, per la stringa dell'argomento a cui è sottoscritto il client AMQP e una coda alias.

Attenzione: Se si definisce la coda alias e si avvia l'applicazione di produzione prima che qualsiasi client AMQP abbia la possibilità di sottoscrivere, i messaggi che l'applicazione di produzione invia alla "coda" (ora un argomento) andranno persi perché non ci sono sottoscrittori.

Le modifiche descritte in questo testo, sostituiscono i client che precedentemente ricevevano i messaggi solo con il client AMQP di sottoscrizione. Per utilizzare una combinazione di AMQP e altri client per ottenere i messaggi, sono necessarie modifiche più estese.

Supponendo che il client AMQP sottoscrivi una stringa argomento /application/out, è possibile definire l'oggetto argomento APP_OUT utilizzando il seguente comando MQSC:

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

I messaggi consegnati a questo oggetto argomento vengono consegnati al client AMQP che effettua la sottoscrizione alla stessa stringa argomento.

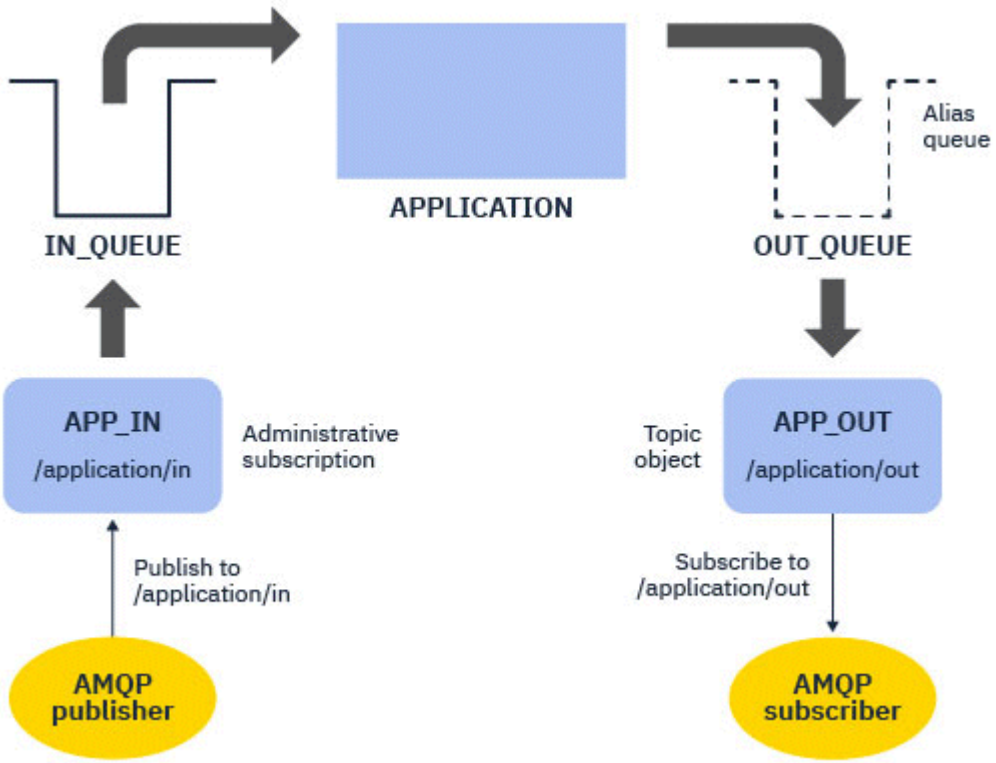
È quindi necessario assicurarsi che i messaggi inseriti in OUT_QUEUE dall'applicazione vengano consegnati a questo nuovo oggetto argomento, in modo che vengano inviati al client di sottoscrizione.

A tale scopo, sostituire la coda esistente OUT_QUEUE con una coda alias con lo stesso nome, con un tipo di destinazione dell'oggetto argomento appena creato, utilizzando il seguente comando MQSC:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Ora, i messaggi inseriti dall'applicazione in OUT_QUEUE non attendono che la coda venga prelevata; vengono invece consegnati alla destinazione di questa coda alias, ossia il nuovo oggetto argomento APP_OUT.

Il client AMQP, che è sottoscritto alla stringa argomento rappresentata da questo oggetto argomento /application/out, riceve quindi tutti i messaggi inviati a questo oggetto argomento dalla coda alias.



Attività correlate

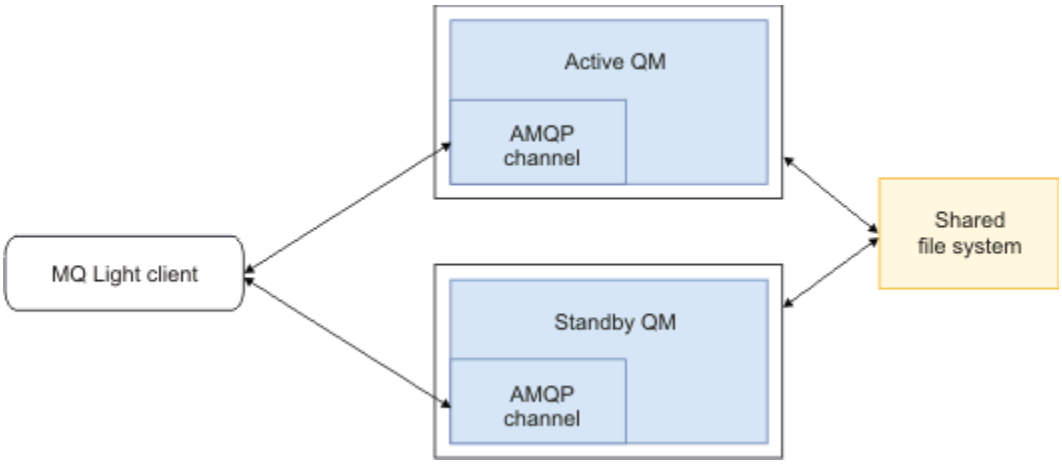
Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Configurazione di un client AMQP per l'alta disponibilità

È possibile configurare le applicazioni AMQP 1.0 per la connessione all'istanza attiva di un gestore code a più istanze IBM MQ e il failover all'istanza in standby del gestore code a più istanze in una coppia HA (High Availability). A tale scopo, configurare l'applicazione AMQP con due indirizzi IP e coppie di porte.

È possibile configurare l'API client AMQP con una funzione personalizzata, che viene richiamata se il client perde la propria connessione al server. La funzione può connettersi a un indirizzo IP alternativo, ad esempio un gestore code IBM MQ in standby o all'indirizzo IP originale. Per altri client AMQP, se il client supporta la configurazione di più endpoint di connessione, configurare l'applicazione con due coppie host - porta e utilizzare le funzioni di riconnessione fornite dalla libreria AMQP per passare al gestore code in standby.



Attività correlate

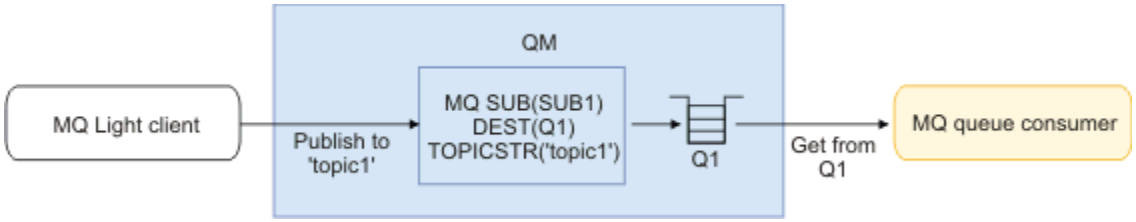
Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Configurazione della pubblicazione / sottoscrizione per i client AMQP

I client AMQP possono pubblicare in un argomento con una sottoscrizione IBM MQ che instrada i messaggi a una coda IBM MQ letta da un'applicazione esistente. Se si desidera che l'applicazione AMQP 1.0 invii messaggi a un'applicazione IBM MQ esistente configurata per la lettura da una coda, è necessario definire una sottoscrizione IBM MQ gestita sul gestore code.

Configurare la sottoscrizione per utilizzare un pattern di argomento che corrisponda alla stringa di argomento utilizzata dall'applicazione AMQP. Impostare la destinazione della sottoscrizione sul nome della coda da cui l'applicazione IBM MQ riceve o sfoglia i messaggi.



Attività correlate

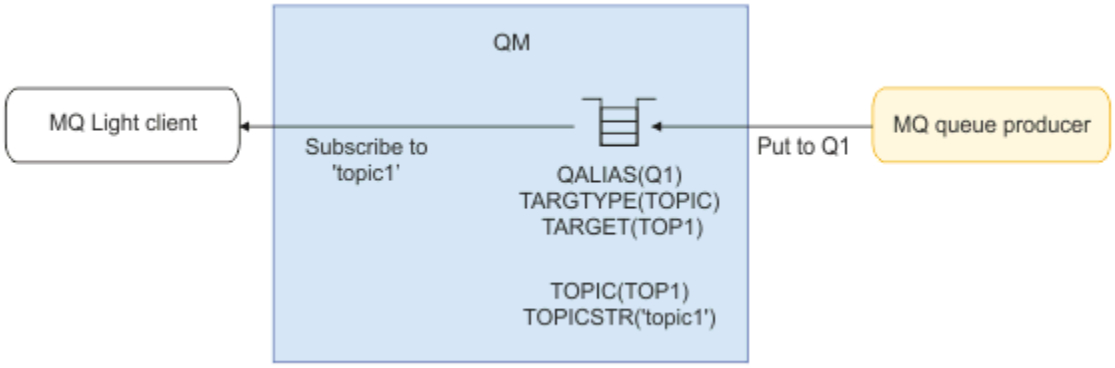
Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Client AMQP che utilizza un alias coda per ricevere messaggi da un'applicazione IBM MQ

Un cliente AMQP può sottoscrivere un argomento e ricevere messaggi inseriti in una coda alias da un'applicazione IBM MQ . Se si desidera che l'applicazione AMQP 1.0 riceva messaggi da un'applicazione IBM MQ esistente configurata per inserire messaggi in una coda, è necessario definire un alias della coda (QALIAS) sul gestore code.

L'alias coda deve avere lo stesso nome della coda che l'applicazione IBM MQ apre per l'inserimento. L'alias della coda deve specificare un tipo di base TOPIC e un oggetto di base di un oggetto argomento IBM MQ che ha una stringa di argomento che corrisponde al modello di argomento sottoscritto dall'applicazione AMQP.



Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Il client AMQP che inoltra richieste e utilizza risposte da un server delle applicazioni

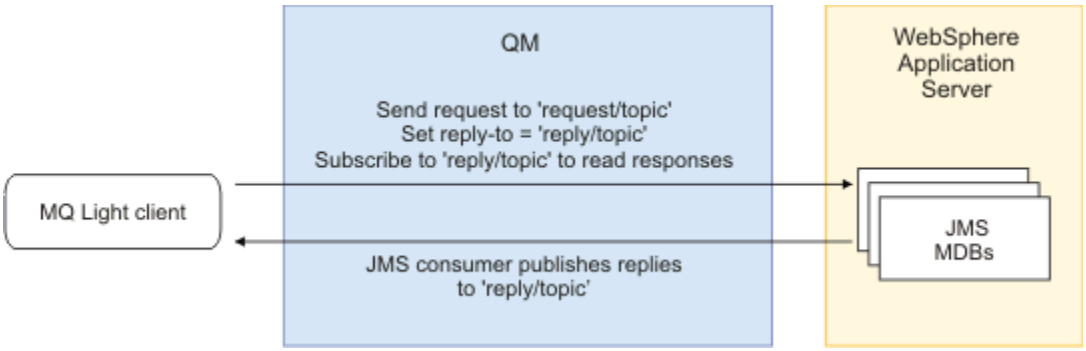
Un client AMQP può inoltrare richieste a un bean basato sui messaggi in esecuzione in un server delle applicazioni e utilizzare le risposte da un argomento di risposta. IBM MQ supporta le applicazioni AMQP 1.0 impostando un argomento di risposta nei messaggi pubblicati da IBM MQ . Quando un messaggio AMQP viene pubblicato con l'attributo reply - to impostato, il valore del campo reply - to viene impostato come proprietà JMS per la ricezione da parte dei consumer JMS. Questa impostazione consente agli utenti JMS di leggere l'argomento di risposta dal messaggio e di inviare un messaggio di risposta al client AMQP.

La proprietà JMS è **JMSReplyTo**. La stringa di risposta AMQP deve essere uno dei seguenti tipi:

- Una stringa di argomenti. Ad esempio, `'reply/topic'`
- Un indirizzo AMQP URL nel formato `amqp://host:port/[topic-string]`. Ad esempio, `amqp://localhost:5672/reply/topic`

Se si specifica un indirizzo AMQP URL come campo di risposta, tutto tranne la stringa di argomenti alla fine dell' URL viene rimosso prima di impostare la proprietà **JMSReplyTo** .

Per ulteriori informazioni sulle associazioni da un indirizzo di risposta AMQP a una proprietà **JMSReplyTo** , consultare “Mappatura dei campi AMQP sui campi IBM MQ (messaggi in entrata)” a pagina 687



Attività correlate

Creazione e utilizzo di canali AMQP

Protezione dei client AMQP

ALW Interoperabilità tra applicazioni MQ Light e Apache Qpid JMS

Le applicazioni MQ Light e Apache Qpid JMS funzionano in modo simile e, quando si sottoscrive un argomento, creano sottoscrizioni IBM MQ che seguono la stessa convenzione di denominazione.

Sottoscrizione privata, non condivisa

Il nome della sottoscrizione IBM MQ creata dall'applicazione è `:private:<clientid>:<topicstring>`.

Un'applicazione che utilizza un ID client diverso non è in grado di accedere alle sottoscrizioni create da altre applicazioni, poiché il nome della sottoscrizione viene generato automaticamente e include l'ID client AMQP.

Sia le applicazioni Apache Qpid JMS che MQ Light utilizzano questa convenzione di denominazione per le sottoscrizioni private.

Sottoscrizioni condivise globalmente

Il nome di una sottoscrizione IBM MQ condivisa globalmente, creata da un client AMQP è `:share:<sharename>:<topicstring>`.

Se più applicazioni con ID client AMQP differenti specificano lo stesso nome di condivisione e la stessa stringa di argomenti, condividono una singola sottoscrizione e possono lavorare insieme per elaborare i messaggi per tale sottoscrizione. Puoi utilizzare questo modello se vuoi scalare il numero di applicazioni di lavoro che svuotano i messaggi da una sottoscrizione.

Entrambe le applicazioni Apache Qpid JMS e MQ Light utilizzano questa convenzione di denominazione per sottoscrizioni condivise globalmente. Nel caso di Apache Qpid JMS, ciò richiede che la connessione JMS non abbia un ID client specificato.

La libreria Apache Qpid JMS genera automaticamente un ID client AMQP, ma questo ID client non viene utilizzato per la denominazione della sottoscrizione IBM MQ .

Nota: Le sottoscrizioni condivise globalmente sono ancora nell'ambito di un singolo gestore code.

Sottoscrizioni condivise private

Il nome di una sottoscrizione IBM MQ condivisa privatamente creata da un client AMQP è `:privateshare:<clientid>:<sharename>:<topicstring>`.

Se diversi thread da una singola applicazione Apache Qpid JMS utilizzano lo stesso nome condivisione e la stessa stringa argomento e un ID client è stato configurato sulla connessione JMS , tali thread condividono lo stesso oggetto sottoscrizione IBM MQ .

Tuttavia, altre connessioni Apache Qpid JMS non sono in grado di condividere la sottoscrizione in quanto devono utilizzare un ID client differente.

I client MQ Light non supportano il concetto di sottoscrizioni condivise private e non possono utilizzare i messaggi di una sottoscrizione condivisa privata creata da un'applicazione Apache Qpid JMS .

Sottoscrizioni IBM MQ JMS

Le sottoscrizioni IBM MQ JMS utilizzano uno schema di denominazione diverso dai canali AMQP. Non è possibile per le applicazioni MQ Light o Apache Qpid JMS condividere le sottoscrizioni con le applicazioni IBM MQ JMS .

Concetti correlati

Sviluppo di applicazioni client AMQP
Il supporto IBM MQ per le API AMQP, permette a un'amministratore IBM MQ di creare un canale AMQP. Quando viene avviato, questo canale definisce un numero di porta che accetta le connessioni dalle applicazioni client AMQP.

ALW IBM MQ Proprietà del controllo listener AMQP

Per prestazioni migliori in un'applicazione a più thread, è possibile ottimizzare il numero di thread di lavoro che il servizio AMQP deve utilizzare configurando una proprietà nel file delle proprietà AMQP.

È possibile configurare le proprietà del servizio listener AMQP nei seguenti file delle proprietà:

- Windows Su sistemi Windows : `amqp_win.properties` .
- Linux AIX Su sistemi AIX and Linux : `amqp_unix.properties` .

Le proprietà che è possibile configurare sono le seguenti:

Tabella 105. Proprietà del servizio listener AMQP

Proprietà	Descrizione
com.ibm.mq.MQXR.Workers	Il numero di thread di lavoro del server creati dal servizio listener AMQP. Se questo valore non viene specificato, viene utilizzato come valore predefinito per essere uguale al numero di processori logici sul sistema.

Tabella 105. Proprietà del servizio listener AMQP (Continua)	
Proprietà	**Descrizione**
MQIBindType	Il tipo di binding per il servizio AMQP: FASTPATH, SHAREDo ISOLATO. Il valore predefinito è FASTPATH.

Il servizio listener AMQP bilancia il carico di lavoro della connessione client su un numero di thread di lavoro. Il numero di thread di lavoro che il servizio AMQP deve utilizzare può essere specificato utilizzando la proprietà **com.ibm.mq.MQXR.Workers** .

L'amministratore del gestore code IBM MQ può ottimizzare il numero di thread di lavoro per migliorare le prestazioni in un'applicazione a più thread. In genere, le prestazioni migliori si ottengono quando il numero di thread di lavoro corrisponde al numero di processori logici sul sistema. Tuttavia, questo potrebbe non essere sempre il caso per alcune configurazioni di macchina e caratteristiche di carico client, quindi potrebbe essere necessario un elemento di ottimizzazione per trovare il valore ottimale per il numero di thread di lavoro.

Prima di eseguire l'ottimizzazione, accertarsi di comprendere a fondo la natura delle applicazioni client e i relativi carichi di lavoro. La misurazione delle prestazioni dell'applicazione con diversi conteggi di thread e benchmark dovrebbe aiutare a determinare il valore ottimale per il numero di thread di lavoro.

Nota: MQ Appliance Queste proprietà non sono applicabili in IBM MQ Appliance. I valori predefiniti vengono utilizzati in IBM MQ Appliance.

Sviluppo di applicazioni REST con IBM MQ

È possibile sviluppare applicazioni REST per inviare e ricevere messaggi. IBM MQ supporta diverse API REST a seconda della piattaforma e della funzionalità.

Le seguenti opzioni sono le IBM MQ opzioni supportate tra cui è possibile scegliere per inviare e ricevere messaggi da IBM MQ:

- IBM MQ messaging REST API
- IBM z/OS Connect EE
- IBM Integration Bus
- DataPower

IBM MQ messaging REST API

È possibile utilizzare messaging REST API per inviare, ricevere e sfogliare i messaggi IBM MQ in formato di testo semplice. messaging REST API è abilitato per impostazione predefinita.

Il supporto viene fornito per un numero di intestazioni HTTP differenti che possono essere utilizzate per impostare le proprietà comuni del messaggio.

messaging REST API è completamente integrato con la sicurezza IBM MQ . Per utilizzare messaging REST API, gli utenti devono essere autenticati sul server mqweb e devono essere un membro del ruolo MQWebUser .

Per ulteriori informazioni, fare riferimento a “Messaggistica mediante REST API” a pagina 705. Vedi anche Esercitazione: Introduzione a IBM MQ messaging REST API su IBM Developer, che include esempi Go e Node.js per l'utilizzo dell'API REST di messaggistica.

IBM z/OS Connect EE

IBM z/OS Connect EE è un prodotto z/OS che ti consente di creare API REST su asset z/OS esistenti, come transazioni CICS o IMS , code e argomenti IBM MQ . L'asset z/OS esistente è nascosto all'utente.

Ciò consente di abilitare REST gli asset senza modificarli o senza alcuna delle applicazioni esistenti che li utilizzano.

IBM z/OS Connect EE fornisce la trasformazione automatica dei dati per la conversione tra i dati JSON utilizzati dalle API REST e le strutture del linguaggio più tradizionali, ad esempio COBOL, previsto da molte applicazioni mainframe.

Il toolkit API IBM z/OS Connect EE basato su Eclipse può essere utilizzato per creare un'API RESTful completa utilizzando i parametri di query e i segmenti di percorso URL , manipolando il formato JSON mentre fluisce attraverso il runtime IBM z/OS Connect EE.

IBM z/OS Connect EE può essere utilizzato per esporre code e argomenti IBM MQ come API RESTful tramite il provider del servizio IBM MQ . Sono supportati due diversi tipi di servizio:

- Servizi unidirezionali: forniscono un'API REST che consente l'esecuzione di una singola operazione IBM MQ su una coda o su un argomento. In base alla configurazione esatta, una richiesta HTTP può causare l'invio di un messaggio a una coda o la pubblicazione in un argomento; oppure una richiesta HTTP può causare la ricezione distruttiva di un messaggio da una coda
- Servizi a due vie: forniscono un'API REST su una coppia di code utilizzate da un'applicazione di tipo richiesta - risposta di back - end. I chiamanti emettono una richiesta HTTP al servizio bidirezionale. Il payload di richiesta HTTP viene trasformato da JSON in una struttura del linguaggio tradizionale e inserito in una coda di richiesta in cui viene elaborato dall'applicazione di back - end e una risposta collocata nella coda di risposta. Questa risposta viene richiamata dal servizio, convertita dalla struttura del linguaggio tradizionale in JSON e restituita al chiamante come corpo della risposta POST.

Per ulteriori informazioni su IBM z/OS Connect EE, consultare z/OS Connect EE.

Per ulteriori informazioni sul provider del servizio IBM MQ , consultare Utilizzo del provider del servizio IBM MQ.

IBM Integration Bus

IBM Integration Bus è la principale tecnologia di integrazione di IBMche può essere utilizzata per collegare applicazioni e sistemi, indipendentemente dai protocolli e dai formati dei messaggi supportati.

IBM Integration Bus ha sempre supportato IBM MQ e fornisce nodi *HTTPInput* e *HTTPRequest* che possono essere utilizzati per creare un'interfaccia RESTful su IBM MQe molti altri sistemi come i database.

IBM Integration Bus può essere utilizzato per fare molto di più che fornire una semplice interfaccia REST su IBM MQ. Le sue funzionalità possono essere utilizzate per fornire la manipolazione del payload avanzato, l'arricchimento del payload e molti altri miglioramenti come parte di REST API.

Per ulteriori informazioni, consultare l' esempio di tecnologia che espone un'interfaccia JSON su REST su un'applicazione IBM MQ che prevede un payload XML.

DataPower

Il gateway DataPower è un singolo gateway multicanale che consente di fornire sicurezza, controllo, integrazione e accesso ottimizzato a una gamma di sistemi, incluso IBM MQ. Viene fornito in entrambi i fattori di forma hardware e virtuale.

Uno dei servizi forniti da DataPower è un gateway multi - protocollo che può prendere input in un protocollo e generare output in un protocollo diverso. In particolare, DataPower può essere configurato per accettare dati HTTP(S) e instradarli a IBM MQ su una connessione client, che può essere utilizzata per creare un'interfaccia REST su IBM MQ. Altri servizi DataPower come la trasformazione possono essere utilizzati anche per migliorare l'interfaccia REST.

Per ulteriori informazioni, vedi Multi - Protocol Gateway.

Messaggistica mediante REST API

È possibile utilizzare messaging REST API per eseguire semplici operazioni point-to-point e pubblicare la messaggistica. È possibile pubblicare messaggi in un argomento, inviare messaggi a una coda, sfogliare i messaggi su una coda e ottenere in maniera distruttiva i messaggi da una coda. Le informazioni vengono inviate e ricevute da messaging REST API in formato testo semplice.

Prima di iniziare

Nota:

- messaging REST API è abilitato per impostazione predefinita. È possibile disabilitare messaging REST API per impedire tutta la messaggistica. Per ulteriori informazioni sull'abilitazione o la disabilitazione di messaging REST API, consultare Configurazione di messaging REST API.
- messaging REST API è integrato con la sicurezza IBM MQ . Per utilizzare messaging REST API, gli utenti devono essere autenticati sul server mqweb e devono essere un membro del ruolo MQWebUser . L'utente deve anche essere autorizzato ad accedere alla coda o all'argomento specificati. Per ulteriori informazioni sulla sicurezza di REST API, consultare Sicurezza di IBM MQ Console e REST API.
- Se si utilizza Advanced Message Security (AMS) con messaging REST API, tenere presente che tutti i messaggi vengono crittografati utilizzando il contesto del server mqweb, non il contesto dell'utente che pubblica il messaggio.
- Quando si riceve o si sfoglia un messaggio, sono supportati solo i messaggi formattati IBM MQ `MQSTR` o `JMS TextMessage` . Successivamente, tutti i messaggi vengono ricevuti in modo distruttivo nel punto di sincronizzazione e tutti i messaggi non gestiti vengono lasciati nella coda. La coda IBM MQ può essere configurata per spostare questi messaggi non elaborabili in una destinazione alternativa. Per ulteriori informazioni, fare riferimento a “Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236.
- Il messaging REST API non fornisce una sola volta la consegna di messaggi con supporto transazionale. Se viene emesso un POST HTTP e la connessione ha esito negativo prima che il client riceva una risposta HTTP , il client non può immediatamente stabilire se il messaggio è stato inviato alla coda specificata o pubblicato nell'argomento specificato. Se viene emesso un comando HTTP DELETE e la connessione ha esito negativo prima che il client riceva una risposta HTTP , è possibile che un messaggio sia stato ricevuto in modo distruttivo dalla coda e che sia stato perso, poiché non è possibile eseguire il rollback del comando di ricezione distruttivo.
- V 9.3.0 Da IBM MQ 9.3.0, le nuove righe nelle stringhe in entrata non sono più rimosse dall'operazione POST HTTP . Le applicazioni REST che utilizzano versioni precedenti non devono utilizzare le nuove righe nei messaggi inviati o pubblicati utilizzando l'API REST, poiché andranno perse.

Procedura

- “Introduzione a messaging REST API” a pagina 705
- “Utilizzo di messaging REST API” a pagina 708
- REST API gestione degli errori
- REST API rilevamento
- REST API supporto lingua nazionale

Riferimenti correlati

Riferimento REST API di messaggistica

Informazioni correlate

Esercitazione: Introduzione alla messaggistica IBM MQ REST API

Introduzione a messaging REST API

Inizia rapidamente con messaging REST API e prova alcuni comandi di esempio utilizzando cURL.

Prima di iniziare

Per iniziare a utilizzare messaging REST API, gli esempi in questa attività hanno i requisiti seguenti:

- Gli esempi utilizzano cURL per inviare richieste REST per inserire e richiamare messaggi da una coda. Pertanto, per completare questa attività è necessario che cURL sia installato sul sistema.
- Gli esempi utilizzano un gestore code QM1. Creare un gestore code con lo stesso nome o sostituire un gestore code esistente sul proprio sistema. Il gestore code deve essere sulla stessa macchina del server mqweb.
- Per completare questa attività, è necessario essere un utente con particolari privilegi in modo da poter utilizzare il comando **dspmqweb**:
  - z/OS Su z/OS, è necessario disporre dell'autorità per eseguire il comando **dspmqweb** e l'accesso in scrittura al file mqwebuser.xml.
  - Multi Su tutti gli altri sistemi operativi, è necessario essere un utente privilegiato.

IBM i Su IBM i, i comandi devono essere in esecuzione in QSHELL.

Procedura

1. Assicurarsi che il server mqweb sia configurato per messaging REST API:
   - Verificare di aver configurato il server mqweb per l'utilizzo da parte di administrative REST API, administrative REST API per MFT, messaging REST APIo IBM MQ Console.Per ulteriori informazioni sulla configurazione del server mqweb con un registro di base, consultare Configurazione di base per il server mqweb.
   - Se il server mqweb è già configurato, assicurarsi di aver aggiunto gli utenti appropriati per abilitare la messaggistica nel passo 5 di Configurazione di base per il server mqweb.
     - Se **mqRestMessagingAdoptWebUserContext** è impostato su true nella configurazione del server mqweb, gli utenti di messaging REST API devono essere membri del ruolo MQWebUser . I ruoli MQWebAdmin e MQWebAdminRO non sono applicabili per messaging REST API. Gli utenti devono anche essere autorizzati ad accedere alle code e agli argomenti utilizzati per la messaggistica tramite OAM o RACF.
     - Se **mqRestMessagingAdoptWebUserContext** è impostato su false nella configurazione del server mqweb, l'ID utente utilizzato per avviare il server mqweb deve essere autorizzato ad accedere alle code utilizzate per la messaggistica tramite OAM o RACF.
2. z/OS
   Su z/OS, impostare la variabile di ambiente WLP_USER_DIR in modo da poter utilizzare il comando **dspmqweb** . Impostare la variabile in modo che punti alla configurazione del server mqweb immettendo il seguente comando:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   dove *WLP_user_directory* è il nome della directory passata a crtmqweb. Ad esempio:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   Per ulteriori informazioni, consultare Creazione del server mqweb.
3. Determinare l'URL REST API immettendo il seguente comando:

   ```
   dspmqweb status
   ```

   Gli esempi riportati di seguito presuppongono che il tuo REST API URL sia l' URL https://localhost:9443/ibmmq/rest/v2/predefinito. Se l'URL è diverso da quello predefinito, sostituirlo nella seguente procedura.
4. Creare una coda, MSGQ, sul gestore code QM1. Questa coda viene utilizzata per la messaggistica. Utilizza uno dei seguenti metodi:

- Utilizzare una richiesta POST sulla risorsa `mqsc` di administrative REST API, autenticandosi come utente `mqadmin` :

```
curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/
json" --data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\":
\"qlocal\",\"name\": \"MSGQ\"}"
```

- Utilizzare i comandi MQSC:

  z/OS Su z/OS, utilizzare un'origine 2CR invece del comando **runmqsc** . Per ulteriori informazioni, consultare Origini da cui è possibile immettere i comandi MQSC e PCF su IBM MQ for z/OS.

  a. Avviare **runmqsc** per il gestore code immettendo il seguente comando:

  ```
  runmqsc QM1
  ```

  b. Utilizzare il comando **DEFINE QLOCAL** MQSC per creare la coda:

  ```
  DEFINE QLOCAL(MSGQ)
  ```

  c. Uscire da **runmqsc** immettendo il comando seguente:

  ```
  end
  ```

5. Concedere l'autorizzazione all'utente aggiunto al `mqwebuser.xml` nel passo 5 di Configurazione base per il server mqweb per accedere alla coda MSGQ. Sostituire l'utente in cui viene utilizzato `myuser` :

- z/OS Su z/OS:

  a. Concedere l'accesso utente alla coda:

  ```
  RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
  PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
  ```

  b. Concedere all'ID utente dell'attività avviata mqweb l'accesso per impostare tutto il contesto sulla coda:

  ```
  RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
  PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
  ```

- Multi Su tutti gli altri sistemi operativi, se l'utente si trova nel gruppo mqm, l'autorità è già concessa. Altrimenti, immettere i seguenti comandi:

  a. Avviare **runmqsc** per il gestore code immettendo il seguente comando:

  ```
  runmqsc QM1
  ```

  b. Utilizzare il comando **SET AUTHREC** MQSC per fornire all'utente le autorizzazioni di ricerca, interrogazione, acquisizione e inserimento sulla coda:

  ```
  SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
  PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
  ```

  c. Uscire da **runmqsc** immettendo il comando seguente:

  ```
  end
  ```

6. Inserire un messaggio con il contenuto `Hello World!` nella coda MSGQ sul gestore code QM1, utilizzando una richiesta POST sulla risorsa `message` . Sostituire l'ID utente e la password da `mqwebuser.xml` per `myuser` e `mypassword`:

Viene utilizzata l'autenticazione base e viene impostata un'intestazione `ibm-mq-rest-csrf-token` HTTP con un valore arbitrario nella richiesta REST cURL . Questa intestazione aggiuntiva è richiesta per le richieste POST, PATCH e DELETE.

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X
POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
plain;charset=utf-8" --data "Hello World!"
```

7. Richiamare in modo distruttivo il messaggio dalla coda `Hello World!` sulla coda `MSGQ` sul gestore code `QM1`, utilizzando una richiesta DELETE sulla risorsa `message` . Sostituire l'ID utente e la password da `mqwebuser.xml` per `myuser` e `mypassword`:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
```

Viene restituito il messaggio `Hello World!` .

Operazioni successive

- Gli esempi utilizzano l'autenticazione di base per proteggere la richiesta. È possibile utilizzare invece l'autenticazione basata su token o sul client. Per ulteriori informazioni, consultare Utilizzo dell'autenticazione del certificato client con la REST API e la IBM MQ Console e Utilizzo dell'autenticazione basata su token con la REST API.
- Ulteriori informazioni sull'utilizzo di messaging REST API e sulla creazione di URL con parametri di query: “Utilizzo di messaging REST API” a pagina 708.
- Quando si utilizza messaging REST API, le connessioni al gestore code vengono raggruppate per ottimizzare le prestazioni. È possibile impostare la dimensione massima del pool e quale azione eseguire quando tutte le connessioni nel pool sono in uso: Configurazione di messaging REST API.
- Sfogliare le informazioni di riferimento per le risorse messaging REST API disponibili e tutti i parametri di query disponibili: messaging REST API reference.
- Rilevare administrative REST API, un'interfaccia RESTful per l'amministrazione IBM MQ : Administration utilizzando REST API.
- Rilevare IBM MQ Console, una GUI basata su browser: Administration utilizzando IBM MQ Console.

Utilizzo di messaging REST API

Quando si utilizza messaging REST API, si richiamano metodi HTTP sugli URL per inviare e ricevere messaggi IBM MQ . Il metodo HTTP , ad esempio POST, rappresenta il tipo di azione da eseguire sull'oggetto rappresentato dall' URL. Ulteriori informazioni sull'azione potrebbero essere codificate nei parametri della query. Le informazioni sul risultato dell'esecuzione dell'azione potrebbero essere restituite come corpo della risposta HTTP .

Prima di iniziare

Considerare quanto segue prima di utilizzare messaging REST API:

- È necessario eseguire l'autenticazione con il server mqweb per utilizzare messaging REST API. È possibile eseguire l'autenticazione utilizzando l'autenticazione di base HTTP , l'autenticazione del certificato client o l'autenticazione basata sul token. Per ulteriori informazioni su come utilizzare questi metodi di autenticazione, consultare SicurezzaIBM MQ Console e REST API.
- REST API è sensibile al maiuscolo / minuscolo. Ad esempio, un POST HTTP sul seguente URL genera un errore se il gestore code è denominato `qmgr1`.

```
/ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
```

- V 9.3.3 Se si sta effettuando la connessione a un gestore code remoto con messaging REST API, è necessario utilizzare il nome univoco per la connessione del gestore code invece del nome del gestore code.

- Non tutti i caratteri che possono essere utilizzati nei nomi oggetto IBM MQ possono essere codificati direttamente in un URL. Per codificare correttamente questi caratteri, è necessario utilizzare la codifica URL appropriata:
  - Una barra deve essere codificata come %2F.
  - Un segno percentuale deve essere codificato come %25.
  - Un punto deve essere codificato come %2E.
  - Un punto interrogativo deve essere codificato come %3F.
- Quando si riceve o si sfoglia un messaggio, sono supportati solo i messaggi formattati IBM MQ MQSTR e `JMS TextMessage` . Successivamente, tutti i messaggi vengono ricevuti in modo distruttivo nel punto di sincronizzazione e tutti i messaggi non gestiti vengono lasciati nella coda. La coda IBM MQ può essere configurata per spostare questi messaggi non elaborabili in una destinazione alternativa. Per ulteriori informazioni, fare riferimento a “Gestione dei messaggi non elaborabili in IBM MQ classes for JMS” a pagina 236.

Informazioni su questa attività

Quando si utilizza REST API per eseguire un'azione di messaggistica su un oggetto coda IBM MQ , è necessario prima creare un URL per rappresentare tale oggetto. Ogni URL inizia con un prefisso, che descrive a quali nome host e porta inviare la richiesta. Il resto dell' URL descrive un particolare oggetto o un instradamento a tale oggetto, noto come risorsa.

L'azione di messaggistica che deve essere eseguita sulla risorsa definisce se l' URL ha bisogno o meno di parametri di query. Definisce inoltre il metodo HTTP utilizzato e se ulteriori informazioni vengono inviate all' URLo restituite da esso. Le informazioni aggiuntive possono far parte della richiesta HTTP o essere restituite come parte della risposta HTTP .

Dopo aver costruito l' URL, puoi inviare la richiesta HTTP a IBM MQ. È possibile inviare la richiesta utilizzando l'implementazione HTTP integrata nel linguaggio di programmazione scelto. Puoi anche inviare la richiesta utilizzando strumenti della riga di comando come cURL, un browser web o un componente aggiuntivo del browser web.

Importante: È necessario, come minimo, eseguire le operazioni “1.a” a pagina 709 e “1.b” a pagina 709.

Procedura

1. Crea l' URL:
   a) Determinare il prefisso URL immettendo il seguente comando:

   ```
   dspmqweb status
   ```

   L' URL che vuoi utilizzare include la frase `/ibmmq/rest/` .
   b) Aggiungere la coda e le risorse del gestore code associate da utilizzare per la messaggistica al percorso URL .

   Nel riferimento di messaggistica, i segmenti di variabile possono essere identificati nell' URL dalle parentesi che lo circondano *{ }*. Per ulteriori informazioni, consultare /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

   Ad esempio, per interagire con la coda *Q1* associata con il gestore code *QM1*, aggiungere `/qmgr` e `/queue` al prefisso URL per creare il seguente URL:

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
   ```

   Suggerimento: V 9.3.3 Se il gestore code è un gestore code remoto, è necessario utilizzare il nome univoco del gestore code al posto del nome del gestore code. Il gestore code remoto deve essere configurato prima di poter essere utilizzato con messaging REST API. Per ulteriori

informazioni, consultare “Configurazione di un gestore code remoto da utilizzare con messaging REST API” a pagina 710.

c) Opzionale: Aggiungere un parametro di query facoltativo all' URL.

Aggiungere un punto interrogativo,?, parametro query, segno uguale = e un valore per l' URL.

Ad esempio, per attendere un massimo di 30 secondi che il messaggio successivo diventi disponibile, creare il seguente URL:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) Opzionale: Aggiungere ulteriori parametri di query facoltativi all' URL.

Aggiungere una e commerciale, &, all' URLe ripetere il passo 1c.

2. Richiama il metodo HTTP pertinente sull' URL. Specificare qualsiasi payload del messaggio facoltativo e fornire le credenziali di sicurezza appropriate per l'autenticazione. Ad esempio:
   - Utilizzare l'implementazione HTTP/REST del linguaggio di programmazione scelto.
   - Utilizzare uno strumento come un componente aggiuntivo del browser del client REST o cURL.

V 9.3.3 Configurazione di un gestore code remoto da utilizzare con messaging REST API

È possibile utilizzare messaging REST API per connettersi ai gestori code remoti per la messaggistica. Prima di potersi connettere a un gestore code remoto, è necessario impostare la configurazione del gestore code remoto. Quindi, è possibile connettersi al gestore code remoto utilizzando il nome univoco definito nelle informazioni di configurazione.

Prima di iniziare

- Verificare di aver configurato il server mqweb per l'utilizzo da parte di administrative REST API, administrative REST API per MFT, messaging REST APIo IBM MQ Console.Per ulteriori informazioni sulla configurazione del server mqweb con un registro di base, consultare Configurazione di base per il server mqweb.
- Se il server mqweb è già configurato, assicurarsi di aver aggiunto gli utenti appropriati per abilitare la messaggistica al passo 5 di Configurazione base per il server mqweb. Gli utenti di messaging REST API devono essere membri del ruolo `MQWebUser` . I ruoli `MQWebAdmin` e `MQWebAdminRO` non sono applicabili per messaging REST API.
  - Se **`mqRestMessagingAdoptWebUserContext`** è impostato su `true` nella configurazione del server mqweb, gli utenti nel ruolo `MQWebUser` devono essere autorizzati ad accedere alle code e agli argomenti utilizzati per la messaggistica tramite OAM o RACF.
  - Se **`mqRestMessagingAdoptWebUserContext`** è impostato su `false` nella configurazione del server mqweb, l'ID utente utilizzato per avviare il server mqweb deve essere autorizzato ad accedere alle code e agli argomenti utilizzati per la messaggistica tramite OAM o RACF.
- Verificare che messaging REST API sia configurato per connettersi ai gestori code remoti. Per ulteriori informazioni, consultare Configurazione della modalità di connessione per messaging REST API.

Informazioni su questa attività

È possibile connettersi ai gestori code remoti utilizzando messaging REST API. Un gestore code remoto può essere un gestore code su un altro sistema, un gestore code in un'altra installazione o un gestore code nella stessa installazione del server mqweb.

Per connettersi a un gestore code remoto, è necessario effettuare le seguenti operazioni di configurazione:

- Configurare un canale di connessione server e un listener.
- Concedere l'autorizzazione a un utente appropriato per accedere al gestore code.

- Creare un file CCDT che contenga le informazioni di connessione per il gestore code.
- Aggiungere le informazioni di connessione a messaging REST API utilizzando il comando **setmqweb remote** .

Quindi, è possibile utilizzare il gestore code remoto fornendo il nome univoco nella risorsa URL al posto del nome del gestore code.

È inoltre possibile configurare i gestori code remoti come parte di un gruppo di gestori code. Per ulteriori informazioni, consultare “Impostazione di un gruppo di gestori code da utilizzare con l'API REST di messaggistica” a pagina 713.

Procedura

1. Sul gestore code remoto, creare un canale di connessione server per consentire connessioni remote al gestore code. È possibile creare canali di connessione server utilizzando il comando **DEFINE CHANNEL** MQSC sulla riga comandi.
   Ad esempio, per creare un canale di connessione server QM1.SVRCONN per il gestore code QM1, immettere il seguente comando:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Per ulteriori informazioni su **DEFINE CHANNEL** e sulle opzioni disponibili, consultare DEFINE CHANNEL.
2. Verificare che un utente appropriato sia autorizzato ad accedere al gestore code. Questo utente deve anche essere autorizzato ad accedere a qualsiasi coda o argomento utilizzato per la messaggistica. L'utente necessita dell'autorità connect, inquire, alternate usere set context sul gestore code. Su UNIX, Linux, and Windows , utilizzare il comando di controllo **setmqaut** sulla riga comandi. Su z/OS, definire i profili RACF per fornire all'utente autorizzato l'accesso al gestore code.
   Ad esempio, in UNIX, Linux, and Windows, per autorizzare un utente, exampleUser, ad accedere al gestore code QM1, immettere il seguente comando:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Per ulteriori informazioni su quale utente deve essere autorizzato, consultare “Determinazione del principal di sicurezza utilizzato da messaging REST API” a pagina 716.
3. ALW
   Se non esiste alcun listener sul gestore code remoto, creare un listener per accettare le connessioni di rete in entrata utilizzando il comando MQSC **DEFINE LISTENER** sulla riga comandi.
   Ad esempio, per creare un listener REMOTE.LISTENER sulla porta 1414 per il gestore code QM1, immettere il seguente comando:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. Verificare che il listener sia in esecuzione utilizzando il comando MQSC **START LISTENER** sulla riga comandi:

   ALW Ad esempio, su AIX, Linux, and Windows per avviare il listener REMOTE.LISTENER per il gestore code QM1, immettere il seguente comando:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS Ad esempio, su z/OS, per avviare il listener, immettere il comando seguente:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

Lo spazio di indirizzo dell'iniziatore di canali deve essere avviato prima di poter avviare un listener su z/OS.

5. Sul sistema su cui è in esecuzione il server mqweb che ospita messaging REST API , creare o aggiornare un file CCDT JSON che contiene le informazioni di connessione del gestore code.

   Il file CCDT deve includere le informazioni `name`, `clientConnectione` `type` . Facoltativamente, è possibile includere ulteriori informazioni come le informazioni `transmissionSecurity` . Per ulteriori informazioni su tutte le definizioni di attributo del canale CCDT, consultare Elenco completo delle definizioni di attributo del canale CCDT.

   Il seguente esempio mostra un file CCDT JSON di base per una connessione del gestore code remoto. Imposta il nome del canale sullo stesso nome del canale di connessione server di esempio creato nel passaggio “1” a pagina 711. La porta di collegamento è impostata sullo stesso valore della porta utilizzata dal listener. L'host di connessione è impostato sul nome host del sistema su cui è in esecuzione il gestore code remoto QM1.

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

6. Dall'installazione che esegue il server mqweb su cui è presente messaging REST API, utilizzare il comando **setmqweb remote** per aggiungere le informazioni sul gestore code remoto alla configurazione del server mqweb.

   Come minimo, è necessario specificare i parametri seguenti:

   - **-qmgrName**, dove si specifica il nome del gestore code.
   - **-ccdtURL**, dove si specifica l' URL CCDT per il gestore code.
   - **-uniqueName**, in cui si specifica un nome univoco per il gestore code. Il nome univoco viene utilizzato per differenziare i gestori code remoti che potrebbero avere lo stesso nome e pertanto non devono esistere per identificare un altro gestore code.

   È possibile specificare diverse altre opzioni, come il nome utente e la password da utilizzare per la connessione al gestore code remoto o i dettagli del truststore e del keystore. Per un elenco completo di parametri che possono essere specificati con il comando **setmqweb remote** , consultare la sezione setmqweb remote.

   Ad esempio, per aggiungere il gestore code remoto QM1, con il file CCDT di esempio, immettere il seguente comando:

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

Risultati

Il gestore code remoto può essere utilizzato con messaging REST API utilizzando il nome univoco nella risorsa URL invece del nome del gestore code.

Esempio

Il seguente esempio imposta la connessione del gestore code remoto per il gestore code QM1. IBM MQ Console autorizzato a gestire il gestore code in base all'autorizzazione concessa all'utente exampleUser. Le credenziali di questo utente vengono fornite a IBM MQ Console quando **setmqweb remote** viene utilizzato per configurare la connessione al gestore code.

1. Sul sistema in cui si trova il gestore code remoto QM1 , vengono creati un canale di connessione server e un listener. Il listener viene avviato e viene concessa l'autorizzazione all'utente `exampleUser` per connettersi al gestore code e accedere ad una coda utilizzata per la messaggistica:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. Sul sistema su cui è in esecuzione il server mqweb, viene creato un file `QM1_ccdt.json` con le seguenti informazioni di connessione:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. Sul sistema su cui è in esecuzione il server mqweb, le informazioni di collegamento per il gestore code QM1 vengono aggiunte al server mqweb. Le credenziali per `exampleUser` sono incluse nelle informazioni di connessione:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. messaging REST API può connettersi al gestore code remoto QM1 utilizzando il nome univoco per la connessione del gestore code al posto del nome del gestore code nella risorsa URL:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:
text/plain;charset=utf-8" --data "Hello World!"
```

V 9.3.3 Impostazione di un gruppo di gestori code da utilizzare con l'API REST di messaggistica

È possibile utilizzare messaging REST API per connettersi ai gruppi di gestori code per la messaggistica. Prima di potersi connettere a un gruppo di gestori code, è necessario impostare la configurazione del gestore code remoto per il gruppo. Quindi, è possibile connettersi al gruppo di gestori code utilizzando il nome univoco definito nelle informazioni di configurazione.

Prima di iniziare

- Verificare di aver configurato il server mqweb per l'utilizzo da parte di administrative REST API, administrative REST API per MFT, messaging REST APIo IBM MQ Console.Per ulteriori informazioni sulla configurazione del server mqweb con un registro di base, consultare Configurazione di base per il server mqweb.
- Se il server mqweb è già configurato, assicurarsi di aver aggiunto gli utenti appropriati per abilitare la messaggistica al passo 5 di Configurazione base per il server mqweb. Gli utenti di messaging REST

API devono essere membri del ruolo `MQWebUser` . I ruoli `MQWebAdmin` e `MQWebAdminRO` non sono applicabili per messaging REST API.

  - Se **`mqRestMessagingAdoptWebUserContext`** è impostato su `true` nella configurazione del server mqweb, gli utenti con il ruolo `MQWebUser` devono essere autorizzati ad accedere alle code e agli argomenti utilizzati per la messaggistica. È possibile autorizzare questi utenti tramite OAM o RACF.
  - Se **`mqRestMessagingAdoptWebUserContext`** è impostato su `false` nella configurazione del server mqweb, l'ID utente che avvia il server mqweb deve essere autorizzato ad accedere alle code e agli argomenti utilizzati per la messaggistica. È possibile autorizzare questo utente tramite OAM o RACF.

- Verificare che messaging REST API sia configurato per connettersi ai gestori code remoti. Per ulteriori informazioni, consultare Configurazione della modalità di connessione per messaging REST API

Informazioni su questa attività

Un gruppo di gestori code consente di connettere le applicazioni a qualsiasi gestore code all'interno del gruppo. Il gruppo è definito come una serie di connessioni in una tabella di definizione di canale client (CCDT). Quando si utilizza una chiamata MQCONN o MQCONNX, si fa riferimento al gruppo prefissando un asterisco al nome del gestore code. Con messaging REST API, si fa riferimento al gruppo utilizzando il nome univoco che è associato al gruppo di gestori code. Il nome univoco è incluso nella risorsa URL al posto del nome del gestore code. Per ulteriori informazioni sui gruppi di gestori code, consultare “Gruppi di gestori code in CCDT” a pagina 925.

È anche possibile configurare i gestori code remoti singolarmente. Per ulteriori informazioni, consultare “Configurazione di un gestore code remoto da utilizzare con messaging REST API” a pagina 710.

Procedura

1. Su ogni gestore code remoto del gruppo, creare un canale di connessione server per consentire connessioni remote al gestore code. È possibile utilizzare il comando **`DEFINE CHANNEL`** MQSC sulla riga comandi per creare canali di connessione server.
   Ad esempio, per creare un canale di connessione server `QM1.SVRCONN` per il gestore code QM1, immettere il seguente comando:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Per ulteriori informazioni su **`DEFINE CHANNEL`** e sulle opzioni disponibili, consultare DEFINE CHANNEL.
2. Su ciascuno dei gestori code remoti nel gruppo, verificare che un utente appropriato sia autorizzato ad accedere al gestore code. Questo utente deve anche essere autorizzato ad accedere a qualsiasi coda o argomento utilizzato per la messaggistica. L'utente necessita dell'autorità `connect`, `inquire`, `alternate usere set context` sul gestore code. Su UNIX, Linux, and Windows , utilizzare il comando di controllo **`setmqaut`** sulla riga comandi. Su z/OS, definire i profili RACF per fornire all'utente autorizzato l'accesso al gestore code.
   Ad esempio, in UNIX, Linux, and Windows, immettere il seguente comando per autorizzare un utente, `exampleUser`, ad accedere al gestore code QM1:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Per ulteriori informazioni su quale utente deve essere autorizzato, consultare “Determinazione del principal di sicurezza utilizzato da messaging REST API” a pagina 716.
3. ALW
   Se non esiste alcun listener su ciascuno dei gestori code remoti nel gruppo, creare i listener per accettare le connessioni di rete in entrata. È possibile utilizzare il comando **`DEFINE LISTENER`** MQSC sulla linea di comandi per creare listener.

Ad esempio, per creare un listener REMOTE.LISTENER sulla porta 1414 per il gestore code QM1, immettere il seguente comando:

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

4. Su ciascuno dei gestori code remoti nel gruppo, assicurarsi che il listener sia in esecuzione utilizzando il comando MQSC **START LISTENER** sulla riga comandi.

   ALW Ad esempio, su AIX, Linux, and Windows per avviare il listener REMOTE.LISTENER per il gestore code QM1, immettere il seguente comando:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS Ad esempio, su z/OS, per avviare il listener, immettere il comando seguente:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

   Lo spazio di indirizzo dell'iniziatore di canali deve essere avviato prima di poter avviare un listener su z/OS.
5. Sul sistema su cui è in esecuzione il server mqweb che ospita messaging REST API , creare un file CCDT JSON. Questo file JSON contiene informazioni di connessione per ogni gestore code nel gruppo.

   Il file CCDT deve contenere le informazioni `name`, `clientConnection`e `type` per ogni connessione del gestore code. Facoltativamente, è possibile includere ulteriori informazioni come le informazioni `transmissionSecurity` . Per ulteriori informazioni su tutte le definizioni di attributo del canale CCDT, consultare Elenco completo delle definizioni di attributo del canale CCDT.

   Il seguente esempio mostra un file CCDT JSON di base per due connessioni del gestore code. La prima connessione è per il gestore code QM1. Ha un canale di connessione server di `QM1.SVRCONN`, un listener sulla porta 1414e viene eseguito sull'host `QM1.example.com`. La seconda connessione è per il gestore code QM2. Ha un canale di connessione server di `QM2.SVRCONN`, un listener sulla porta 1415e viene eseguito sull'host `QM2.example.com`. Tuttavia, poiché le connessioni fanno parte del gruppo QMGRP del gestore code, il campo **queueManager** per entrambe le connessioni è impostato sul nome del gruppo del gestore code.

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "QM1.example.com",
               "port": 1414
             }],
           "queueManager": "QMGRP"
         },
         "type": "clientConnection"
      }],
      "channel": [{
         "name": "QM2.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "QM2.example.com",
               "port": 1415
             }],
           "queueManager": "QMGRP"
         },
         "type": "clientConnection"
       }]
   }
   ```
6. Dall'installazione che esegue il server mqweb su cui è presente messaging REST API, utilizzare il comando **setmqweb remote** per aggiungere il gruppo di gestori code alla configurazione del server mqweb.

Come minimo, è necessario specificare i parametri seguenti:

- **-qmgrName**, dove si specifica il nome del gruppo per il gruppo di gestori code.
- **-ccdtURL**, dove si specifica l' URL CCDT per i gestori code.
- **-uniqueName**, in cui si specifica un nome univoco per identificare il gruppo di gestori code.
- **-group**, per impostare le informazioni sul gestore code come per un gruppo.

È possibile specificare diverse altre opzioni, come il nome utente e la parola d'ordine da utilizzare per la connessione o i dettagli del truststore e del keystore. Per un elenco completo di parametri che possono essere specificati con il comando **setmqweb remote** , consultare la sezione setmqweb remote.

Ad esempio, per aggiungere il gruppo gestore code QMGRP, con il file CCDT di esempio, immettere il seguente comando:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

Risultati

Il gruppo gestore code remoto può essere utilizzato con messaging REST API utilizzando il nome univoco nella risorsa URL. Un gestore code del gruppo è selezionato per completare la richiesta e le informazioni su quale gestore code ha completato la richiesta vengono restituite nell'intestazione della risposta `ibm-mq-resolved-qmgr`.

V 9.3.3 Determinazione del principal di sicurezza utilizzato da messaging REST API

Quando si utilizza messaging REST API, è necessario che un utente appropriato sia autorizzato ad accedere ai gestori code, alle code e agli argomenti a cui si desidera connettersi per la messaggistica. L'utente che deve essere autorizzato dipende da come è configurato il server mqweb e se si stanno utilizzando gestori code remoti con messaging REST API.

Per impostazione predefinita, il principal di protezione utilizzato per autorizzare l'accesso al gestore code è l'utente che avvia il server mqweb che esegue messaging REST API. Il principal di sicurezza utilizzato per autorizzare l'accesso alle code e agli argomenti è l'utente collegato a messaging REST API. Tuttavia, la connessione del server mqweb o del gestore code remoto potrebbe essere configurata in modo da utilizzare un principal di sicurezza differente.

Determinazione del principal di sicurezza utilizzato per la connessione al gestore code

Per le connessioni del gestore code locale, il principal di sicurezza utilizzato per la connessione al gestore code è l'utente che avvia il server mqweb che esegue messaging REST API. Per le connessioni del gestore code remoto, i seguenti principal di sicurezza vengono utilizzati da messaging REST API per autorizzare l'accesso al gestore code, in ordine di priorità. Ovvero, se gli utenti vengono specificati in più modi all'interno della configurazione del gestore code remoto, il primo nell'elenco viene utilizzato per l'autorizzazione.

1. Il principal di sicurezza è un contesto utente adottato da un'uscita di sicurezza.
2. Il principal della sicurezza è un contesto utente adottato in una regola CHLAUTH sul canale di connessione server utilizzato per connettersi al gestore code remoto.
3. Il principal di sicurezza è l'ID utente incluso nella configurazione del gestore code remoto per messaging REST API. Questo ID utente è facoltativamente incluso nelle informazioni di connessione del gestore code quando si aggiunge il gestore code con il comando **setmqweb remote** .
4. Il principal di sicurezza è l'utente che avvia il server mqweb che esegue messaging REST API.

Per ulteriori informazioni sull'impostazione dei gestori code remoti da utilizzare con messaging REST API, consultare “Configurazione di un gestore code remoto da utilizzare con messaging REST API” a pagina 710.

Determinazione del principal di sicurezza utilizzato per connettersi a code e argomenti

È possibile impostare una proprietà nella configurazione del server mqweb per stabilire quale principal di sicurezza viene utilizzato per autorizzare connessioni a code e argomenti quando si utilizza messaging REST API. Questa proprietà è la proprietà **mqRestMessagingAdoptWebUserContext** . È possibile visualizzare le proprietà impostate utilizzando il comando **dspmqweb properties** .

- Se **mqRestMessagingAdoptWebUserContext** è impostato su true, messaging REST API utilizza l'ID utente dell'utente collegato a messaging REST API per l'autorizzazione. Pertanto, l'ID utente o gli ID utente che esistono nella configurazione del server mqweb da utilizzare con messaging REST API sono i principal di sicurezza che devono essere autorizzati ad accedere alle code e agli argomenti.
- Se **mqRestMessagingAdoptWebUserContext** è impostato su false, messaging REST API utilizza l'ID utente dell'utente che ha avviato il server mqweb su cui è presente messaging REST API per l'autorizzazione. Pertanto, un ID utente che è lo stesso dell'ID utente che avvia il server mqweb che ospita messaging REST API deve essere autorizzato ad accedere alle code e agli argomenti.

  Se le code e gli argomenti si trovano su un gestore code remoto, il principal di sicurezza utilizzato per l'autorizzazione potrebbe essere determinato dalle impostazioni nella configurazione del gestore code. I seguenti principal di sicurezza potrebbero essere utilizzati, in ordine di priorità:

  1. Il principal di sicurezza è un contesto utente adottato da un'uscita di sicurezza.
  2. Il principal della sicurezza è un contesto utente adottato in una regola CHLAUTH sul canale di connessione server utilizzato per connettersi al gestore code remoto. Ad esempio, è possibile configurare una regola CHLAUTH sul canale di connessione server per utilizzare il parametro MCAUSER. Quindi, tutte le connessioni vengono associate a un ID utente autorizzato a utilizzare il gestore code.
  3. Il principal di sicurezza è un contesto utente adottato da AUTHINFO del gestore code. Se l'oggetto AUTHINFO a cui fa riferimento l'attributo CONNAUTH del gestore code è configurato per utilizzare **ADOPTCTX(yes)**, il principal di sicurezza utilizzato per autorizzare le connessioni al gestore code viene utilizzato anche per autorizzare le code e gli argomenti. Ad esempio, questo principal di sicurezza potrebbe essere l'ID utente incluso nelle informazioni di connessione del gestore code remoto come parte del comando **setmqweb remote** .

Informazioni correlate

CHLAUTH

CONNAUTH

proprietà dspmqweb

Sviluppo di applicazioni MQI con IBM MQ

IBM MQ fornisce il supporto per C, Visual Basic, COBOL, Assembler, RPG, pTALe PL/I. Questi linguaggi procedurali utilizzano l'interfaccia MQI (Message Queue Interface) per accedere ai servizi di accodamento messaggi.

Per informazioni dettagliate su come scrivere le applicazioni nella lingua scelta, consultare i topic secondari.

Per una panoramica dell'interfaccia di chiamata per le lingue procedurali, consultare Descrizioni delle chiamate. Questo argomento contiene un elenco delle chiamate MQI e ciascuna chiamata mostra come codificare le chiamate in ognuna di queste lingue.

IBM MQ fornisce file di definizione dei dati che consentono di scrivere le applicazioni. Per una descrizione completa, consultare “File di definizione dati IBM MQ” a pagina 718.

Per aiutare a scegliere il linguaggio procedurale in cui codificare i programmi, considerare la lunghezza massima dei messaggi che i programmi elaboreranno. Se i programmi elaboreranno solo messaggi di lunghezza massima nota, è possibile codificarli in una qualsiasi delle lingue supportate. Se non si conosce la lunghezza massima dei messaggi che i programmi dovranno elaborare, la lingua scelta dipenderà dalla scrittura di un'applicazione CICS, IMSo batch:

IMS e batch
Codificare i programmi in linguaggio C, PL/I o assembler per utilizzare le funzioni offerte da questi linguaggi per ottenere e rilasciare quantità arbitrarie di memoria. In alternativa, è possibile codificare i programmi in COBOL, ma utilizzare il linguaggio assembler, le sottoroutine PL/I o C per ottenere e rilasciare la memoria.

CICS
Codificare i programmi in qualsiasi lingua supportata da CICS. L'interfaccia EXEC CICS fornisce le chiamate per la gestione della memoria, se necessario.

Concetti correlati

“Applicazioni orientate agli oggetti” a pagina 16
IBM MQ fornisce supporto per JMS, Java, C + + e .NET. Questi linguaggi e framework utilizzano IBM MQ Object Model, che fornisce classi che forniscono la stessa funzionalità delle chiamate e delle strutture IBM MQ .

Panoramica tecnica

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

Riferimenti correlati

Riferimento per lo sviluppo di applicazioni

File di definizione dati IBM MQ

IBM MQ fornisce file di definizione dei dati che consentono di scrivere le applicazioni.

I file di definizione dati sono noti anche come:

Lingua	Definizioni di dati
C	Includi file o file di intestazione
Visual Basic	File di modulo (solo versioni a 32 bit)
COBOL	Copia file
Assembler	Macro
PL/I	Includi file

I file di definizione dati che consentono di scrivere le uscite del canale sono descritti in IBM MQ File COPY, intestazione, inclusione e modulo.

I file di definizione dei dati che consentono di scrivere le uscite dei servizi installabili sono descritti in “Uscite utente, uscite API e servizi installabili IBM MQ” a pagina 938.

Per i file di definizione dati supportati su C + +, consultare Utilizzo di C++.

IBM i

Per i file di definizione dati supportati su RPG, consultare IBM i Application Programming Reference (ILE/RPG).

I nomi dei file di definizione dati hanno il prefisso CMQ e un suffisso determinato dal linguaggio di programmazione:

Suffisso	Lingua
a	Linguaggio assembler
b	Visual Basic
c	C
l	COBOL (senza valori inizializzati)
p	PL/I
v	COBOL (con valori predefiniti impostati)

Libreria di installazione

Il nome **thlqual** è il qualificatore di alto livello della libreria di installazione su z/OS.

Questo argomento introduce i file di definizione dati IBM MQ , sotto queste intestazioni:

- "File di inclusione linguaggio C" a pagina 719
- "File del modulo Visual Basic" a pagina 719
- "File di copia COBOL" a pagina 720
- z/OS "Macro linguaggio assembler System/390" a pagina 721
- z/OS "File di inclusione PL/I" a pagina 721

File di inclusione linguaggio C

I file di inclusione in C IBM MQ sono elencati in File di intestazione C. Sono installati nelle seguenti directory o librerie:

Piattaforma	Directory di installazione o libreria
IBM i IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\c\includi
z/OS z/OS	**thlqual**.SCSQC370

dove *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ .

Nota: Per AIX and Linux, i file di inclusione sono simbolicamente collegati in `/usr/include`.

Per ulteriori informazioni sulla struttura delle directory, consultare Pianificazione del supporto del file system.

File del modulo Visual Basic

IBM MQ for Windows fornisce quattro file del modulo Visual Basic.

Sono elencati in File del modulo Visual Basic e installati in

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

File di copia COBOL

Per COBOL, IBM MQ fornisce file di copia separati contenenti le costanti denominate e due file di copia per ogni struttura.

Ci sono due file di copia per ogni struttura perché ognuno è fornito sia con che senza valori iniziali:

- Nella WORKING - STORAGE SECTION di un programma COBOL, utilizzare i file che inizializzano i campi di struttura sui valori predefiniti. Queste strutture sono definite nei file di copia che hanno nomi con suffisso la lettera V (valori).
- Nella LINKAGE SECTION di un programma COBOL, utilizzare le strutture senza valori iniziali. Queste strutture sono definite in file di copia i cui nomi hanno come suffisso la lettera L (collegamento).

IBM i I file di copia contenenti i dati e le definizioni di interfaccia per IBM i sono forniti per i programmi ILE COBOL che utilizzano le chiamate prototipo a MQI. I file esistono in QMQM/QCBLLESRC con nomi membro che hanno un suffisso L (per strutture senza valori iniziali) o un suffisso V (per strutture con valori iniziali).

I file di copia IBM MQ COBOL sono elencati in File COBOL COPY. Sono installati nelle directory seguenti:

Piattaforma	Directory di installazione o libreria
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\Copybook (per Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (per IBM VisualAge COBOL)
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Includere nel programma solo quei file necessari. Eseguire questa operazione con una o più istruzioni COPY dopo una dichiarazione level-01 . Ciò significa che è possibile includere più versioni delle strutture in un programma, se necessario. Notare che CMQV è un file di grandi dimensioni.

Di seguito è riportato un esempio di codice COBOL per includere il file di copia CMQMDV:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

Ogni dichiarazione di struttura inizia con un elemento level-01 ; è possibile dichiarare diverse istanze della struttura codificando la dichiarazione level-01 seguita da un'istruzione COPY da copiare nel resto della dichiarazione di struttura. Per fare riferimento all'istanza appropriata, utilizzare la parola chiave IN.

Di seguito è riportato un esempio di codice COBOL per includere due istanze di CMQMDV:

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

Allineare le strutture sui limiti a 4 byte. Se si utilizza l'istruzione COPY per includere una struttura che segue un elemento che non è l'elemento level-01 , verificare che la struttura sia un multiplo di 4 byte dall'inizio dell'elemento level-01 . Se non si esegue questa operazione, è possibile ridurre le prestazioni dell'applicazione.

Le strutture sono descritte in Tipi di dati utilizzati in MQI. Le descrizioni dei campi nelle strutture mostrano i nomi dei campi senza prefisso. Nei programmi COBOL, anteporre ai nomi dei campi il nome della struttura seguito da un trattino, come mostrato nelle dichiarazioni COBOL. I campi nei file di copia della struttura hanno come prefisso questo modo.

I nomi campo nelle dichiarazioni nei file di copia della struttura sono in maiuscolo. È possibile utilizzare caratteri minuscoli o maiuscoli. Ad esempio, il campo *StrucId* della struttura MQGMO viene visualizzato come MQGMO - STRUCID nella dichiarazione COBOL e nel file di copia.

Le strutture con suffisso V vengono dichiarate con i valori iniziali per tutti i campi, quindi è necessario impostare solo quei campi in cui il valore richiesto è diverso dal valore iniziale.

Macro linguaggio assembler System/390

z/OS

IBM MQ for z/OS fornisce due macro in linguaggio assembler contenenti le costanti denominate e una macro per creare ciascuna struttura.

Sono elencati in z/OS Assembler COPY files e installati in **thlqual**.SCSQMACS.

Queste macro vengono richiamate utilizzando il seguente codice:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

File di inclusione PL/I

z/OS

IBM MQ for z/OS fornisce file di inclusione che contengono tutte le definizioni necessarie quando si scrivono applicazioni IBM MQ in PL/I.

I file sono elencati in PL/I include files e installati nella directory **thlqual**.SCSQPLIC:

Includere questi file nel programma se si intende collegare lo stub IBM MQ al programma (consultare “Preparazione del programma per l'esecuzione” a pagina 1025). Includere solo CMQP se si intende collegare dinamicamente le chiamate IBM MQ (consultare “Richiamo dinamico dello stub IBM MQ” a pagina 1031). Il collegamento dinamico può essere eseguito solo per programmi batch e IMS .

Scrittura di un'applicazione procedurale per l'accodamento

Utilizzare queste informazioni per informazioni sulla scrittura delle applicazioni di accodamento, la connessione e la disconnessione da un gestore code, la pubblicazione / sottoscrizione e l'apertura e la chiusura di oggetti.

Utilizzare i seguenti collegamenti per ulteriori informazioni sulla scrittura di applicazioni:

- “Panoramica su Message Queue Interface” a pagina 722
- “Connessione e disconnessione da un gestore code” a pagina 735
- “Apertura e chiusura di oggetti” a pagina 742
- “Inserimento di messaggi in una coda” a pagina 753
- “Richiamo dei messaggi da una coda” a pagina 768
- “Scrittura di applicazioni di pubblicazione / sottoscrizione” a pagina 809
- “Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
- “Commit e backout delle unità di lavoro” a pagina 854
- “Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
- “Utilizzo di MQI e cluster” a pagina 886
- z/OS “Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890

- z/OS “Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71

Concetti correlati

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

“Sviluppo di applicazioni per IBM MQ” a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

“Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51
Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

“Scrittura di applicazioni procedurali client” a pagina 915
Cosa devi sapere per scrivere le applicazioni client su IBM MQ utilizzando un linguaggio procedurale.

“Creazione di un'applicazione procedurale” a pagina 1002
È possibile scrivere un'applicazione IBM MQ in uno dei diversi linguaggi procedurali ed eseguire l'applicazione su diverse piattaforme.

“Gestione degli errori del programma procedurale” a pagina 1040
Queste informazioni spiegano gli errori associati alle chiamate MQI delle applicazioni quando effettua una chiamata o quando il suo messaggio viene consegnato alla destinazione finale.

Attività correlate

“Utilizzo dei programmi procedurali di esempio IBM MQ” a pagina 1060
Questi programmi di esempio sono scritti in linguaggi procedurali e dimostrano gli usi tipici di MQI (Message Queue Interface). Programmi IBM MQ su diverse piattaforme.

Panoramica su Message Queue Interface

Informazioni sui componenti MQI (Message Queue Interface).

L'interfaccia della coda messaggi è composta da:

- *Chiamate* tramite cui i programmi possono accedere al gestore code e alle relative funzioni
- *Strutture* che i programmi utilizzano per trasmettere e richiamare i dati dal gestore code
- *Tipi di dati elementari* per la trasmissione e l'acquisizione di dati dal gestore code

z/OS IBM MQ for z/OS fornisce inoltre:

- Due chiamate aggiuntive tramite cui i programmi batch z/OS possono eseguire il commit e il backout delle modifiche.
- *File di definizione dati* (a volte noti come file di copia, macro, file di inclusione e file di intestazione) che definiscono i valori delle costanti fornite con IBM MQ for z/OS.
- *Programmi stub* per collegare le modifiche alle applicazioni.
- Una suite di programmi di esempio che dimostrano come utilizzare MQI sulla piattaforma z/OS . Per ulteriori informazioni su questi esempi, consultare “Utilizzo dei programmi di esempio per z/OS” a pagina 1164.

IBM i IBM MQ for IBM i fornisce inoltre:

- *File di definizione dati* (a volte noti come file di copia, macro, file di inclusione e file di intestazione) che definiscono i valori delle costanti fornite con IBM MQ for IBM i.
- Tre programmi stub da collegare - modificare alle proprie applicazioni ILE C, ILE COBOL e ILE RPG.
- Una suite di programmi di esempio che dimostrano come utilizzare MQI sulla piattaforma IBM i .

I sistemi AIX, Linux, and Windows forniscono inoltre:

- Chiamate tramite cui i programmi di sistema IBM MQ for AIX, Linux, and Windows possono eseguire il commit e il backout delle modifiche.
- *Includi file* che definiscono i valori delle costanti fornite su queste piattaforme.
- *File di libreria* per collegare le applicazioni.
- Una suite di programmi di esempio che dimostrano come utilizzare MQI su queste piattaforme. Per ulteriori informazioni su questi esempi, consultare "Utilizzo dei programmi di esempio su Multiplatforms" a pagina 1061.
- Codice di origine ed eseguibile di esempio per i bind ai gestori transazioni esterni.

Utilizzare i seguenti link per ulteriori informazioni su MQI:

- "Chiamate MQI" a pagina 723
- "Chiamate punto di sincronizzazione" a pagina 724
- "Conversione dati, tipi di dati, definizioni di dati e strutture" a pagina 725
- "File di libreria e programmi stub IBM MQ" a pagina 726
- "Parametri comuni a tutte le chiamate" a pagina 731
- "Specifica dei buffer" a pagina 732
- z/OS "Considerazioni sul batch z/OS" a pagina 732
- "Gestione del segnale AIX and Linux" a pagina 733

Concetti correlati

"Connessione e disconnessione da un gestore code" a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

"Apertura e chiusura di oggetti" a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

"Inserimento di messaggi in una coda" a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

"Richiamo dei messaggi da una coda" a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

"Richiesta di informazioni e impostazione degli attributi dell'oggetto" a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

"Commit e backout delle unità di lavoro" a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

"Avvio delle applicazioni IBM MQ utilizzando i trigger" a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

"Utilizzo di MQI e cluster" a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

"Utilizzo e scrittura di applicazioni su IBM MQ for z/OS" a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

"Applicazioni bridge IMS e IMS su IBM MQ for z/OS" a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Chiamate MQI

Utilizzare queste informazioni per informazioni sulle chiamate in MQI (Message Queue Interface).

Le chiamate in MQI possono essere raggruppate come segue:

MQCONN, MQCONNX e MQDISC
Utilizzare queste chiamate per connettere un programma a (con o senza opzioni) e disconnettere un programma da un gestore code. Se si scrivono programmi CICS per z/OS, non è necessario utilizzare tali chiamate. Tuttavia, si consiglia di utilizzarli se si desidera trasferire l'applicazione su altre piattaforme.

MQOPEN e MQCLOSE
Utilizzare queste chiamate per aprire e chiudere un oggetto, ad esempio una coda.

MQPUT e MQPUT1
Utilizzare queste chiamate per inserire un messaggio in una coda.

MQGET
Utilizzare questa chiamata per esaminare i messaggi su una coda o per rimuovere i messaggi da una coda.

MQSUB, MQSUBRQ
Utilizzare queste chiamate per registrare una sottoscrizione ad un argomento e per richiedere le pubblicazioni corrispondenti alla sottoscrizione.

MQINQ
Utilizzare questa chiamata per esaminare gli attributi di un oggetto.

MQSET
Utilizzare questa chiamata per impostare alcuni attributi di una coda. Non è possibile impostare gli attributi di altri tipi di oggetto.

MQBEGIN, MQCMIT e MQBACK
Utilizzare queste chiamate quando IBM MQ è il coordinatore di un'unità di lavoro. MQBEGIN avvia l'unità di lavoro. MQCMIT e MQBACK terminano l'unità di lavoro, eseguendo il commit o il rollback degli aggiornamenti effettuati durante l'unità di lavoro. IBM i Il controller di commit IBM i viene utilizzato per coordinare le unità di lavoro globali su IBM MQ for IBM i. Vengono utilizzati i comandi nativi di avvio del controllo di commit, commit e rollback.

MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH
Utilizzare queste chiamate per creare un handle del messaggio, per convertire un handle del messaggio in un buffer o un buffer in un handle del messaggio e per eliminare un handle del messaggio.

MQSETMP, MQINQMP, MQDLTMP
Utilizzare queste chiamate per impostare una proprietà del messaggio su un handle del messaggio, analizzare una proprietà del messaggio ed eliminare una proprietà da un handle del messaggio.

MQCB, MQCB_FUNCTION, MQCTL
Utilizzare queste chiamate per registrare e controllare una funzione di callback.

MQSTAT
Utilizzare questa chiamata per richiamare le informazioni sullo stato delle precedenti operazioni di inserimento asincrone.

Consultare Descrizioni chiamata per una descrizione delle chiamate MQI.

Chiamate punto di sincronizzazione
Utilizzare queste informazioni per informazioni sulle chiamate sync point su diverse piattaforme.

Le chiamate del punto di sincronizzazione sono disponibili come segue:

IBM MQ for z/OS chiamate
z/OS

IBM MQ for z/OS fornisce le chiamate MQCMIT e MQBACK.

Utilizzare queste chiamate nei programmi batch z/OS per indicare al gestore code che tutte le operazioni MQGET e MQPUT dall'ultimo punto di sincronizzazione devono essere rese permanenti (sottoposte a

commit) o devono essere sottoposte a backout. Per eseguire il commit e il backout delle modifiche in altri ambienti:

CICS
Utilizzare comandi come EXEC CICS SYNCPOINT e EXEC CICS SYNCPOINT ROLLBACK.

IMS
Utilizzare le funzioni del punto di sincronizzazione IMS , come ad esempio GU (get unique) per le chiamate IOPCB, CHKP (checkpoint) e ROLB (rollback).

RRS
Utilizzare MQCMIT e MQBACK o SRRCMIT e SRRBACK come appropriato. (Consultare “Gestione delle transazioni e servizi di gestione delle risorse recuperabili” a pagina 859.)

Nota: SRRCMIT e SRRBACK sono comandi RRS nativi, non sono chiamate MQI.

IBM i chiamate

IBM i

IBM MQ for IBM i fornisce i comandi MQCMIT e MQBACK. È anche possibile utilizzare i comandi IBM i COMMIT e ROLLBACK o qualsiasi altro comando o chiamata che avvia le funzioni di controllo del commit IBM i (ad esempio, EXEC CICS SYNCPOINT).

Chiamate IBM MQ su piattaforme AIX, Linux, and Windows

ALW

IBM MQ for AIX, Linux, and Windows fornisce le chiamate MQCMIT e MQBACK.

Utilizzare le chiamate al punto di sincronizzazione nei programmi per indicare al gestore code che tutte le operazioni MQGET e MQPUT dall'ultimo punto di sincronizzazione devono essere rese permanenti (sottoposte a commit) o devono essere sottoposte a backout. Per eseguire il commit e il backout delle modifiche nell'ambiente CICS , utilizzare comandi quali EXEC CICS SYNCPOINT e EXEC CICS SYNCPOINT ROLLBACK.

Conversione dati, tipi di dati, definizioni di dati e strutture

Utilizzare queste informazioni per informazioni sulle conversioni di dati, sui tipi di dati elementari, sulle definizioni di dati IBM MQ e sulle strutture quando si utilizza l'interfaccia della coda messaggi.

Conversione dati

La chiamata MQXCNVC (convert characters) converte i dati del carattere del messaggio da una serie di caratteri a un altra. Tranne che su IBM MQ for z/OS, questa chiamata viene utilizzata solo da un'uscita di conversione dati.

Consultare MQXCNVC - Converti caratteri per la sintassi utilizzata con la chiamata MQXCNVC e “Scrittura delle uscite di conversione dati” a pagina 985 per istruzioni sulla scrittura e il richiamo delle uscite di conversione dati.

Tipi di dati elementari

Per i linguaggi di programmazione supportati, MQI fornisce tipi di dati elementari o campi non strutturati.

Questi tipi di dati sono descritti completamente in Tipi di dati elementari.

IBM MQ definizioni di dati

z/OS IBM MQ for z/OS fornisce definizioni di dati sotto forma di file di copia COBOL, macro di linguaggio assembly, un singolo file di inclusione PL/I, un singolo file di inclusione di linguaggio C e file di inclusione di linguaggio C + +.

IBM i IBM MQ for IBM i fornisce definizioni di dati sotto forma di file di copia COBOL, file di copia RPG, file di inclusione in linguaggio C e file di inclusione in linguaggio C + +.

I file di definizioni dati forniti con IBM MQ contengono:

- Definizioni di tutte le costanti e codici di ritorno IBM MQ
- Definizioni delle strutture e dei tipi di dati IBM MQ
- Definizioni costanti per l'inizializzazione delle strutture
- Prototipi di funzione per ogni chiamata (solo per PL/I e il linguaggio C)

Per una descrizione completa dei file di definizione dati IBM MQ , consultare “File di definizione dati IBM MQ” a pagina 718.

Strutture

Le strutture, utilizzate con le chiamate MQI elencate in “Chiamate MQI” a pagina 723, vengono fornite nei file di definizione dati per ciascuno dei linguaggi di programmazione supportati.

IBM i z/OS IBM MQ for z/OS e IBM MQ for IBM i forniscono file che contengono costanti da utilizzare quando si completano alcuni campi di queste strutture. Per ulteriori informazioni su queste, consultare Definizioni di dati IBM MQ.

Consultare Tipi di dati della struttura per un riepilogo delle strutture.

File di libreria e programmi stub IBM MQ

I programmi stub e i file di libreria forniti sono elencati qui, per ogni piattaforma.

Per ulteriori informazioni su come utilizzare i programmi stub e i file di libreria quando si crea un'applicazione eseguibile, consultare “Creazione di un'applicazione procedurale” a pagina 1002. Per informazioni sul collegamento ai file della libreria C++, consultare Utilizzo di C++ *IBM MQ Utilizzo di C++*.

AIX *IBM MQ for AIX File di libreria*

Su IBM MQ for AIX, è necessario collegare il programma ai file della libreria MQI forniti per l'ambiente in cui viene eseguita l'applicazione, oltre a quelli forniti dal sistema operativo.

In un'applicazione non sottoposta a thread, collegarsi a una delle seguenti librerie:

Tabella 106. File di libreria per applicazioni AIX senza thread

File di libreria	**Ambiente**
libmqm.a	Server per C
libmqic.a e libmqm.a	Client per C
libmqmzf.a	Uscite di servizio installabili per C
libmqmxa.a	Interfaccia XA server
libmqmxa64.a	Interfaccia XA alternativa server
libmqcxa.a	Interfaccia XA client
libmqcxa64.a	Interfaccia XA alternativa del client
libmqmcbrt.o	Libreria di runtime IBM MQ per il supporto Micro Focus COBOL
libmqmcb.a	Server per COBOL
libmqicb.a	Client per COBOL
libimqc23ia.a	Client per C++ (XLC 16)
libimqs23ia.a	Server per C++ (XLC 16)

Tabella 106. File di libreria per applicazioni AIX senza thread (Continua)	
File di libreria	**Ambiente**
V 9.3.5 libimqc23ca.a	Client per C++ (XLC 17)
V 9.3.5 libimqs23ca.a	Server per C++ (XLC 17)

V 9.3.5 Le librerie che contengono "ia" sono state create con il compilatore XLC 16, mentre le librerie con "ca" nel nome sono state create con il compilatore XLC 17.

In un'applicazione con thread, collegarsi a una delle seguenti librerie:

Tabella 107. File di libreria per applicazioni AIX con thread. Una tabella a due colonne che elenca i file della libreria e l'ambiente per ciascun file della libreria.	
File di libreria	**Ambiente**
libmqm_r.a	Server per C
libmqic_r.a e libmqm_r.a	Client per C
libmqmzf_r.a	Uscite di servizio installabili per C
libmqmxa_r.a	Interfaccia XA server
libmqmxa64_r.a	Interfaccia XA alternativa server
libmqcxa_r.a	Interfaccia XA client
libmqcxa64_r.a	Interfaccia XA alternativa del client
libimqc23ia_r.a	Client per C++ (XLC 16)
libimqs23ia_r.a	Server per C++ (XLC 16)
V 9.3.5 libimqc23ca_r.a	Client per C++ (XLC 17)
V 9.3.5 libimqs23ca_r.a	Server per C++ (XLC 17)

V 9.3.5 Le librerie che contengono "ia" sono state create con il compilatore XLC 16, mentre le librerie con "ca" nel nome sono state create con il compilatore XLC 17.

Nota: Non è possibile collegarsi a più di una libreria. In altre parole, non è possibile collegarsi contemporaneamente a una libreria con thread e non con thread.

IBM i *IBM MQ for IBM i File di libreria*

In IBM MQ for IBM i, collegare i file della libreria MQI forniti per l'ambiente in cui si sta eseguendo l'applicazione, in aggiunta a quelli forniti dal sistema operativo.

Per applicazioni senza thread:

Tabella 108. File di libreria per applicazioni IBM i senza thread	
File di libreria	**Ambiente**
LIBMQM	Programma di servizio server e client
LIBMQIC	Programma di servizio client
IMQB23I4	Programma di servizio di base C++
IMQS23I4	Programma di servizio server C++

Tabella 108. File di libreria per applicazioni IBM i senza thread (Continua)	
File di libreria	**Ambiente**
LIBMQMZF	Uscite installabili per C

In un'applicazione con thread:

Tabella 109. File di libreria per applicazioni IBM i con thread	
File di libreria	**Ambiente**
LIBRERIA_R	Programma di servizio server & client
IMQB23I4_R	Programma di servizio di base C++
IMQS23I4_R	Programma di servizio server C++
LIBMQMZF_P	Uscite installabili per C
LIBMQIC_R	Programma di servizio client

Su IBM MQ for IBM i, puoi scrivere le tue applicazioni in C + +. Per vedere come collegare le tue applicazioni C + + e per i dettagli completi di tutti gli aspetti dell'utilizzo di C + +, vedi Utilizzo di C++.

Linux *File della libreria IBM MQ per Linux*

Su IBM MQ per Linux, è necessario collegare il programma ai file della libreria MQI forniti per l'ambiente in cui si sta eseguendo l'applicazione, oltre a quelli forniti dal sistema operativo.

In un'applicazione non sottoposta a thread, collegarsi a una delle seguenti librerie:

Tabella 110. File di libreria per applicazioni Linux senza thread	
File di libreria	**Ambiente**
libmqm.so	Server per C
libmqic.so e libmqm.so	Client per C
libmqmzf.so	Uscite di servizio installabili per C
libmqmxa.so	Interfaccia XA server
libmqmxa64.so	Interfaccia XA alternativa server
libmqcxa.so	Interfaccia XA client
libmqcxa64.so	Interfaccia XA alternativa del client
libimqc23gl.so	Client per C++
libimqs23gl.so	Server per C++

In un'applicazione con thread, collegarsi a una delle seguenti librerie:

Tabella 111. File di libreria per applicazioni Linux con thread	
File di libreria	**Ambiente**
libmqm_r.so	Server per C
libmqic_r.so e libmqm_r.so	Client per C
libmqmzf_r.so	Uscite di servizio installabili per C
libmqmxa_r.so	Interfaccia XA server

Tabella 111. File di libreria per applicazioni Linux con thread (Continua)	
File di libreria	**Ambiente**
libmqmxa64_r.so	Interfaccia XA alternativa server
libmqcxa_r.so	Interfaccia XA client
libmqcxa64_r.so	Interfaccia XA alternativa del client
libimqc23gl_r.so	Client per C++
libimqs23gl_r.so	Server per C++

Nota: Non è possibile collegarsi a più di una libreria. In altre parole, non è possibile collegarsi contemporaneamente a una libreria con thread e non con thread.

Windows *IBM MQ for Windows File di libreria*

Su IBM MQ for Windows, è necessario collegare il programma ai file della libreria MQI forniti per l'ambiente in cui viene eseguita l'applicazione, oltre a quelli forniti dal sistema operativo:

Tabella 112. File di libreria per applicazioni Windows	
File di libreria	**Ambiente**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Server per C (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Client per C (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	Interfaccia XA server per C (32-bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	Interfaccia client XA per C (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Client MTS per C (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	Supporto TXSeries CICS server per C (32-bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	Supporto TXSeries CICS client per C (32-bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Uscite di servizi installabili per C (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Server per IBM COBOL (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Server per Micro Focus COBOL (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Client per IBM COBOL (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Client per Micro Focus COBOL (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Server per C++ (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Client per C+ + (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	Base per C++ (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Client MTS per C+ + (32 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Server per C (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Client per C (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	Interfaccia XA server per C (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	Interfaccia XA client per C (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	Client MTS per C (64 bit)

Tabella 112. File di libreria per applicazioni Windows (Continua)	
File di libreria	**Ambiente**
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Server per IBM COBOL (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Server per Micro Focus COBOL (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	Client per IBM COBOL (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Client per Micro Focus COBOL (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	Server per C++ (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	Client per C++ (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	Base per C++ (64 bit)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	Client MTS per C++ (64 bit)

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Utilizzare `amqmdnet.dll` per compilare i programmi .NET . Per ulteriori informazioni, consultare “Compilazione dei programmi IBM MQ .NET” a pagina 612 all'interno della sezione “Sviluppo di applicazioni .NET” a pagina 556 .

Questi file vengono forniti per la compatibilit ... con le release precedenti:

```
mqic32.lib
mqic32xa.lib
```

z/OS *Programmi stub IBM MQ for z/OS*

Prima di poter eseguire un programma scritto con IBM MQ for z/OS, è necessario collegarlo - modificarlo al programma stub fornito con IBM MQ for z/OS per l'ambiente in cui si sta eseguendo l'applicazione.

Il programma stub fornisce la prima fase dell'elaborazione delle chiamate in richieste che IBM MQ for z/OS può elaborare.

IBM MQ for z/OS fornisce i seguenti programmi stub:

CSQBSTUB
Programma stub per i programmi batch z/OS

CSQBRRSI
Programma stub per programmi batch z/OS che utilizzano RRS tramite MQI

CSQBRSTB
Programma stub per programmi batch z/OS utilizzando direttamente RRS

CSQCSTUB
Programma stub per programmi CICS

SEGNAPOSTO
Programma stub per programmi IMS

CSQXSTUB
Programma stub per uscite nonCICS di accodamento distribuito

CSQASTUB
Programma stub per uscite di conversione dati

Attenzione: Se si utilizza un programma stub diverso da uno elencato per un ambiente specifico, potrebbe avere risultati imprevedibili.

Nota: Se si utilizza il programma stub CSQBRSTB, modificare il collegamento con ATRSCSS da SYS1.CSSLIB. (SYS1.CSSLIB è noto anche come *Callable Services Library*). Per ulteriori informazioni su RRS, consultare "Gestione delle transazioni e servizi di gestione delle risorse recuperabili" a pagina 859.

In alternativa, è possibile richiamare dinamicamente lo stub dall'interno del programma. Questa tecnica è descritta in "Richiamo dinamico dello stub IBM MQ" a pagina 1031.

In IMS, potrebbe essere necessario utilizzare anche un modulo di interfaccia di lingua speciale fornito da IBM MQ.

Non eseguire le applicazioni che sono state modificate tramite link con CSQBSTUB e CSQQSTUB nella stessa regione MPP IMS . Ciò può causare problemi come i messaggi DFS3607I o CSQQ005E . La prima chiamata MQCONN in uno spazio di indirizzo determina quale interfaccia viene utilizzata, quindi le transazioni CSQQSTUB e CSQBSTUB devono essere eseguite in regioni di messaggi IMS differenti.

Parametri comuni a tutte le chiamate

Esistono due tipi di parametri comuni a tutte le chiamate: handle e codici di ritorno.

Utilizzo delle maniglie

Tutte le chiamate MQI utilizzano uno o più *handle*. Questi identificano il gestore code, la coda o un altro oggetto, messaggio o sottoscrizione, come appropriato per la chiamata.

Perché un programma comunichi con un gestore code, il programma deve avere un identificativo univoco con cui riconosce tale gestore code. Questo identificativo viene definito *handle di collegamento*, a volte indicato come *Hconn*. Per programmi CICS , l'handle di collegamento è sempre zero. Per tutte le altre piattaforme o stili di programmi, l'handle di connessione viene restituito dalla chiamata MQCONN o MQCONNX quando il programma si connette al gestore code. I programmi passano l'handle di collegamento come un parametro di input quando utilizzano le altre chiamate.

Perché un programma possa gestire un oggetto IBM MQ , il programma deve avere un identificativo univoco con cui riconosce tale oggetto. Questo identificativo è denominato *handle dell'oggetto*, a volte indicato come *Hobj*. L'handle viene restituito dalla chiamata MQOPEN quando il programma apre l'oggetto per gestirlo. I programmi passano la gestione dell'oggetto come un parametro di input quando utilizzano chiamate MQPUT, MQGET, MQINQ, MQSET o MQCLOSE successive.

Allo stesso modo, la chiamata MQSUB restituisce un *handle della sottoscrizione* o *Hsub*, utilizzato per identificare la sottoscrizione nelle successive chiamate MQGET, MQCB o MQSUBRQ e alcune chiamate che elaborano le proprietà del messaggio utilizzano un *handle del messaggio* o *Hmsg*.

Informazioni sui codici di ritorno

Un codice di completamento e un codice motivo vengono restituiti come parametri di output da ogni chiamata. Questi sono noti collettivamente come *codici di ritorno*.

Per mostrare se una chiamata ha esito positivo, ogni chiamata restituisce un *codice di completamento* quando la chiamata è completa. Il codice di completamento è in genere MQCC_OK che indica l'esito positivo o MQCC_FAILED che indica l'errore. Alcune chiamate possono restituire uno stato intermedio, MQCC_WARNING, che indica un esito positivo parziale.

Inoltre, ogni chiamata restituisce un *codice di errore* che mostra il motivo dell'errore o dell'esito positivo parziale della chiamata. Ci sono molti codici motivo, che coprono circostanze come una coda piena, operazioni di richiamo non consentite per una coda e una particolare coda non definita per il gestore code. I programmi possono utilizzare il codice di errore per decidere come procedere. Ad esempio, possono richiedere agli utenti di modificare i propri dati di input, quindi effettuare di nuovo la chiamata oppure possono restituire un messaggio di errore all'utente.

Quando il codice di completamento è MQCC_OK, il codice di errore è sempre MQRC_NONE.

I codici di completamento e motivo per ogni chiamata sono elencati con la descrizione di tale chiamata. Consultare Descrizioni delle chiamate e selezionare la chiamata appropriata dall'elenco.

Per informazioni più dettagliate, incluse le idee per un'azione correttiva, consultare:

- z/OS IBM MQ for z/OS , completamento, e codici di errore per IBM MQ for z/OS
- Messaggi e codici di errore per tutte le altre piattaforme IBM MQ

Specifica dei buffer

Il gestore code fa riferimento ai buffer solo se sono richiesti. Se non è richiesto un buffer su una chiamata o se il buffer è di lunghezza zero, è possibile utilizzare un puntatore null su un buffer.

Utilizzare sempre la lunghezza dati quando si specifica la dimensione del buffer richiesta.

Quando si utilizza un buffer per conservare l'emissione da una chiamata (ad esempio, per conservare i dati del messaggio per una chiamata MQGET o i valori degli attributi sottoposti a query dalla chiamata MQINQ), il gestore code tenta di restituire un codice motivo se il buffer specificato non è valido o si trova nella memoria di sola lettura. Tuttavia, potrebbe non essere sempre in grado di restituire un codice motivo.

z/OS *Considerazioni sul batch z/OS*

I programmi batch z/OS che richiamano MQI possono trovarsi nello stato supervisore o problema.

Tuttavia, devono soddisfare le condizioni seguenti:

- Devono essere in modalità attività, non in modalità SRB (service request block).
- Devono essere in modalità ASC (Primary address space control) (non in modalità ASC Access Register).
- Non devono essere in modalità memoria incrociata. L'ASN (address space number) primario deve essere uguale all'ASN secondario e all'ASN principale.
- Non devono essere utilizzati come programmi di uscita MPF.
- Non è possibile congelare alcun blocco z/OS .
- Non possono esistere FRR (function recovery routine) sullo stack FRR.
- Qualsiasi chiave PSW (program status word) può essere attiva per la chiamata MQCONN o MQCONNX (purché la chiave sia compatibile con l'utilizzo della memoria presente nella chiave TCB), ma le chiamate successive che utilizzano l'handle di connessione restituito da MQCONN o MQCONNX:
  - Deve avere la stessa chiave PSW utilizzata nella chiamata MQCONN o MQCONNX
  - Deve avere parametri accessibili (per la scrittura, dove appropriato) sotto la stessa chiave PSW
  - Deve essere emesso sotto la stessa attività (TCB), ma non in alcuna attività secondaria dell'attività
- Possono essere in modalità di indirizzamento a 24 bit o a 31 bit. Tuttavia, se la modalità di indirizzamento a 24 bit è attiva, gli indirizzi dei parametri devono essere interpretati come indirizzi a 31 bit validi.

Se una di queste condizioni non viene soddisfatta, potrebbe verificarsi un controllo del programma. In alcuni casi la chiamata avrà esito negativo e verrà restituito un codice di errore.

Linux AIX *AIX and Linux Considerazioni*

Considerazioni che è necessario tenere presenti quando si sviluppano applicazioni AIX and Linux .

Linux AIX *La chiamata di sistema fork nei sistemi AIX and Linux*

Tenere presenti queste considerazioni quando si utilizza una chiamata di sistema fork nelle applicazioni IBM MQ .

Se l'applicazione desidera utilizzare `fork`, il processo parent di tale applicazione deve richiamare `fork` prima di effettuare qualsiasi chiamata IBM MQ , ad esempio, `MQCONN`o creare un oggetto IBM MQ utilizzando **`ImqQueueManager`**.

Se l'applicazione desidera creare un processo secondario dopo aver eseguito le chiamate IBM MQ , il codice dell'applicazione deve utilizzare un `fork()` con `exec()` per garantire che l'elemento secondario sia una nuova istanza e non una copia esatta dell'elemento principale.

Se l'applicazione non utilizza `exec()`, la chiamata API IBM MQ effettuata all'interno del processo child restituisce MQRC_ENVIRONMENT_ERROR.

Linux AIX *Gestione del segnale AIX and Linux*

In generale, i sistemi AIX and Linux sono stati spostati da un ambiente non thread (processo) a un ambiente multithread. In molti casi, i segnali e la gestione dei segnali, sebbene supportati, non si adattano bene all'ambiente multithread e esistono varie limitazioni.

In generale, i sistemi AIX and Linux sono stati spostati da un ambiente non thread (processo) a un ambiente multithread. Nell'ambiente non thread, alcune funzioni potevano essere implementate solo utilizzando segnali, sebbene la maggior parte delle applicazioni non avesse bisogno di essere a conoscenza dei segnali e della loro gestione. Nell'ambiente a più thread, le primitive basate su thread supportano alcune delle funzioni che erano implementate negli ambienti non thread utilizzando i segnali.

In molti casi, i segnali e la gestione dei segnali, sebbene supportati, non si adattano bene all'ambiente multithread e esistono varie limitazioni. Ciò può essere problematico quando si integra il codice dell'applicazione con diverse librerie middleware (in esecuzione come parte dell'applicazione) in un ambiente a più thread in cui ciascuno tenta di gestire i segnali. L'approccio tradizionale di salvare e ripristinare i gestori di segnale (definiti per processo), che funzionava quando c'era un solo thread di esecuzione all'interno di un processo, non funziona in un ambiente a più thread. Questo perché molti thread di esecuzione potrebbero tentare di salvare e ripristinare una risorsa a livello di processo, con risultati imprevedibili.

Linux AIX *Applicazioni senza thread*

Ogni funzione MQI imposta il proprio gestore di segnali per i segnali. I gestori degli utenti per questi vengono sostituiti per la durata della funzione MQI. Altri segnali possono essere catturati in modo normale dai gestori scritti dall'utente.

Ogni funzione MQI configura il proprio gestore di segnali per i segnali:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

I gestori degli utenti per questi vengono sostituiti per la durata della funzione MQI. Altri segnali possono essere catturati in modo normale dai gestori scritti dall'utente. Se non si installa un gestore, le azioni predefinite (ad esempio, ignora, core dump o esci) vengono lasciate al loro posto.

Dopo che IBM MQ gestisce un segnale sincrono (SIGSEGV, SIGBUS, SIGFPE, SIGILL), tenta di passare il segnale a qualsiasi gestore del segnale registrato prima di effettuare la chiamata della funzione MQI.

Linux AIX *Applicazioni con thread*

Un thread viene considerato connesso a IBM MQ da MQCONN (o MQCONNX) fino a MQDISC.

Segnali sincroni

I segnali sincroni si verificano in un thread specifico.

I sistemi AIX and Linux consentono l'impostazione di un gestore di segnali per tali segnali per l'intero processo. Tuttavia, IBM MQ imposta il proprio gestore per i seguenti segnali, nel processo dell'applicazione, mentre qualsiasi thread è connesso a IBM MQ:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Se si stanno scrivendo applicazioni a più thread, esiste un solo gestore di segnali a livello di processo per ogni segnale. Quando IBM MQ imposta i propri gestori di segnali sincroni, salva i gestori precedentemente registrati per ogni segnale. Dopo che IBM MQ gestisce uno dei segnali elencati, IBM MQ tenta di richiamare il gestore del segnale che era attivo al momento della prima connessione IBM MQ all'interno del processo. I gestori precedentemente registrati vengono ripristinati quando tutti i thread dell'applicazione si sono scollegati da IBM MQ.

Poiché i gestori di segnale vengono salvati e ripristinati da IBM MQ, i thread dell'applicazione non devono stabilire i gestori di segnale per questi segnali mentre esiste la possibilità che un altro thread dello stesso processo sia collegato anche a IBM MQ.

Nota: Quando un'applicazione o una libreria middleware (in esecuzione come parte di un'applicazione) stabilisce un gestore del segnale mentre un thread è connesso a IBM MQ, il gestore del segnale dell'applicazione deve richiamare il gestore IBM MQ corrispondente durante l'elaborazione di tale segnale.

Quando si stabiliscono e si ripristinano gestori di segnali, il principio generale è che l'ultimo gestore di segnali da salvare deve essere il primo ad essere ripristinato:

- Quando un'applicazione stabilisce un gestore del segnale dopo la connessione a IBM MQ, il gestore del segnale precedente deve essere ripristinato prima che l'applicazione si scolleghi da IBM MQ.
- Quando un'applicazione stabilisce un gestore del segnale prima di connettersi a IBM MQ, l'applicazione deve disconnettersi da IBM MQ prima di ripristinare il gestore del segnale.

Nota: La mancata osservanza del principio generale secondo cui l'ultimo gestore del segnale da salvare deve essere il primo da ripristinare può comportare una gestione del segnale non prevista nell'applicazione e, potenzialmente, la perdita di segnali da parte dell'applicazione.

Segnali asincroni

IBM MQ non utilizza segnali asincroni nelle applicazioni con thread a meno che non siano applicazioni client.

Ulteriori considerazioni per le applicazioni client con thread

IBM MQ gestisce i seguenti segnali durante l'I/O su un server. Questi segnali sono definiti dallo stack di comunicazioni. L'applicazione non deve stabilire un gestore segnali per questi segnali mentre un thread è connesso a un gestore code:

SIGPIPE (per TCP/IP)

Linux AIX *Considerazioni aggiuntive quando si utilizza la gestione del segnale AIX and Linux in MQI*

Quando si utilizza MQI per la gestione del segnale su AIX and Linux, ci sono ulteriori considerazioni per le applicazioni fastpath, le chiamate di funzione MQI all'interno dei gestori del segnale, i segnali durante le chiamate MQI, le uscite utente e i servizi installabili e i gestori di uscita VMS.

Applicazioni Fastpath (attendibili)

Le applicazioni Fastpath vengono eseguite nello stesso processo di IBM MQ e quindi vengono eseguite nell'ambiente a più thread.

in questo ambiente IBM MQ gestisce i segnali sincroni SIGSEGV, SIGBUS, SIGFPE e SIGILL. Tutti gli altri segnali non devono essere forniti all'applicazione Fastpath mentre è connessa a IBM MQ. Devono invece essere bloccati o gestiti dall'applicazione. Se un'applicazione Fastpath intercetta un evento di questo tipo, il gestore code deve essere arrestato e riavviato oppure può essere lasciato in uno stato non definito. Per un elenco completo delle limitazioni per le applicazioni Fastpath in MQCONNX, consultare "Connessione a un gestore code utilizzando la chiamata MQCONNX" a pagina 738.

Chiamate della funzione MQI all'interno dei gestori di segnali

Mentre ci si trova in un gestore segnali, non richiamare una funzione MQI.

Se si tenta di richiamare una funzione MQI da un gestore del segnale mentre è attiva un'altra funzione MQI, viene restituito MQRC_CALL_IN_PROGRESS. Se si tenta di richiamare una funzione MQI da un gestore segnali mentre non è attiva nessun' altra funzione MQI, è probabile che abbia esito negativo durante l'operazione a causa delle limitazioni del sistema operativo in cui solo le chiamate selettive possono essere emesse da o all'interno di un gestore.

Per i metodi del distruttore C++, che potrebbero essere richiamati automaticamente durante l'uscita del programma, potrebbe non essere possibile arrestare la chiamata delle funzioni MQI. Ignorare eventuali errori relativi a MQRC_CALL_IN_PROGRESS. Se un gestore del segnale richiama exit (), IBM MQ esegue il backout dei messaggi non sottoposti a commit nel punto di sincronizzazione come al solito e chiude tutte le code aperte.

Segnali durante le chiamate MQI

Le funzioni MQI non restituiscono il codice EINTR o qualsiasi equivalente ai programmi applicativi.

Se un segnale si verifica durante una chiamata MQI e il gestore richiama *return*, la chiamata continua ad essere eseguita come se il segnale non si fosse verificato. In particolare, MQGET non può essere interrotto da un segnale per restituire immediatamente il controllo all'applicazione. Se si desidera uscire da un MQGET, impostare la coda su GET_DISABLED; in alternativa, utilizzare un loop intorno a una chiamata a MQGET con una scadenza a tempo finito (MQGMO_WAIT con gmo.WaitInterval impostato) e utilizzare il gestore del segnale (in un ambiente senza thread) o una funzione equivalente in un ambiente con thread per impostare un indicatore che interrompe il loop.

AIX Nell'ambiente AIX , IBM MQ richiede che le chiamate di sistema interrotte dai segnali vengano riavviate. Quando si stabilisce il proprio gestore del segnale con sigaction (2), impostare l'indicatore SA_RESTART nel campo sa_flags della nuova struttura di azioni altrimenti IBM MQ potrebbe non essere in grado di completare una chiamata interrotta da un segnale.

Uscite utente e servizi installabili

Le uscite utente e i servizi installabili eseguiti come parte di un processo IBM MQ in un ambiente a più thread hanno le stesse limitazioni delle applicazioni fastpath. Considerare che questi siano permanentemente connessi a IBM MQ e quindi non utilizzare segnali o chiamate di sistema operativo non thread - safe.

Connessione e disconnessione da un gestore code

Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

Il modo in cui viene effettuata questa connessione dipende dalla piattaforma e dall'ambiente in cui il programma sta operando:

Multi **IBM MQ for Multiplatforms**

I programmi in esecuzione in questi ambienti possono utilizzare la chiamata MQCONN MQI per connettersi a un gestore code e la chiamata MQDISC per disconnettersi da un gestore code. In alternativa, i programmi possono utilizzare la chiamata MQCONNX.

z/OS **IBM MQ for z/OS batch**

I programmi eseguiti in questo ambiente possono utilizzare la chiamata MQCONN MQI per connettersi a, e la chiamata MQDISC per disconnettersi da, un gestore code. In alternativa, i programmi possono utilizzare la chiamata MQCONNX.

I programmi batch z/OS possono connettersi, consecutivamente o simultaneamente, a più gestori code sullo stesso TCB.

z/OS IMS

La control region IMS è connessa a uno o più gestori code all'avvio. Questa connessione è controllata dai comandi IMS . Per informazioni su come controllare l'adattatore IMS su z/OS, consultare Amministrazione IBM MQ for z/OS. Tuttavia, i writer dei programmi IMS di accodamento messaggi devono utilizzare la chiamata MQCONN MQI per specificare il gestore code a cui si desidera connettersi. Possono utilizzare la chiamata MQDISC per disconnettersi da tale gestore code.

Dopo una chiamata IMS che stabilisce un punto di sincronizzazione e prima di elaborare un messaggio per un altro utente, l'adattatore IMS garantisce che l'applicazione chiuda gli handle e si disconnetta dal gestore code. Consultare “Punti di sincronizzazione nelle applicazioni IMS” a pagina 858.

I programmi IMS possono connettersi, consecutivamente o simultaneamente, a più gestori code sullo stesso TCB.

z/OS CICS Transaction Server per z/OS

I programmi CICS non devono eseguire alcun lavoro per connettersi a un gestore code perché il sistema CICS stesso è connesso. Questa connessione viene generalmente effettuata automaticamente all'inizializzazione, ma è anche possibile utilizzare la transazione CKQC fornita con IBM MQ for z/OS. Per ulteriori informazioni su CKQC, consultare Amministrazione IBM MQ for z/OS.

Le attività CICS possono connettersi solo al gestore code a cui è connessa la region CICS .

I programmi CICS possono anche utilizzare le chiamate di connessione e disconnessione MQI (MQCONN e MQDISC). È possibile eseguire questa operazione in modo da poter trasferire queste applicazioni in ambienti nonCICS con un minimo di ricodifica. Tuttavia, queste chiamate *sempre* vengono completate correttamente in un ambiente CICS . Ciò significa che il codice di ritorno potrebbe non riflettere lo stato reale della connessione al gestore code.

TXSeries per Windows e Open Systems

Questi programmi non hanno bisogno di eseguire alcun lavoro per connettersi a un gestore code perché il sistema CICS stesso è connesso. Pertanto, è supportata solo una connessione alla volta. Le applicazioni CICS devono emettere una chiamata MQCONN per ottenere un gestore connessioni e una chiamata MQDISC prima di uscire.

Utilizzare i seguenti link per ulteriori informazioni sulla connessione e la disconnessione da un gestore code:

- “Connessione a un gestore code utilizzando la chiamata MQCONN” a pagina 737
- “Connessione a un gestore code utilizzando la chiamata MQCONNX” a pagina 738
- “Disconnessione di programmi da un gestore code utilizzando MQDISC” a pagina 742

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886

Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Connessione a un gestore code utilizzando la chiamata MQCONN
Utilizzare queste informazioni per informazioni su come connettersi a un gestore code utilizzando la chiamata MQCONN.

In generale, è possibile connettersi a un gestore code specifico o al gestore code predefinito:

- Per IBM MQ for z/OS, nell'ambiente batch, il gestore code predefinito è specificato nel modulo CSQBDEFV.
- Per IBM MQ for Multiplatforms, il gestore code predefinito è specificato nel file mqs.ini .

In alternativa, negli ambienti z/OS MVS batch, TSO e RRS è possibile connettersi a qualsiasi gestore code all'interno di un gruppo di condivisione code. La richiesta MQCONN o MQCONNX seleziona uno qualsiasi dei membri attivi del gruppo.

Quando ci si connette a un gestore code, deve essere locale per l'attività. Deve appartenere allo stesso sistema dell'applicazione IBM MQ .

Nell'ambiente IMS , il gestore code deve essere connesso alla control region IMS e alla region dipendente utilizzata dal programma. Il gestore code predefinito viene specificato nel modulo CSQQDEFV quando è installato IBM MQ for z/OS .

Con l'ambiente TXSeries CICS e TXSeries per Windows e AIX, il gestore code deve essere definito come una risorsa XA per CICS.

Per connettersi al gestore code predefinito, richiamare MQCONN, specificando un nome composto interamente da spazi vuoti o che inizia con un carattere null (X'00 ').

Un'applicazione deve essere autorizzata per connettersi correttamente a un gestore code. Per ulteriori informazioni, consultare Protezione.

L'output di MQCONN è:

- Un handle di collegamento (**Hconn**)
- Un codice di completamento
- Un codice di errore

Utilizzare l'handle di connessione nelle chiamate MQI successive.

Se il codice motivo indica che l'applicazione è già connessa a tale gestore code, l'handle di connessione restituito è uguale a quello restituito quando l'applicazione si è connessa per la prima volta. L'applicazione non deve emettere la chiamata MQDISC in questa situazione perché l'applicazione chiamante prevede di rimanere connessa.

L'ambito dell'handle di connessione è uguale all'ambito dell'handle di oggetto (consultare “Apertura di oggetti utilizzando la chiamata MQOPEN” a pagina 744).

Le descrizioni dei parametri sono fornite nella descrizione della chiamata MQCONN in MQCONN.

La chiamata MQCONN ha esito negativo se il gestore code è in uno stato di inattività quando si emette la chiamata o se il gestore code è in fase di arresto.

Ambito di MQCONN o MQCONNX

L'ambito di una chiamata MQCONN o MQCONNX è in genere il thread che l'ha emessa. Ovvero, l'handle di connessione restituito dalla chiamata è valido solo all'interno del thread che ha emesso la chiamata. È possibile effettuare una sola chiamata alla volta utilizzando l'handle. Se viene utilizzato da un thread

differente, viene rifiutato come non valido. Se si dispone di più thread nell'applicazione e ciascuno desidera utilizzare le chiamate IBM MQ , ciascuno deve emettere MQCONN o MQCONNX.

Non è necessario che ogni chiamata venga effettuata allo stesso gestore code quando un processo effettua più chiamate MQCONN. Tuttavia, è possibile stabilire una sola connessione IBM MQ da un thread alla volta. In alternativa, considerare "Connessioni condivise (indipendenti dal thread) con MQCONNX" a pagina 739 per consentire l'utilizzo di più connessioni IBM MQ da un singolo thread e di una connessione IBM MQ da qualsiasi thread.[7]

Se l'applicazione è in esecuzione come client, può connettersi a più di un gestore code all'interno di un thread.

Connessione a un gestore code utilizzando la chiamata MQCONNX

La chiamata MQCONNX è simile alla chiamata MQCONN, ma include opzioni per controllare il funzionamento della chiamata.

Come input per MQCONNX, è possibile fornire un nome gestore code z/OS o un nome gruppo di condivisione code su z/OS sistemi di code condivisi. Le opzioni per controllare la modalità di connessione al gestore code vengono fornite in una struttura denominata MQCNO.

L'output da MQCONNX è:

- Un handle di connessione (Hconn)
- Un codice di completamento
- Un codice di errore

Utilizzare l'handle di connessione nelle chiamate MQI successive.

Le opzioni di collegamento, impostate nel campo *Options* della struttura MQCNO, consentono di controllare diversi attributi della connessione. Di particolare nota sono i seguenti gruppi di opzioni:

- Le opzioni di collegamento consentono di creare *applicazioni attendibili* . Le applicazioni attendibili implicano che l'applicazione IBM MQ e l'agent del gestore code locale diventano lo stesso processo. Poiché il processo agent non ha più bisogno di utilizzare un'interfaccia per accedere al gestore code, queste applicazioni diventano un'estensione del gestore code. Questo comportamento viene richiesto specificando l'opzione MQCNO_FASTPATH_BINDING . Per ulteriori informazioni sulle limitazioni applicabili alle applicazioni attendibili, consultare "Limitazioni per le applicazioni attendibili" a pagina 738.
- Le opzioni di condivisione della gestione consentono di creare connessioni condivise. Le connessioni condivise possono condividere handle tra thread differenti all'interno dello stesso processo. Per ulteriori informazioni sulle connessioni condivise, consultare "Connessioni condivise (indipendenti dal thread) con MQCONNX" a pagina 739.

MQCNO consente inoltre all'applicazione di controllare come viene autenticata la connessione al gestore code. Le credenziali di autenticazione possono essere specificate in una struttura MQCSP a cui si fa riferimento dalla struttura MQCNO.

Per una descrizione completa dei parametri della chiamata MQCONNX e degli attributi di connessione che possono essere controllati, consultare MQCONNX - Connetti gestore code (esteso).

Limitazioni per le applicazioni attendibili

Limitazioni che si applicano alle applicazioni attendibili. Alcune restrizioni si applicano a tutte le piattaforme e altre sono specifiche della piattaforma.

T

- È necessario disconnettere esplicitamente le applicazioni attendibili dal gestore code.

[7] Quando si utilizzano applicazioni a più sottoprocessi con sistemi IBM MQ for AIX or Linux , è necessario verificare che le applicazioni abbiano una dimensione di stack sufficiente per i sottoprocessi. Considerare l'utilizzo di una dimensione di stack pari o superiore a 256 KB, quando le applicazioni a più thread effettuano chiamate MQI, da sole o con altri gestori di segnali (ad esempio, CICS).

- È necessario arrestare le applicazioni attendibili prima di arrestare il gestore code con il comando **endmqm** .
- Non utilizzare segnali asincroni e interruzioni timer (come `sigkill`) con MQCNO_FASTPATH_BINDING.
- Su tutte le piattaforme, un thread all'interno di un'applicazione attendibile non può connettersi a un gestore code mentre un altro thread nello stesso processo è connesso a un gestore code differente.
- Linux AIX Sui sistemi AIX and Linux è necessario utilizzare mqm come userID e groupID effettivi per tutte le chiamate MQI. È possibile modificare questi ID prima di effettuare una chiamata non MQI che richiede l'autenticazione (ad esempio, l'apertura di un file), ma è necessario modificarlo di nuovo in mqm prima di effettuare la successiva chiamata MQI.
- IBM i Su IBM i:
  1. Le applicazioni attendibili devono essere eseguite con il profilo utente QMQM. Non è sufficiente che il profilo utente sia un membro del gruppo QMQM o che il programma adotti l'autorizzazione QMQM. È possibile che non sia possibile utilizzare il profilo utente QMQM per collegarsi ai lavori interattivi o per essere specificato nella descrizione del lavoro per i lavori che eseguono applicazioni attendibili. In questo caso, un approccio consiste nell'utilizzare le funzioni API di scambio profili IBM i , QSYGETPH, QWTSETP e QSYRLSPH per modificare temporaneamente l'utente corrente del lavoro in QMQM mentre vengono eseguiti i programmi IBM MQ . I dettagli di queste funzioni, insieme a un esempio di utilizzo, sono forniti nella sezione API di sicurezza della documentazione IBM i*Application programming interfaces* .
  2. Non annullare le applicazioni attendibili utilizzando l'opzione 2 della richiesta di sistema o terminando i lavori in cui sono in esecuzione utilizzando ENDJOB.
- ALW Su sistemi AIX, Linux, and Windows , le applicazioni a 32 bit attendibili non sono supportate. Se si tenta di eseguire un'applicazione a 32 bit attendibile, verrà eseguito il downgrade a una connessione collegata standard.

Connessioni condivise (indipendenti dal thread) con MQCONNX
Utilizzare queste informazioni per informazioni sulle connessioni condivise con MQCONNX e alcune note di utilizzo da considerare.

Nota: Non supportato su IBM MQ for z/OS.

Su piattaforme IBM MQ diverse da IBM MQ for z/OS, una connessione effettuata con `MQCONN` è disponibile solo per il thread che ha effettuato la connessione. Le opzioni sulla chiamata MQCONNX consentono di creare una connessione che può essere condivisa da tutti i thread in un processo. Se l'applicazione è in esecuzione in un ambiente transazionale che richiede l'emissione di chiamate MQI sullo stesso thread, è necessario utilizzare la seguente opzione predefinita:

MQCNO_HANDLE_SHARE_NONE
Crea una connessione non condivisa.

Nella maggior parte degli altri ambienti, è possibile utilizzare una delle seguenti opzioni di connessione condivisa, indipendente dal thread:

MQCNO_HANDLE_SHARE_BLOCK
Crea una connessione condivisa. Su una connessione `MQCNO_HANDLE_SHARE_BLOCK` , se la connessione è attualmente utilizzata da una chiamata MQI su un altro thread, la chiamata MQI attende il completamento della chiamata MQI corrente.

MQCNO_HANDLE_SHARE_NO_BLOCK
Crea una connessione condivisa. Su una connessione `MQCNO_HANDLE_SHARE_NO_BLOCK` , se la connessione è attualmente utilizzata da una chiamata MQI su un altro thread, la chiamata MQI non riesce immediatamente con un motivo `MQRC_CALL_IN_PROGRESS`.

Tranne che per l'ambiente MTS (Microsoft Transaction Server), il valore predefinito è `MQCNO_HANDLE_SHARE_NONE`. Nell'ambiente MTS, il valore predefinito è `MQCNO_HANDLE_SHARE_BLOCK`.

Un handle di connessione viene restituito dalla chiamata MQCONNX . L'handle può essere utilizzato da chiamate MQI successive da qualsiasi thread nel processo, associando tali chiamate all'handle restituito da MQCONNX. Le chiamate MQI che utilizzano un singolo handle condiviso vengono serializzate tra i thread.

Ad esempio, la seguente sequenza di attività è possibile con un handle condiviso:

1. Il thread 1 emette MQCONNX e ottiene un handle condiviso *h1*
2. Il thread 1 apre una coda ed emette una richiesta get utilizzando *h1*
3. Il thread 2 emette una richiesta di inserimento utilizzando *h1*
4. Il thread 3 emette una richiesta di inserimento utilizzando *h1*
5. Il thread 2 emette MQDISC utilizzando *h1*

Mentre l'handle è utilizzato da qualsiasi thread, l'accesso alla connessione non è disponibile per altri thread. In circostanze in cui è accettabile che un thread attenda il termine di qualsiasi chiamata precedente da un altro thread, utilizzare MQCONNX con l'opzione MQCNO_HANDLE_SHARE_BLOCK.

Tuttavia, il blocco può causare difficoltà. Si supponga che nel passo “2” a pagina 740, il thread 1 emetti una richiesta get che attende i messaggi che potrebbero non essere ancora arrivati (un get con attesa). In questo caso, anche i thread 2 e 3 vengono lasciati in attesa (bloccati) per tutto il tempo richiesto dalla richiesta get sul thread 1. Se si preferisce che una chiamata MQI restituisca un messaggio di errore se un'altra chiamata MQI è già in esecuzione sull'handle, utilizzare MQCONNX con l'opzione MQCNO_HANDLE_SHARE_NO_BLOCK.

Note sull'utilizzo della connessione condivisa

1. Tutti gli handle di oggetto (Hobj) creati aprendo un oggetto sono associati a un Hconn; quindi per un Hconn condiviso, gli Hobj sono anche condivisi e utilizzabili da qualsiasi thread che utilizza l'Hconn. Allo stesso modo, qualsiasi unità di lavoro avviata sotto un Hconn è associata a tale Hconn; quindi anche questo è condiviso tra i thread con l'Hconn condiviso.
2. *Qualsiasi thread* può chiamare MQDISC per scollegare un Hconn condiviso, non solo il thread che ha richiamato il corrispondente MQCONNX. MQDISC termina l'Hconn rendendolo non disponibile per tutti i thread.
3. Un singolo thread può utilizzare più Hconn condivisi in modo seriale, ad esempio utilizzare MQPUT per inserire un messaggio in un Hconn condiviso, quindi inserire un altro messaggio utilizzando un altro Hconn condiviso, con ogni operazione in una diversa unità di lavoro locale.
4. Gli hconn condivisi non possono essere utilizzati all'interno di un'unità di lavoro globale.

Multi *Utilizzo delle opzioni di chiamata MQCONNX con MQ_CONNECT_TYPE*

Utilizzare queste informazioni per comprendere le differenti opzioni di chiamata MQCONNX e come vengono utilizzate con la variabile d'ambiente **MQ_CONNECT_TYPE** .

Nota: MQ_CONNECT_TYPE ha effetto solo per i bind STANDARD. Per altri bind, **MQ_CONNECT_TYPE** viene ignorato.

Su IBM MQ for Multiplatforms, è possibile utilizzare la variabile di ambiente **MQ_CONNECT_TYPE** in combinazione con il tipo di collegamento specificato nel campo Options della struttura MQCNO utilizzata su una chiamata MQCONNX.

*Tabella 113. Modalità di utilizzo delle opzioni di chiamata MQCONNX con la variabile d'ambiente **MQ_CONNECT_TYPE***

Opzione chiamata MQCONNX	Variabile di ambiente MQ_CONNECT_TYPE	Risultato
STANDARD	NON DEFINITO	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	Percorso veloce	STANDARD

*Tabella 113. Modalità di utilizzo delle opzioni di chiamata MQCONNX con la variabile d'ambiente **MQ_CONNECT_TYPE** (Continua)*		
Opzione chiamata MQCONNX	**Variabile di ambiente MQ_CONNECT_TYPE**	**Risultato**
STANDARD	CLIENTE	CLIENTE
STANDARD	LOCALE	STANDARD

Se MQCNO_STANDARD_BINDING non viene specificato, è possibile utilizzare MQCNO_NONE, che assume il valore predefinito MQCNO_STANDARD_BINDING.

Autenticazione e identità per MQCONN e MQCONNX

Utilizzare questa attività per informazioni sul modo in cui le applicazioni possono fornire credenziali utilizzate per l'autenticazione quando si collegano a IBM MQ.

L'identità utente predefinita

Quando un'applicazione utilizza MQI (message queue interface) per connettersi a IBM MQ con MQCONN o MQCONNX, un'identità utente viene sempre stabilita e associata alla connessione.

Per impostazione predefinita, l'identità utente iniziale è sempre quella del processo del sistema operativo con cui è in esecuzione l'applicazione. Questa identità iniziale potrebbe essere sufficiente per le connessioni delle applicazioni collegate localmente o attendibili.

Quando un'applicazione si connette a un gestore code con una chiamata MQCONN, l'applicazione non può modificare l'ID utente predefinito. Tuttavia, i meccanismi seguenti possono modificare l'ID utente associato al collegamento:

- Un'uscita di sicurezza lato client o lato server.
- Regole di autenticazione del canale sul gestore code.
- Un ID utente client stabilito durante l'autenticazione reciproca TLS.

Utilizzo di MQCONNX per fornire credenziali

MQCONNX fornisce a una applicazione un maggiore controllo sull'identità associata alla connessione. Un'applicazione può fornire una struttura MQCSP come parte delle opzioni di connessione specificate nel parametro **ConnectOpts** per MQCONNX. La struttura di MQCSP può contenere credenziali utilizzate per stabilire un'identità utente. IBM MQ supporta le seguenti credenziali nella struttura MQCSP:

- Un ID utente e una password.
- V 9.3.4 Da IBM MQ 9.3.4, un token di autenticazione, se l'applicazione si connette a un gestore code che viene eseguito su sistemi AIX o Linux .

L'autenticazione della connessione del gestore code e la configurazione dell'autenticazione del canale controllano il modo in cui vengono elaborate le credenziali fornite da un'applicazione. Ad esempio, questa configurazione influenza i seguenti aspetti:

- Se le credenziali nella struttura MQCSP vengono convalidate e come vengono convalidate.
- Indica se l'ID utente nelle credenziali nella struttura MQCSP è associato ad un altro ID utente.
- Se l'utente autenticato viene quindi adottato come contesto per l'applicazione.

Per ulteriori informazioni sull'autenticazione della connessione, vedi Autenticazione della connessione. Per ulteriori informazioni sull'autenticazione di canale, vedi Record di autenticazione di canale.

Molti dei programmi di esempio scritti in C che utilizzano MQI dimostrano come la struttura MQCSP viene utilizzata per fornire le credenziali di autenticazione. Per ulteriori informazioni, consultare i seguenti programmi di esempio:

- "I programmi di esempio Get" a pagina 1095

- “Programmi di esempio Put” a pagina 1107
- “Il programma di esempio Browser” a pagina 1082
- “Il programma di esempio TLS” a pagina 1122

Informazioni correlate
Identificazione e autenticazione degli utenti utilizzando la struttura MQCSP
MQCSP - Parametri di sicurezza
Identificazione e autenticazione degli utenti

Disconnessione di programmi da un gestore code utilizzando MQDISC
Utilizzare queste informazioni per informazioni sulla disconnessione dei programmi da un gestore code utilizzando MQDISC.

Quando un programma connesso a un gestore code utilizzando la chiamata MQCONN o MQCONNX ha terminato tutte le interazioni con il gestore code, interrompe la connessione utilizzando la chiamata MQDISC, tranne:

- Su CICS Transaction Server per applicazioni z/OS , dove la chiamata è facoltativa a meno che non sia stato utilizzato MQCONNX e si desideri eliminare la tag di connessione prima che l'applicazione termini.
- Su IBM MQ for IBM i , quando ci si scollega dal sistema operativo, viene effettuata una chiamata MQDISC implicita.

Come input per la chiamata MQDISC, è necessario fornire l'handle di connessione (Hconn) restituito da MQCONN o MQCONNX quando si è connessi al gestore code.

Ad eccezione di CICS su z/OS, dopo che MQDISC è stato richiamato, l'handle di connessione (Hconn) non è più valido e non è possibile emettere ulteriori chiamate MQI finché non si richiama nuovamente MQCONN o MQCONNX. MQDISC esegue un MQCLOSE implicito per tutti gli oggetti ancora aperti utilizzando questo handle.

z/OS Per un client connesso a z/OS, quando viene emessa una chiamata MQDISC, viene eseguito un commit implicito, ma tutti gli handle di coda ancora aperti non vengono chiusi fino a quando il canale non termina.

Se si utilizza MQCONNX per connettersi a IBM MQ for z/OS, MQDISC termina anche l'ambito della tag di connessione stabilita da MQCONNX. Tuttavia, in un'applicazione CICS, IMSo RRS, se esiste un'unità di ripristino attiva associata a una tag di connessione, MQDISC viene rifiutato con un codice motivo MQRC_CONN_TAG_NOT_RELEASED.

Le descrizioni dei parametri vengono fornite nella descrizione della chiamata MQDISC in MQDISC.

Quando non viene emesso alcun MQDISC

Una connessione standard non condivisa (Hconn) viene ripulita quando il thread di creazione termina. Una connessione condivisa viene solo implicitamente ripristinata e disconnessa quando termina l'intero processo. Se il thread che ha creato l'Hconn condiviso termina mentre l'Hconn esiste ancora, l'Hconn è ancora utilizzabile.

Controllo autorizzazione

Le chiamate MQCLOSE e MQDISC di solito non eseguono alcun controllo di autorizzazione.

Nel normale corso degli eventi, un lavoro che dispone dell'autorità per aprire o connettersi a un oggetto IBM MQ si chiude o si disconnette da tale oggetto. Anche se l'autorizzazione di un lavoro che si è collegato o ha aperto un oggetto IBM MQ viene revocata, le chiamate MQCLOSE e MQDISC vengono accettate.

Apertura e chiusura di oggetti

Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

Per eseguire una delle seguenti operazioni, è necessario prima *aprire* l'oggetto IBM MQ pertinente:

- Inserire i messaggi su una coda
- Richiamare (sfogliare o richiamare) i messaggi da una coda
- Impostare gli attributi di un oggetto
- Interroga sugli attributi di qualsiasi oggetto

Utilizzare la chiamata MQOPen per aprire l'oggetto, utilizzando le opzioni della chiamata per specificare cosa si desidera fare con l'oggetto. L'unica eccezione è se si desidera inserire un singolo messaggio in una coda, quindi chiudere immediatamente la coda. In questo caso, puoi ignorare lo stage *opening* utilizzando la chiamata MQPUT1 (vedi “Inserimento di un messaggio su una coda utilizzando la chiamata MQPUT1” a pagina 762).

Prima di aprire un oggetto utilizzando la chiamata MQOPEN, è necessario connettere il programma a un gestore code. Questo è spiegato in dettaglio, per tutti gli ambienti, in “Connessione e disconnessione da un gestore code” a pagina 735.

Esistono quattro tipi di oggetto IBM MQ che è possibile aprire:

- Coda
- Elenco nomi
- Definizione di processo
- Gestore code

Tutti questi oggetti vengono aperti in modo simile utilizzando la chiamata MQOPEN. Per ulteriori informazioni sugli oggetti IBM MQ , consultare Tipi di oggetto.

È possibile aprire lo stesso oggetto più di una volta e ogni volta che si ottiene un nuovo handle di oggetto. È possibile esaminare i messaggi su una coda utilizzando un handle e rimuovere i messaggi dalla stessa coda utilizzando un altro handle. Ciò consente di risparmiare utilizzando le risorse per chiudere e riaprire lo stesso oggetto. È inoltre possibile aprire una coda per sfogliare *e* rimuovere i messaggi contemporaneamente.

Inoltre, è possibile aprire più oggetti con un singolo MQOPEN e chiuderli utilizzando MQCLOSE. Consultare “Liste di distribuzione” a pagina 763 per informazioni su come eseguire questa operazione.

Quando si tenta di aprire un oggetto, il gestore code verifica che l'utente sia autorizzato ad aprire tale oggetto per le opzioni specificate nella chiamata MQOPEN.

Gli oggetti vengono chiusi automaticamente quando un programma si disconnette dal gestore code. Nell'ambiente IMS , la disconnessione viene forzata quando un programma avvia l'elaborazione per un nuovo utente in seguito a una chiamata GU (get unique) IMS . Sulla piattaforma IBM i , gli oggetti vengono chiusi automaticamente al termine di un lavoro.

È buona prassi di programmazione chiudere gli oggetti aperti. Utilizzare la chiamata MQCLOSE per eseguire questa operazione.

Utilizzare i seguenti collegamenti per ulteriori informazioni sull'apertura e la chiusura di oggetti:

- “Apertura di oggetti utilizzando la chiamata MQOPEN” a pagina 744
- “Creazione di code dinamiche” a pagina 751
- “Apertura di code remote” a pagina 752
- “Chiusura di oggetti mediante la chiamata MQCLOSE” a pagina 752

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Inserimento di messaggi in una coda” a pagina 753

Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Apertura di oggetti utilizzando la chiamata MQOPEN

Utilizzare queste informazioni per informazioni sull'apertura di oggetti utilizzando la chiamata MQOPEN.

Come input della chiamata MQOPEN, è necessario fornire:

- Un handle di connessione. Per le applicazioni CICS su z/OS, è possibile specificare la costante MQHC_DEF_HCONN (che ha il valore zero) oppure utilizzare l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX. Per altri programmi, utilizzare sempre l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX.
- Una descrizione dell'oggetto che si desidera aprire, utilizzando MQOD (object descriptor structure).
- Una o più opzioni che controllano l'azione della chiamata.

L'output di MQOPEN è:

- Un handle di oggetto che rappresenta il proprio accesso all'oggetto. Utilizzare questa opzione per l'input a tutte le chiamate MQI successive.
- Una struttura descrittore oggetto modificata, se si sta creando una coda dinamica (ed è supportata sulla piattaforma).
- Un codice di completamento.
- Un codice di errore.

Ambito di un handle di oggetto

L'ambito di un handle di oggetto (Hobj) è uguale all'ambito di un handle di connessione (Hconn).

Ciò è trattato in “Ambito di MQCONN o MQCONNX” a pagina 737 e “Connessioni condivise (indipendenti dal thread) con MQCONNX” a pagina 739. Tuttavia, ci sono ulteriori considerazioni in alcuni ambienti:

CICS
In un programma CICS , è possibile utilizzare l'handle solo all'interno della stessa attività CICS da cui è stata effettuata la chiamata MQOPEN.

Batch IMS e z/OS
Negli ambienti IMS e batch, è possibile utilizzare l'handle all'interno della stessa attività, ma non all'interno di alcuna attività secondaria.

Le descrizioni dei parametri della chiamata MQOPEN sono fornite in MQOPEN.

Le seguenti sezioni descrivono le informazioni che è necessario fornire come input per MQOPEN.

Identificazione degli oggetti (la struttura MQOD)

Utilizzare la struttura MQOD per identificare l'oggetto che si desidera aprire. Questa struttura è un parametro di immissione per la chiamata MQOPEN. (La struttura viene modificata dal gestore code quando si utilizza una chiamata MQOPEN per creare una coda dinamica).

Per i dettagli completi sulla struttura MQOD, consultare MQOD.

Per informazioni sull'utilizzo della struttura MQOD per elenchi di distribuzione, consultare “Utilizzo della struttura MQOD” a pagina 764 in “Liste di distribuzione” a pagina 763.

Risoluzione nomi
In che modo la chiamata MQOPEN risolve i nomi di code e gestori code.

Nota: Un alias gestore code è una definizione di coda remota senza un campo RNAME .

Quando si apre una coda IBM MQ , la chiamata MQOPEN esegue una funzione di risoluzione dei nomi sul nome della coda specificato. Ciò determina su quale coda il gestore code esegue operazioni successive. Ciò significa che quando si specifica il nome di una coda alias o di una coda remota nel descrittore oggetto (MQOD), la chiamata risolve il nome in una coda locale o in una coda di trasmissione. Se una coda viene aperta per qualsiasi tipo di input, ricerca o impostazione, si risolve in una coda locale se ne esiste una e non riesce se non ne esiste una. Si risolve in una coda non locale solo se è aperta solo per l'emissione, solo per l'interrogazione o solo per l'emissione e l'interrogazione. Consultare Tabella 114 a pagina 745 per una panoramica del processo di risoluzione dei nomi. Il nome fornito in *ObjectQMgrName* viene risolto *prima* di quello in *ObjectName*.

Tabella 114 a pagina 745 mostra inoltre come utilizzare una definizione locale di coda remota per definire un alias per il nome di un gestore code. Ciò consente di selezionare quale coda di trasmissione viene utilizzata quando si inseriscono i messaggi su una coda remota, in modo da poter, ad esempio, utilizzare una sola coda di trasmissione per i messaggi destinati a molti gestori code remoti.

Per utilizzare la seguente tabella, leggere prima le due colonne di sinistra, sotto l'intestazione **Input to MQOD**, e selezionare il caso appropriato. Quindi leggere la riga corrispondente, seguendo le istruzioni. Seguendo le istruzioni nelle colonne **Nomi risolti** , è possibile tornare alle colonne **Input to MQOD** e inserire i valori come indicato oppure è possibile uscire dalla tabella con i risultati forniti. Ad esempio, potrebbe essere necessario immettere *ObjectName*.

Tabella 114. Risoluzione dei nomi delle code quando si utilizza MQOPEN

Input in MQOD	**Input in MQOD**	**Nomi risolti**	**Nomi risolti**	**Nomi risolti**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
Gestore code locale o vuoto	Coda locale senza attributo CLUSTER	Gestore code locale	Immettere *ObjectName*	Non applicabile (coda locale utilizzata)
Gestore code vuoto	Coda locale con attributo CLUSTER	Gestore code cluster selezionato di gestione del carico di lavoro o gestore code cluster specifico selezionato in PUT	Immettere *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE e coda locale utilizzata SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)
Gestore code locale	Coda locale con attributo CLUSTER	Gestore code locale	Immettere *ObjectName*	Non applicabile (coda locale utilizzata)
Gestore code locale o vuoto	Coda modello	Gestore code locale	Nome generato	Non applicabile (coda locale utilizzata)

Tabella 114. Risoluzione dei nomi delle code quando si utilizza MQOPEN (Continua)

Input in MQOD	**Input in MQOD**	**Nomi risolti**	**Nomi risolti**	**Nomi risolti**
Gestore code locale o vuoto	Coda alias con o senza attributo CLUSTER	Eseguire di nuovo la risoluzione del nome con *ObjectQMgrNome* non modificato e immettere *ObjectName* impostato su *BaseQName* nell'oggetto definizione coda alias. Non deve risolversi in un alias definito localmente in cui è specificato *ObjectQMgrNome* , ma può risolversi in un alias cluster (ospitato su altri gestori code) in cui *ObjectQMgrNome* è vuoto.		
Gestore code locale	Coda alias con attributo CLUSTER	L'alias non deve risolversi in una coda cluster non definita localmente o in una coda cluster con lo stesso *ObjectName* dell'alias.		
Gestore code vuoto	Coda alias con attributo CLUSTER	L'alias può essere risolto in una coda cluster con lo stesso *ObjectName* dell'alias.		
Gestore code locale o vuoto	definizione locale di coda remota	Eseguire di nuovo la risoluzione del nome con *ObjectQMgrName* impostato su *RemoteQMgrName*e *ObjectName* impostato su *RemoteQName*. Non deve risolvere le code remote		Nome dell'attributo *XmitQName* , se non vuoto; altrimenti *RemoteQMgrName* nell'oggetto di definizione della coda remota. SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)
Gestore code vuoto	Nessun oggetto locale corrispondente ; coda cluster trovata	Gestore code cluster selezionato di gestione del carico di lavoro o gestore code cluster specifico selezionato in PUT	Immettere *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)

Tabella 114. Risoluzione dei nomi delle code quando si utilizza MQOPEN (Continua)

Input in MQOD	Input in MQOD	Nomi risolti	Nomi risolti	Nomi risolti
Gestore code locale o vuoto	Nessun oggetto locale corrispondente ; coda cluster non trovata		Errore, coda non trovata	Non applicabile
Nome del gestore code nello stesso gruppo di condivisione code del gestore code locale	Coda condivisa locale	Gestore code locale	Immettere *ObjectName*	Non applicabile
Nome di una coda di trasmissione locale	(Non risolto)	Immettere il nome *ObjectQMgr*	Immettere *ObjectName*	Immettere il nome *ObjectQMgr* SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)
Definizione alias del gestore code (*RemoteQMgrName* può essere il gestore code locale)	(Non risolto, coda remota)	Eseguire nuovamente la risoluzione dei nomi con *ObjectQMgrName* impostato su *RemoteQMgrName*. Non deve essere risolto in code remote	Immettere *ObjectName*	Nome dell'attributo *XmitQName* , se non vuoto; altrimenti *RemoteQMgrName* nell'oggetto di definizione della coda remota. SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)
Il gestore code non è il nome di alcun oggetto locale; sono stati trovati i gestori code del cluster o l'alias del gestore code	(Non risolto)	*ObjectQMgrNome* o gestore code cluster specifico selezionato in PUT	Immettere *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)
Il gestore code non è il nome di alcun oggetto locale; nessun oggetto cluster trovato	(Non risolto)	Immettere il nome *ObjectQMgr*	Immettere *ObjectName*	Attributo *DefXmitQName* del gestore code in cui è supportato *DefXmitQName* . SYSTEM.QSG.TRANSMIT.QUEUE (vedere nota)

Note:

1. *BaseQName* è il nome della coda di base dalla definizione della coda alias.
2. *RemoteQName* è il nome della coda remota dalla definizione locale della coda remota.
3. *RemoteQMgrName* è il nome del gestore code remoto dalla definizione locale della coda remota.
4. *XmitQName* è il nome della coda di trasmissione dalla definizione locale della coda remota.
5. Quando si utilizzano i gestori code IBM MQ for z/OS che fanno parte di un gruppo di condivisione code (QSG), è possibile utilizzare il nome del gruppo di condivisione code invece del nome del gestore code locale in Tabella 114 a pagina 745.

Se il gestore code locale non può aprire la coda di destinazione o inserire un messaggio nella coda, il messaggio viene trasferito al nome ObjectQMgrspecificato tramite l'accodamento all'interno del gruppo o un canale IBM MQ .

6. Nella colonna *ObjectName* della tabella, CLUSTER fa riferimento agli attributi CLUSTER e CLUSNL della coda.
7. Il SISTEMA SYSTEM.QSG.TRANSMIT.QUEUE viene utilizzato se i gestori code locali e remoti si trovano nello stesso gruppo di condivisione code; l'accodamento all'interno del gruppo è abilitato.
8. Se è stata assegnata una diversa coda di trasmissione cluster a ciascun canale mittente del cluster, SYSTEM.CLUSTER.TRANSMIT.QUEUE potrebbe non essere il nome della coda di trasmissione cluster. Per ulteriori informazioni su più code di trasmissione del cluster, consultare Cluster: Pianificazione della configurazione delle code di trasmissione del cluster.
9. Nella situazione in cui il gestore code non è il nome di alcun oggetto locale; sono stati trovati gestori code cluster o alias del gestore code.

   Quando è stato fornito un nome gestore code utilizzando **ObjectQMgrName**e sono presenti più canali cluster con nomi cluster differenti noti al gestore code locale che raggiungerebbero tale destinazione, è possibile che uno di questi canali venga utilizzato per spostare il messaggio, indipendentemente dal nome cluster della coda di destinazione.

   Ciò potrebbe essere imprevisto, se si stavano anticipando i messaggi per quella coda da inviare solo attraverso un canale che ha lo stesso nome cluster della coda.

   Tuttavia, **ObjectQMgrName** ha la precedenza in questo caso e il bilanciamento del carico di lavoro del cluster prende in considerazione tutti i canali che potrebbero raggiungere tale gestore code, indipendentemente dal nome del cluster in cui si trovano.

L'apertura di una coda alias apre anche la coda di base in cui l'alias viene risolto e l'apertura di una coda remota apre anche la coda di trasmissione. Pertanto, non è possibile eliminare la coda specificata o la coda in cui viene risolta mentre l'altra è aperta.

Mentre una coda alias non è in grado di risolvere un'altra coda alias definita localmente (condivisa o meno in un cluster), la risoluzione in una coda alias cluster definita in remoto è consentita e può quindi essere specificata come coda di base.

Il nome della coda risolta e il nome del gestore code risolto vengono memorizzati nei campi *ResolvedQName* e *ResolvedQMgrName* in MQOD.

Per ulteriori informazioni sulla risoluzione dei nomi in un ambiente di accodamento distribuito, consultare What is queue name resolution?.

Utilizzo delle opzioni della chiamata MQOPEN

Nel parametro **Options** della chiamata MQOPEN, è necessario scegliere una o più opzioni per controllare l'accesso fornito all'oggetto che si sta aprendo. Con queste opzioni è possibile:

- Aprire una coda e specificare che tutti i messaggi inseriti in tale coda devono essere indirizzati alla stessa istanza di essa
- Aprire una coda per consentire l'inserimento di messaggi su di essa
- Aprire una coda per consentire di sfogliare i messaggi su di essa
- Aprire una coda per consentire la rimozione dei messaggi da essa
- Aprire un oggetto per consentire di interrogare e impostare i suoi attributi (ma è possibile impostare solo gli attributi delle code)
- Aprire un argomento o una stringa di argomenti per pubblicare i messaggi
- Associa informazioni di contesto a un messaggio
- Designare un identificativo utente alternativo da utilizzare per i controlli di sicurezza
- Controlla la chiamata se il gestore code è in uno stato di inattività

Opzione MQOPEN per la coda cluster
Il bind utilizzato per l'handle della coda viene preso dall'attributo della coda **DefBind** , che può assumere il valore MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDo MQBND_BIND_ON_GROUP.

Per instradare tutti i messaggi inseriti in una coda utilizzando MQPUT allo stesso gestore code tramite lo stesso instradamento, utilizzare l'opzione MQOO_BIND_ON_OPEN sulla chiamata MQOPEN .

Per specificare che una destinazione deve essere selezionata al momento di MQPUT, ossia, su base messaggio per messaggio, utilizzare l'opzione MQOO_BIND_NOT_FIXED sulla chiamata MQOPEN .

Per specificare che tutti i messaggi in un gruppo di messaggi inseriti in una coda utilizzando MQPUT sono assegnati alla stessa istanza di destinazione, utilizzare l'opzione MQOO_BIND_ON_GROUP nella chiamata MQOPEN .

È necessario specificare MQOO_BIND_ON_OPEN o MQOO_BIND_ON_GROUP quando si utilizzano gruppi di messaggi con cluster per garantire che tutti i messaggi nel gruppo vengano elaborati nella stessa destinazione.

Se non si specifica alcuna di queste opzioni, viene utilizzato il valore predefinito, MQOO_BIND_AS_Q_DEF.

Se si specifica il nome di un gestore code in MQOD, viene selezionata la coda in tale gestore code. Se il nome del gestore code è vuoto, è possibile selezionare qualsiasi istanza. Per ulteriori informazioni, fare riferimento a “MQOPEN e cluster” a pagina 887.

Se si apre una coda cluster utilizzando una definizione QALIAS , alcuni attributi della coda vengono definiti dalla coda alias e non dalla coda di base. Gli attributi del cluster sono tra gli attributi della definizione della coda di base sovrascritti dalla coda alias. Ad esempio, nel seguente frammento, la coda cluster viene aperta con MQOO_BIND_NOT_FIXED e non con MQOO_BIND_ON_OPEN. La definizione della coda del cluster viene pubblicizzata in tutto il cluster, la definizione della coda alias è locale per il gestore code.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

Opzione MQOPEN per l'inserimento di messaggi
Per aprire una coda o un argomento in cui inserire i messaggi, utilizzare l'opzione MQOO_OUTPUT.

Opzione MQOPEN per la ricerca dei messaggi
Per aprire una coda in modo da poter *sfogliare* i messaggi su di essa, utilizzare la chiamata MQOPEN con l'opzione MQOO_BROWSE.

Questo crea un *cursore di esplorazione* che il gestore code utilizza per identificare il successivo messaggio sulla coda. Per ulteriori informazioni, consultare “Visualizzazione dei messaggi su una coda” a pagina 803.

Nota:

1. Non è possibile ricercare i messaggi su una coda remota; non aprire una coda remota utilizzando l'opzione MQOO_BROWSE.
2. Non è possibile specificare questa opzione quando si apre un elenco di distribuzione. Per ulteriori informazioni sugli elenchi di distribuzione, consultare “Liste di distribuzione” a pagina 763.
3. Utilizzare MQOO_CO_OP insieme a MQOO_BROWSE se si utilizza la navigazione cooperativa; consultare Opzioni

Opzioni MQOPEN per la rimozione dei messaggi
Tre opzioni controllano l'apertura di una coda per rimuovere i messaggi da essa.

È possibile utilizzarne solo uno in qualsiasi chiamata MQOPEN. Queste opzioni definiscono se il proprio programma ha accesso esclusivo o condiviso alla coda. *Accesso esclusivo* significa che, fino a quando non si chiude la coda, solo è possibile rimuovere i messaggi. Se un altro programma tenta di aprire la coda per rimuovere i messaggi, la chiamata MQOPEN ha esito negativo. *Accesso condiviso* significa che più di un programma può rimuovere dalla coda.

L'approccio più consigliato consiste nell'accettare il tipo di accesso previsto per la coda al momento della definizione della coda. La definizione della coda comportava l'impostazione di **Shareability** e **DefInputOpenOption** Attributi. Per accettarlo, utilizzare l'opzione MQOO_INPUT_AS_Q_DEF. Fare riferimento a Tabella 115 a pagina 750 per informazioni su come l'impostazione di questi attributi influisce sul tipo di accesso che verrà fornito quando si utilizza questa opzione.

Tabella 115. In che modo gli attributi e le opzioni della coda della chiamata MQOPEN influenzano l'accesso alle code

Attributi Coda		**Tipo di accesso con opzioni MQOPEN**		
Shareability	**DefInputOpenOption**	**DEF Q_AS**	**CONDIVISO**	**Esclusivo**
Condivisibile	CONDIVISO	condiviso	condiviso	esclusivo
Condivisibile	Esclusivo	esclusivo	condiviso	esclusivo
NON_CONDIVISIBILE *	SARA*	esclusivo	esclusivo	esclusivo
NON_CONDIVISIBILE	Esclusivo	esclusivo	esclusivo	esclusivo

Nota: * Anche se è possibile definire una coda per avere questa combinazione di attributi, l'opzione di apertura di input predefinita viene sovrascritta dall'attributo di condivisibilità.

In alternativa:

- Se si sa che la propria applicazione può funzionare correttamente anche se altri programmi possono rimuovere i messaggi dalla coda contemporaneamente, utilizzare l'opzione MQOO_INPUT_SHARED. Tabella 115 a pagina 750 mostra come, in alcuni casi, si avrà accesso esclusivo alla coda, anche con questa opzione.
- Se si sa che l'applicazione può funzionare correttamente solo se ad altri programmi viene impedito di rimuovere i messaggi dalla coda contemporaneamente, utilizzare l'opzione MQOO_INPUT_EXCLUSIVE.

Nota:

1. Non è possibile rimuovere messaggi da una coda remota. Pertanto, non è possibile aprire una coda remota utilizzando le opzioni MQOO_INPUT_ *.
2. Non è possibile specificare questa opzione quando si apre un elenco di distribuzione. Per ulteriori informazioni, fare riferimento a “Liste di distribuzione” a pagina 763.

Opzioni MQOPEN per l'impostazione e la richiesta di informazioni sugli attributi
Per aprire una coda in modo da poterne impostare gli attributi, utilizzare l'opzione MQOO_SET.

Non è possibile impostare gli attributi di qualsiasi altro tipo di oggetto (consultare “Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851).

Per aprire un oggetto in modo da poter richiedere informazioni sui relativi attributi, utilizzare l'opzione MQOO_INQUIRE.

Nota: Non è possibile specificare questa opzione quando si apre un elenco di distribuzione.

Opzioni MQOPEN relative al contesto del messaggio
Se si desidera poter associare le informazioni di contesto a un messaggio quando le si inserisce in una coda, è necessario utilizzare una delle opzioni di contesto del messaggio quando si apre la coda.

Le opzioni consentono di distinguere tra le informazioni di contesto relative all' *utente* che ha originato il messaggio e quelle relative all'applicazione che ha originato il messaggio. Inoltre, è possibile scegliere di impostare le informazioni di contesto quando si inserisce il messaggio nella coda oppure è possibile scegliere che il contesto venga acquisito automaticamente da un altro handle della coda.

Concetti correlati

“Contesto messaggio” a pagina 48
Le informazioni relative al *contesto del messaggio* consentono all'applicazione che richiama il messaggio di individuare il creatore del messaggio.

“Controllo delle informazioni di contesto del messaggio” a pagina 760
Quando si utilizza la chiamata MQPUT o MQPUT1 per inserire un messaggio in una coda, è possibile specificare che il gestore code deve aggiungere alcune informazioni di contesto predefinite al descrittore del messaggio. Le applicazioni che dispongono del livello di autorizzazione appropriato possono aggiungere ulteriori informazioni di contesto. È possibile utilizzare il campo opzioni nella struttura MQPMO per controllare le informazioni di contesto.

Opzione MQOPEN per autorizzazione utente alternativa
Quando si tenta di aprire un oggetto utilizzando la chiamata MQOPEN, il gestore code verifica che si disponga dell'autorizzazione per aprire tale oggetto. Se non si dispone dell'autorizzazione, la chiamata ha esito negativo.

Tuttavia, i programmi server potrebbero richiedere al gestore code di controllare l'autorizzazione dell'utente per cui stanno lavorando, piuttosto che l'autorizzazione del server. Per effettuare questa operazione, è necessario utilizzare l'opzione MQOO_ALTERNATE_USER_AUTHORITY della chiamata MQOPEN e specificare l'ID utente alternativo nel campo *AlternateUserId* della struttura MQOD. Generalmente, il server ottiene l'ID utente dalle informazioni di contesto nel messaggio che sta elaborando.

z/OS *L'opzione MQOPEN per la sospensione del gestore code*
Se si utilizza la chiamata MQOPEN quando il gestore code si trova in uno stato di inattività, la chiamata potrebbe non riuscire, a seconda dell'ambiente utilizzato.

Nell'ambiente CICS su z/OS, se si utilizza la chiamata MQOPEN quando il gestore code è in stato di sospensione, la chiamata ha sempre esito negativo.

In altri ambienti z/OS e Multiplatforms, la chiamata ha esito negativo quando il gestore code è in fase di sospensione solo se si utilizza l'opzione MQOO_FAIL_IF_QUIESCING della chiamata MQOPEN.

Opzione MQOPEN per la risoluzione dei nomi delle code locali
Quando si apre una coda locale, alias o modello, viene restituita la coda locale.

Tuttavia, quando si apre una coda remota o una coda cluster, i campi *ResolvedQName* e *ResolvedQMgrName* della struttura MQOD vengono riempiti con i nomi della coda remota e del gestore code remoto trovati nella definizione della coda remota o con la coda cluster remota scelta.

Utilizzare l'opzione MQOO_RESOLVE_LOCAL_Q della chiamata MQOPEN per riempire il *ResolvedQName* nella struttura MQOD con il nome della coda locale aperta. *ResolvedQMgrName* è simile al nome del gestore code locale che ospita la coda locale. Questo campo è disponibile solo con la versione 3 della struttura MQOD; se la struttura è inferiore alla versione 3, MQOO_RESOLVE_LOCAL_Q viene ignorato senza che venga restituito un errore.

Se si specifica MQOO_RESOLVE_LOCAL_Q durante l'apertura, ad esempio, di una coda remota, *ResolvedQName* è il nome della coda di trasmissione in cui verranno inseriti i messaggi. *ResolvedQMgrName* è il nome del gestore code locale che ospita la coda di trasmissione.

Creazione di code dinamiche
Utilizzare una coda dinamica quando non è necessaria la coda dopo il termine dell'applicazione.

Ad esempio, è possibile utilizzare una coda dinamica per la coda di risposta. Specificare il nome della coda di risposta nel campo *ReplyToQ* della struttura MQMD quando si inserisce un messaggio su una coda (consultare “Definizione dei messaggi utilizzando la struttura MQMD” a pagina 755).

Per creare una coda dinamica, utilizzare un modello noto come coda modello, insieme alla chiamata MQOPEN. Creare una coda modello utilizzando i comandi IBM MQ o le operazioni e i pannelli di controllo. La coda dinamica creata prende gli attributi della coda modello.

Quando si richiama MQOPEN, specificare il nome della coda modello nel campo *ObjectName* della struttura MQOD. Una volta completata la chiamata, il campo *ObjectName* viene impostato sul nome della coda dinamica creata. Inoltre, il campo *ObjectQMgrName* è impostato sul nome del gestore code locale.

È possibile specificare il nome della coda dinamica creata in tre modi:

- Fornire il nome completo desiderato nel campo *DynamicQName* della struttura MQOD.
- Specificare un prefisso (meno di 33 caratteri) per il nome e consentire al gestore code di generare il resto del nome. Ciò significa che il gestore code genera un nome univoco, ma si dispone ancora di un certo controllo (ad esempio, è possibile che si desideri che ciascun utente utilizzi un determinato prefisso o che si desideri fornire una classificazione di sicurezza speciale alle code con un determinato prefisso nel proprio nome). Per utilizzare questo metodo, specificare un asterisco (*) per l'ultimo carattere non vuoto del campo *DynamicQName* . Non specificare un singolo asterisco (*) per il nome della coda dinamica.
- Consentire al gestore code di generare il nome completo. Per utilizzare questo metodo, specificare un asterisco (*) nella prima posizione del carattere del campo *DynamicQName* .

Per ulteriori informazioni su questi metodi, consultare la descrizione del campo DynamicQName .

Sono disponibili ulteriori informazioni sulle code dinamiche in Code dinamiche e modello.

Apertura di code remote

Una coda remota è una coda di proprietà di un gestore code diverso da quello a cui è connessa l'applicazione.

Per aprire una coda remota, utilizzare la chiamata MQOPEN come per una coda locale. È possibile specificare il nome della coda nel modo seguente:

1. Nel campo *ObjectName* della struttura MQOD, specificare il nome della coda remota come è noto al gestore code *locale* .

   Nota: Lasciare vuoto il campo *ObjectQMgrName* in questo caso.
2. Nel campo *ObjectName* della struttura MQOD, specificare il nome della coda remota, come noto al gestore code *remoto* . Nel campo *ObjectQMgrName* , specificare:

   - Il nome della coda di trasmissione con lo stesso nome del gestore code remoto. Il nome e il maiuscolo / minuscolo (maiuscolo, minuscolo o misto) devono corrispondere *esattamente*.
   - Il nome di un oggetto alias del gestore code che si risolve nel gestore code di destinazione o nella coda di trasmissione.

   Ciò indica al gestore code la destinazione del messaggio e la coda di trasmissione su cui è necessario inserirlo per ottenerlo.
3. Se *DefXmitQname* è supportato, nel campo *ObjectName* della struttura MQOD, specificare il nome della coda remota come è noto al gestore code *remoto* .

   Nota: Impostare il campo *ObjectQMgrName* sul nome del gestore code remoto (non può essere lasciato vuoto in questo caso).

Solo i nomi locali vengono convalidati quando si richiama MQOPEN; l'ultimo controllo è relativo all'esistenza della coda di trasmissione da utilizzare.

Questi metodi sono riepilogati inTabella 114 a pagina 745.

Chiusura di oggetti mediante la chiamata MQCLOSE

Per chiudere un oggetto, utilizzare la chiamata MQCLOSE.

Se l'oggetto è una coda, tenere presente quanto segue:

- Non è necessario svuotare una coda dinamica temporanea prima di chiuderla.

  Quando si chiude una coda dinamica temporanea, la coda viene eliminata, insieme a tutti i messaggi che potrebbero essere ancora su di essa. Questo è vero anche se ci sono chiamate MQGET, MQPUT o MQPUT1 non sottoposte a commit in sospeso rispetto alla coda.
- Su IBM MQ for z/OS, se si dispone di richieste MQGET con un'opzione MQGMO_SET_SIGNAL in sospeso per tale coda, vengono annullate.
- Se la coda è stata aperta utilizzando l'opzione MQOO_BROWSE, il cursore di ricerca viene distrutto.

La chiusura non è correlata al punto di sincronizzazione, quindi è possibile chiudere le code prima o dopo il punto di sincronizzazione.

Come input per la chiamata MQCLOSE, è necessario fornire:

- Un handle di connessione. Utilizzare lo stesso handle di connessione utilizzato per aprirlo o, in alternativa, per le applicazioni CICS su z/OS, è possibile specificare la costante MQHC_DEF_HCONN (che ha il valore zero).
- L'handle dell'oggetto che si desidera chiudere. Acquisirlo dall'output della chiamata MQOPEN.
- MQCO_NONE nel campo *Options* (a meno che non si stia chiudendo una coda dinamica permanente).
- L'opzione di controllo per determinare se il gestore code deve eliminare la coda anche se contiene ancora messaggi (quando si chiude una coda dinamica permanente).

L'output di MQCLOSE è:

- Un codice di completamento
- Un codice di errore
- L'handle dell'oggetto, reimpostato sul valore MQHO_UNUSABLE_HOBJ

Le descrizioni dei parametri della chiamata MQCLOSE sono fornite in MQCLOSE.

Inserimento di messaggi in una coda

Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

Utilizzare la chiamata MQPUT per inserire i messaggi nella coda. È possibile utilizzare MQPUT ripetutamente per inserire molti messaggi nella stessa coda, seguendo la chiamata MQOPEN iniziale. Richiamare MQCLOSE una volta terminato l'inserimento di tutti i messaggi nella coda.

Se si desidera inserire un singolo messaggio in una coda e chiudere la coda immediatamente dopo, è possibile utilizzare la chiamata MQPUT1 . MQPUT1 esegue le stesse funzioni della seguente sequenza di chiamate:

- MQOPEN
- MQPUT
- MQCLOSE

In genere, tuttavia, se si dispone di più di un messaggio da inserire nella coda, è più efficiente utilizzare la chiamata MQPUT. Ciò dipende dalla dimensione del messaggio e dalla piattaforma su cui si sta lavorando.

Utilizzare i seguenti collegamenti per ulteriori informazioni sull'inserimento di messaggi in una coda:

- “Inserimento di messaggi su una coda locale utilizzando la chiamata MQPUT” a pagina 754
- “Inserimento di messaggi su una coda remota” a pagina 759
- “Impostazione delle proprietà di un messaggio” a pagina 759
- “Controllo delle informazioni di contesto del messaggio” a pagina 760
- “Inserimento di un messaggio su una coda utilizzando la chiamata MQPUT1” a pagina 762
- “Liste di distribuzione” a pagina 763
- “Alcuni casi in cui le chiamate put hanno esito negativo” a pagina 768

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Inserimento di messaggi su una coda locale utilizzando la chiamata MQPUT

Utilizzare queste informazioni per informazioni sull'inserimento di messaggi su una coda locale utilizzando la chiamata MQPUT.

Come input per la chiamata MQPUT, è necessario fornire:

- Un handle di connessione (Hconn).
- Un gestore code (Hobj).
- Una descrizione del messaggio che si desidera inserire nella coda. È sotto forma di una struttura del descrittore del messaggio (MQMD).
- Informazioni di controllo, sotto forma di MQPMO (put - message options structure).
- La lunghezza dei dati contenuti nel messaggio (MQLONG).
- I dati del messaggio.

L'output della chiamata MQPUT è il seguente:

- Un codice motivo (MQLONG)
- Un codice di completamento (MQLONG)

Se la chiamata viene completata correttamente, restituisce anche la struttura delle opzioni e la struttura del descrittore del messaggio. La chiamata modifica la propria struttura di opzioni per visualizzare il nome della coda e il gestore code a cui è stato inviato il messaggio. Se si richiede che il gestore code generi un valore univoco per l'identificativo del messaggio che si sta immettendo (specificando zero binario nel campo *MsgId* della struttura MQMD), la chiamata inserisce il valore nel campo *MsgId* prima di restituire questa struttura all'utente. Reimposta questo valore prima di emettere un altro MQPUT.

Esiste una descrizione della chiamata MQPUT in MQPUT.

Per ulteriori informazioni sulle informazioni necessarie come input per la chiamata MQPUT, consultare i seguenti link:

- “Specifica di handle” a pagina 755
- “Definizione dei messaggi utilizzando la struttura MQMD” a pagina 755
- “Specifica delle opzioni utilizzando la struttura MQPMO” a pagina 755
- “I dati nel messaggio” a pagina 758
- “Inserimento dei messaggi: utilizzo degli handle dei messaggi” a pagina 759

Specifica di handle

Per l'handle di connessione (*Hconn*) in CICS su applicazioni z/OS , è possibile specificare la costante MQHC_DEF_HCONN (che ha il valore zero) oppure è possibile utilizzare l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX. Per altre applicazioni, utilizzare sempre l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX.

Indipendentemente dall'ambiente in cui si sta lavorando, utilizzare lo stesso handle di coda (*Hobj*) restituito dalla chiamata MQOPEN.

Definizione dei messaggi utilizzando la struttura MQMD

MQMD (message descriptor structure) è un parametro di input / output per le chiamate MQPUT e MQPUT1 . Utilizzarlo per definire il messaggio che si sta inserendo in una coda.

Se MQPRI_PRIORITY_AS_Q_DEF o MQPER_PERSISTENCE_AS_Q_DEF è specificato per il messaggio e la coda è una coda cluster, i valori utilizzati sono quelli della coda in cui MQPUT si risolve. Se la coda è disabilitata per MQPUT, la chiamata avrà esito negativo. Per ulteriori informazioni, consultare Configurazione di un cluster di gestori code .

Nota: Utilizzare MQPMO_NEW_MSG_ID e MQPMO_NEW_CORREL_ID prima di inserire un nuovo messaggio per assicurarsi che *MsgId* e *CorrelId* siano univoci. I valori in questi campi vengono restituiti in un MQPUT riuscito.

È disponibile un'introduzione alle proprietà del messaggio descritte da MQMD in “IBM MQ messaggi” a pagina 18e una descrizione della struttura stessa in MQMD.

Specifica delle opzioni utilizzando la struttura MQPMO

Utilizzare la struttura MQPMO (Put Message Option) per passare opzioni alle chiamate MQPUT e MQPUT1 .

Le seguenti sezioni forniscono assistenza per compilare i campi di questa struttura. C'è una descrizione della struttura in MQPMO.

La struttura include i seguenti campi:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

Il contenuto di questi campi è il seguente:

StrucId
Identifica la struttura come una struttura di opzioni put - message. Questo è un campo di 4 caratteri. Specificare sempre MQPMO_STRUC_ID.

Versione
Descrive il numero di versione della struttura. Il valore predefinito è MQPMO_VERSION_1. Se si immette MQPMO_VERSION_2, è possibile utilizzare gli elenchi di distribuzione (consultare "Liste di distribuzione" a pagina 763). Se si immette MQPMO_VERSION_3, è possibile utilizzare gli handle del messaggio e le proprietà del messaggio. Se si immette MQPMO_CURRENT_VERSION, l'applicazione è sempre impostata per utilizzare il livello più recente.

Opzioni
Ciò controlla quanto segue:

- Se l'operazione di inserimento è inclusa in un'unità di lavoro
- Quante informazioni di contesto sono associate a un messaggio
- Da dove vengono prese le informazioni di contesto
- Indica se la chiamata ha esito negativo se il gestore code è in stato di sospensione
- Se il raggruppamento o la segmentazione sono consentiti
- Creazione di un nuovo identificativo di messaggio e di correlazione
- L'ordine in cui i messaggi e i segmenti vengono inseriti in coda
- Indica se risolvere i nomi delle code locali

Se si lascia il campo *Options* impostato sul valore predefinito (MQPMO_NONE), al messaggio inserito sono associate informazioni di contesto predefinite.

Inoltre, il modo in cui la chiamata opera con i punti di sincronizzazione è determinato dalla piattaforma. Il valore predefinito del controllo del punto di sincronizzazione è yes in z/OS ; per altre piattaforme, è no.

Contesto
Indica il nome dell'handle della coda da cui si desidera copiare le informazioni di contesto (se richiesto nel campo *Options*).

Per un'introduzione al contesto del messaggio, consultare "Contesto messaggio" a pagina 48. Per informazioni sull'utilizzo della struttura MQPMO per controllare le informazioni di contesto in un messaggio, consultare "Controllo delle informazioni di contesto del messaggio" a pagina 760.

ResolvedQName
Contiene il nome (dopo la risoluzione di qualsiasi nome alias) della coda che è stata aperta per ricevere il messaggio. Questo è un campo di output.

Nome ResolvedQMgr
Contiene il nome (dopo la risoluzione di qualsiasi nome alias) del gestore code proprietario della coda in *ResolvedQName*. Questo è un campo di output.

MQPMO può contenere anche i campi richiesti per gli elenchi di distribuzione (consultare "Liste di distribuzione" a pagina 763). Se si desidera utilizzare questa funzionalità, viene utilizzata la versione 2 della struttura MQPMO. Sono inclusi i seguenti campi:

RecsPresent
Questo campo contiene il numero di code nell'elenco di distribuzione, ossia il numero di MQPMR (Put Message Records) e MQRR (Response Records) corrispondenti.

Il valore immesso può essere lo stesso del numero di record oggetto fornito in MQOPEN. Tuttavia, se il valore è inferiore al numero di record oggetto forniti nella chiamata MQOPEN o se non si fornisce alcun record di inserimento messaggi, i valori delle code non definite vengono presi dai valori predefiniti forniti dal descrittore del messaggio. Inoltre, se il valore è maggiore del numero di record oggetto forniti, i record di messaggi immessi in eccesso vengono ignorati.

Si consiglia di effettuare una delle seguenti operazioni:

- Se si desidera ricevere un report o una risposta da ciascuna destinazione, immettere lo stesso valore visualizzato nella struttura MQOR e utilizzare MQPMR contenenti campi *MsgId* . Inizializzare questi campi *MsgId* su zeri oppure specificare MQPMO_NEW_MSG_ID.

Una volta inserito il messaggio nella coda, i valori *MsgId* che il gestore code ha creato diventano disponibili in MQPMR; è possibile utilizzarli per identificare quale destinazione è associata a ciascun report o risposta.

- Se non si desidera ricevere report o risposte, scegliere una delle seguenti opzioni:

  1. Se si desidera identificare le destinazioni che hanno esito negativo immediatamente, è comunque possibile immettere lo stesso valore nel campo *RecsPresent* visualizzato nella struttura MQOR e fornire MQRR per identificare tali destinazioni. Non specificare alcun MQPMR.
  2. Se non si desidera identificare le destinazioni non riuscite, immettere zero nel campo *RecsPresent* e non fornire MQPMR né MQRR.

Nota: Se si sta utilizzando MQPUT1, il numero di Puntatori del record di risposta e Offset del record di risposta deve essere zero.

Per una descrizione completa di MQPMR (Put Message Records) e MQRR (Response Records), consultare MQPM e MQRR.

PutMsgRecFields
Indica quali campi sono presenti in ciascun record MQPMR (Put Message Record). Per un elenco di questi campi, consultare "Utilizzo della struttura MQPMR" a pagina 767.

PutMsgRecOffset* e *PutMsgRecPtr
I puntatori (di solito in C) e gli offset (di solito in COBOL) vengono utilizzati per indirizzare i record Put Message (consultare "Utilizzo della struttura MQPMR" a pagina 767 per una panoramica della struttura MQPMR).

Utilizzare il campo *PutMsgRecPtr* per specificare un puntatore al primo record di inserimento messaggi o il campo *PutMsgRecOffset* per specificare lo scostamento del primo record di inserimento messaggi. Questo è l'offset dall'inizio di MQPM. A seconda del campo *PutMsgRecFields* , immettere un valore non null per *PutMsgRecOffset* o *PutMsgRecPtr*.

Offset ResponseRece Ptr ResponseRec
Utilizzare anche i puntatori e gli offset per indirizzare i record di risposta (consultare "Utilizzo della struttura MQRR" a pagina 766 per ulteriori informazioni sui record di risposta).

Utilizzare il campo *ResponseRecPtr* per specificare un puntatore al primo record di risposta oppure il campo *ResponseRecOffset* per specificare lo scostamento del primo record di risposta. Questo è l'offset dall'inizio della struttura MQPMO. Immettere un valore non null per *ResponseRecOffset* o *ResponseRecPtr*.

Nota: Se si utilizza MQPUT1 per inserire i messaggi in un elenco di distribuzione, *ResponseRecPtr* deve essere null o zero e *ResponseRecOffset* deve essere zero.

La Versione 3 della struttura di MQPMO include anche i seguenti campi:

Handle OriginalMsg
L'utilizzo di questo campo dipende dal valore del campo *Azione* . Se si sta inserendo un nuovo messaggio con le proprietà del messaggio associate, impostare questo campo sull'handle del messaggio precedentemente creato e impostare le proprietà su. Se si sta inoltrando, rispondendo o generando un report in risposta a un messaggio precedentemente richiamato, questo campo contiene l'handle del messaggio di tale messaggio.

NewMsgHandle
Se si specifica un *NewMsgHandle*, tutte le proprietà associate all'handle sovrascrivono le proprietà associate all' *OriginalMsgHandle*. Per ulteriori informazioni, consultare Azione (MQLONG).

Azione
Utilizzare questo campo per specificare il tipo di immissione da eseguire. I valori possibili e i relativi significati sono i seguenti:

NUOVO
Questo è un nuovo messaggio non correlato ad altri.

FORWARD MQACTP
Questo messaggio è stato richiamato in precedenza ed è ora in fase di inoltro.

MQACTP_REPLY
Questo messaggio è una risposta ad un messaggio precedentemente richiamato.

PROSPETTO MQACTP_REPORT
Questo messaggio è un report generato come risultato di un messaggio precedentemente richiamato.

Per ulteriori informazioni, consultare Azione (MQLONG).

PubLevel
Se questo messaggio è una pubblicazione, è possibile impostare questo campo per determinare quali sottoscrizioni lo ricevono. Solo le sottoscrizioni con un *SubLevel* inferiore o uguale a questo valore riceveranno questa pubblicazione. Il valore predefinito è 9, che è il livello più alto e significa che le sottoscrizioni con qualsiasi *SubLevel* possono ricevere questa pubblicazione.

I dati nel messaggio

Fornire l'indirizzo del buffer che contiene i dati nel parametro **Buffer** della chiamata MQPUT. È possibile includere qualsiasi cosa nei dati nei messaggi. La quantità di dati nei messaggi, tuttavia, influenza le prestazioni dell'applicazione che li sta elaborando.

La dimensione massima dei dati è determinata da:

- L'attributo **MaxMsgLength** del gestore code
- L'attributo **MaxMsgLength** della coda in cui si sta inserendo il messaggio
- La dimensione di qualsiasi intestazione del messaggio aggiunta da IBM MQ (inclusa l'intestazione dead-letter, MQDLH e l'intestazione dell'elenco di distribuzione, MQDH)

L'attributo **MaxMsgLength** del gestore code contiene la dimensione del messaggio che il gestore code può elaborare. Ha un valore predefinito di 100 MB per tutti i prodotti IBM MQ alla versione V6 o superiore.

Per determinare il valore di questo attributo, utilizzare la chiamata MQINQ sull'oggetto gestore code. Per messaggi di grandi dimensioni, è possibile modificare questo valore.

L'attributo **MaxMsgLength** di una coda determina la dimensione massima del messaggio che è possibile inserire sulla coda. Se si tenta di inserire un messaggio con una dimensione maggiore del valore di questo attributo, la chiamata MQPUT ha esito negativo. Se si sta inserendo un messaggio su una coda remota, la dimensione massima del messaggio che è possibile inserire correttamente è determinata dall'attributo **MaxMsgLength** della coda remota, da tutte le code di trasmissione intermedie su cui il messaggio viene inserito lungo l'instradamento alla relativa destinazione e dai canali utilizzati.

Per un'operazione MQPUT, la dimensione del messaggio deve essere inferiore o uguale all'attributo **MaxMsgLength** della coda e del gestore code. I valori di questi attributi sono indipendenti, ma si consiglia di impostare il *MaxMsgLength* della coda su un valore inferiore o uguale a quello del gestore code.

IBM MQ aggiunge le informazioni di intestazione ai messaggi nelle circostanze seguenti:

- Quando si inserisce un messaggio su una coda remota, IBM MQ aggiunge una struttura di intestazioni di trasmissione (MQXQH) al messaggio. Questa struttura comprende il nome della coda di destinazione e il suo gestore code proprietario.
- Se IBM MQ non è in grado di consegnare un messaggio a una coda remota, tenta di inserire il messaggio nella coda dei messaggi non recapitabili (messaggio non recapitabile). Aggiunge una struttura MQDLH al messaggio. Questa struttura comprende il nome della coda di destinazione e il motivo per cui il messaggio è stato inserito nella coda di messaggi non instradabili.
- Se si desidera inviare un messaggio a più code di destinazione, IBM MQ aggiunge un'intestazione MQDH al messaggio. Descrive i dati presenti in un messaggio, appartenenti a un elenco di distribuzione, su una coda di trasmissione. Considerare questa opzione quando si sceglie un valore ottimale per la lunghezza massima del messaggio.
- Se il messaggio è un segmento o un messaggio in un gruppo, IBM MQ potrebbe aggiungere un MQMDE.

Queste strutture sono descritte in MQDH e MQMDE.

Se i messaggi sono della dimensione massima consentita per queste code, l'aggiunta di queste intestazioni significa che le operazioni di inserimento hanno esito negativo perché i messaggi sono ora troppo grandi. Per ridurre la possibilità che le operazioni di inserimento non funzionino:

- Ridurre la dimensione dei messaggi rispetto all'attributo **MaxMsgLength** delle code di trasmissione e di messaggi non recapitabili. Consentire almeno il valore della costante MQ_MSG_HEADER_LENGTH (più per elenchi di distribuzione di grandi dimensioni).
- Assicurarsi che l'attributo **MaxMsgLength** della coda di messaggi non instradabili sia impostato sullo stesso valore di *MaxMsgLength* del gestore code proprietario della coda di messaggi non instradabili.

Gli attributi per il gestore code e le costanti di accodamento dei messaggi sono descritti in Attributi per il gestore code.

z/OS Per informazioni su come vengono gestiti i messaggi non consegnati in un ambiente di accodamento distribuito, consultare Messaggi non consegnati/non elaborati.

Inserimento dei messaggi: utilizzo degli handle dei messaggi

Sono disponibili due handle del messaggio nella struttura MQPM, *OriginalMsgHandle* e *NewMsgHandle*. La relazione tra questi handle del messaggio viene definita dal valore del campo *Azione* MQPMO.

Per dettagli completi, consultare Azione (MQLONG). Un handle del messaggio non è necessariamente richiesto per inserire un messaggio. Il suo scopo è quello di associare le proprietà a un messaggio, quindi è richiesto solo se si utilizzano le proprietà del messaggio.

Inserimento di messaggi su una coda remota

Quando si desidera inserire un messaggio su una coda remota (ossia, una coda di un gestore code diverso da quello a cui è connessa l'applicazione) piuttosto che su una coda locale, l'unica considerazione aggiuntiva è il modo in cui si specifica il nome della coda quando viene aperta.Ciò è descritto in “Apertura di code remote” a pagina 752. Non vi sono modifiche al modo in cui si utilizza la chiamata MQPUT o MQPUT1 per una coda locale.

Per ulteriori informazioni sull'utilizzo delle code remote e di trasmissione, consultare Tecniche di accodamento distribuito IBM MQ.

Impostazione delle proprietà di un messaggio

Richiamare MQSETMP per ogni proprietà che si desidera impostare. Quando si inserisce la serie di messaggi, l'handle del messaggio e i relativi campi di azione della struttura MQPMO.

Per associare le proprietà ad un messaggio, il messaggio deve disporre di un handle del messaggio. Creare un handle del messaggio utilizzando la chiamata alla funzione MQCRTMH. Richiamare MQSETMP specificando questo handle del messaggio per ciascuna proprietà che si desidera impostare. Un programma di esempio, amqsstma.c, viene fornito per illustrare l'utilizzo di MQSETMP.

Se si tratta di un nuovo messaggio, quando lo si inserisce in una coda, utilizzando MQPUT o MQPUT1, impostare il campo Handle OriginalMsgin MQPMO sul valore di questo handle del messaggio e impostare il campo Azione MQPMO su MQACTP_NEW (questo è il valore predefinito).

Se si tratta di un messaggio richiamato in precedenza e si sta ora inoltrando o rispondendo ad esso o inviando un report in risposta ad esso, inserire l'handle del messaggio originale nel campo Handle OriginalMsgdi MQPMO e il nuovo handle del messaggio nel campo Handle NewMsg. Impostare il campo Azione su MQACTP_FORWARD, MQACTP_REPLY o MQACTP_REPORT, come appropriato.

Se si dispone di proprietà in un'intestazione MQRFH2 da un messaggio precedentemente richiamato, è possibile convertirle in proprietà di gestione messaggi utilizzando la chiamata MQBUFMH.

Se si sta inserendo il messaggio in una coda su un gestore code a un livello precedente a IBM WebSphere MQ 7.0, che non può elaborare le proprietà del messaggio, è possibile impostare il parametro PropertyControl nella definizione del canale per specificare come devono essere trattate le proprietà.

Controllo delle informazioni di contesto del messaggio

Quando si utilizza la chiamata MQPUT o MQPUT1 per inserire un messaggio in una coda, è possibile specificare che il gestore code deve aggiungere alcune informazioni di contesto predefinite al descrittore del messaggio. Le applicazioni che dispongono del livello di autorizzazione appropriato possono aggiungere ulteriori informazioni di contesto. È possibile utilizzare il campo opzioni nella struttura MQPMO per controllare le informazioni di contesto.

Le informazioni di contesto del messaggio consentono all'applicazione che richiama il messaggio di individuare il creatore del messaggio. Tutte le informazioni di contesto vengono memorizzate nei campi di contesto del descrittore del messaggio. Il tipo di informazioni ricade nelle informazioni di identità, origine e contesto utente.

Per controllare le informazioni di contesto, utilizzare il campo *Options* nella struttura MQPMO.

Se non si specificano le opzioni per le informazioni di contesto, il gestore code sovrascrive le informazioni di contesto che potrebbero già trovarsi nel descrittore del messaggio con le informazioni di identità e di contesto generate per il messaggio. Ciò equivale a specificare l'opzione MQPMO_DEFAULT_CONTEXT.Si potrebbero desiderare queste informazioni di contesto predefinite quando si crea un nuovo messaggio (ad esempio, quando si elabora l'input dell'utente da una schermata di interrogazione).

Se non si desidera che le informazioni di contesto siano associate al messaggio, utilizzare l'opzione MQPMO_NO_CONTEXT. Quando si inserisce un messaggio senza contesto, i controlli di autorizzazione eseguiti da IBM MQ vengono eseguiti utilizzando un ID utente vuoto. Un ID utente vuoto non può essere assegnato all'autorizzazione esplicita per le risorse IBM MQ , ma viene considerato come membro del gruppo speciale 'nobody'. Per ulteriori dettagli sul gruppo speciale nobody, consultare Installable services interface reference information.

È possibile eseguire l'impostazione del contesto utilizzando MQOPEN seguito da MQPUT utilizzando le opzioni MQOO_ e MQPMO_ indicate nelle seguenti sezioni. È anche possibile eseguire l'impostazione del contesto utilizzando solo MQPUT1, nel qual caso è necessario selezionare l'opzione MQPMO_ indicata nelle sezioni riportate di seguito.

Le seguenti sezioni di questo argomento spiegano l'uso del contesto identità, del contesto utente e di tutto il contesto.

- “Passaggio contesto identità” a pagina 760
- “Passaggio del contesto utente” a pagina 761
- “Passaggio di tutti i contesti” a pagina 761
- “Impostazione contesto identità” a pagina 761
- “Impostazione contesto utente” a pagina 761
- “Impostazione di tutti i contesti” a pagina 762

Passaggio contesto identità

In generale, i programmi devono trasmettere le informazioni sul contesto di identità da messaggio a messaggio intorno a una applicazione fino a quando i dati non raggiungono la destinazione finale.

I programmi devono modificare le informazioni sul contesto di origine ogni volta che modificano i dati. Tuttavia, le applicazioni che desiderano modificare o impostare le informazioni di contesto devono disporre del livello di autorizzazione appropriato. Il gestore code controlla questa autorizzazione quando le applicazioni aprono le code; devono disporre dell'autorizzazione per utilizzare le opzioni di contesto appropriate per la chiamata MQOPEN.

Se l'applicazione riceve un messaggio, elabora i dati dal messaggio, quindi inserisce i dati modificati in un altro messaggio (possibilmente per l'elaborazione da parte di un'altra applicazione), l'applicazione deve passare le informazioni del contesto di identità dal messaggio originale al nuovo messaggio. È possibile consentire al gestore code di creare le informazioni sul contesto di origine.

Per salvare le informazioni di contesto dal messaggio originale, utilizzare l'opzione MQOO_SAVE_ALL_CONTEXT quando si apre la coda per ottenere il messaggio. Questo è in aggiunta a

tutte le altre opzioni che si utilizzano con la chiamata MQOPEN. Notare, tuttavia, che non è possibile salvare le informazioni di contesto se si sfoglia solo il messaggio.

Quando si crea il secondo messaggio:

- Aprire la coda utilizzando l'opzione MQOO_PASS_IDENTITY_CONTEXT (in aggiunta all'opzione MQOO_OUTPUT).
- Nel campo *Context* della struttura di opzioni di inserimento del messaggio, fornire l'handle della coda da cui sono state salvate le informazioni di contesto.
- Nel campo *Options* della struttura delle opzioni put - message, specificare l'opzione MQPMO_PASS_IDENTITY_CONTEXT.

Passaggio del contesto utente

Non è possibile scegliere di passare solo il contesto utente. Per passare il contesto utente durante l'inserimento di un messaggio, specificare MQPMO_PASS_ALL_CONTEXT. Tutte le proprietà nel contesto utente vengono trasmesse nello stesso modo del contesto di origine.

Quando si verifica un MQPUT o MQPUT1 e il contesto viene passato, tutte le proprietà nel contesto utente vengono trasmesse dal messaggio richiamato al messaggio di inserimento. Tutte le proprietà del contesto utente che l'applicazione di inserimento ha modificato vengono inserite con i relativi valori originali. Tutte le proprietà del contesto utente che l'applicazione di inserimento ha eliminato vengono ripristinate nel messaggio di inserimento. Tutte le proprietà del contesto utente che l'applicazione di inserimento ha aggiunto al messaggio vengono conservate.

Passaggio di tutti i contesti

Se l'applicazione riceve un messaggio e inserisce i dati del messaggio (non modificati) in un altro messaggio, l'applicazione deve trasmettere tutte le informazioni di contesto (identità, origine e utente) dal messaggio originale al nuovo messaggio. Un esempio di un'applicazione che potrebbe eseguire questa operazione è un programma di spostamento messaggi, che sposta i messaggi da una coda all'altra.

Seguire la stessa procedura utilizzata per il trasferimento del contesto di identità, ad eccezione del fatto che si utilizzano le opzioni MQOPEN MQOO_PASS_ALL_CONTEXT e put - message MQPMO_PASS_ALL_CONTEXT.

Impostazione contesto identità

Se si desidera impostare le informazioni sul contesto di identità per un messaggio:

- Aprire la coda utilizzando l'opzione MQOO_SET_IDENTITY_CONTEXT.
- Inserire il messaggio nella coda, specificando l'opzione MQPMO_SET_IDENTITY_CONTEXT. Nel descrittore del messaggio, specificare le informazioni di contesto di identità richieste.

Nota: Quando si impostano alcuni (ma non tutti) dei campi del contesto di identità utilizzando le opzioni MQOO_SET_IDENTITY_CONTEXT e MQPMO_SET_IDENTITY_CONTEXT, è importante rendersi conto che il gestore code non imposta nessuno degli altri campi.

Per modificare le opzioni di contesto del messaggio, è necessario disporre delle autorizzazioni appropriate per emettere la chiamata. Ad esempio, per utilizzare MQOO_SET_IDENTITY_CONTEXT o MQPMO_SET_IDENTITY_CONTEXT, è necessario disporre dell'autorizzazione `+setid` .

Impostazione contesto utente

Per impostare una proprietà nel contesto utente, impostare il campo Contesto di MQPD (message property descriptor) su MQPD_USER_CONTEXT quando si effettua la chiamata MQSETMP.

Non è necessaria alcuna autorizzazione speciale per impostare una proprietà nel contesto utente. Il contesto utente non ha opzioni di contesto MQOO_SET_ * o MQPMO_SET_ *.

Impostazione di tutti i contesti

Se si desidera impostare sia l'identità che le informazioni sul contesto di origine per un messaggio:

1. Aprire la coda utilizzando l'opzione MQOO_SET_ALL_CONTEXT.
2. Inserire il messaggio nella coda, specificando l'opzione MQPMO_SET_ALL_CONTEXT. Nel descrittore del messaggio, specificare le informazioni di identità e contesto di origine richieste.

Per ogni tipo di impostazione di contesto è necessaria l'autorizzazione appropriata.

Concetti correlati

“Contesto messaggio” a pagina 48
Le informazioni relative al *contesto del messaggio* consentono all'applicazione che richiama il messaggio di individuare il creatore del messaggio.

Riferimenti correlati

“Opzioni MQOPEN relative al contesto del messaggio” a pagina 750
Se si desidera poter associare le informazioni di contesto a un messaggio quando le si inserisce in una coda, è necessario utilizzare una delle opzioni di contesto del messaggio quando si apre la coda.

Inserimento di un messaggio su una coda utilizzando la chiamata MQPUT1

Utilizzare la chiamata MQPUT1 quando si desidera chiudere la coda immediatamente dopo aver inserito un messaggio singolo. Ad esempio, un'applicazione server probabilmente utilizzerà la chiamata MQPUT1 quando invia una risposta a ognuna delle diverse code.

MQPUT1 è funzionalmente equivalente alla chiamata MQOPEN seguita da MQPUT, seguita da MQCLOSE. L'unica differenza nella sintassi per chiamate MQPUT e MQPUT1 è che per MQPUT si specifica un handle di oggetto, mentre per MQPUT1 si specifica una struttura del descrittore di oggetti (MQOD) come definito in MQOPEN (consultare “Identificazione degli oggetti (la struttura MQOD)” a pagina 745). Ciò è dovuto al fatto che è necessario fornire informazioni alla chiamata MQPUT1 sulla coda che deve aprire, mentre quando si richiama MQPUT, la coda deve essere già aperta.

Come input per la chiamata MQPUT1 , è necessario fornire:

- Un handle di connessione.
- Una descrizione dell'oggetto che si desidera aprire. È sotto forma di MQOD (object descriptor structure).
- Una descrizione del messaggio che si desidera inserire nella coda. È sotto forma di una struttura del descrittore del messaggio (MQMD).
- Controllare le informazioni sotto forma di MQPMO (put - message options structure).
- La lunghezza dei dati contenuti nel messaggio (MQLONG).
- L'indirizzo dei dati del messaggio.

L'output da MQPUT1 è:

- Un codice di completamento
- Un codice di errore

Se la chiamata viene completata correttamente, restituisce anche la struttura delle opzioni e la struttura del descrittore del messaggio. La chiamata modifica la propria struttura di opzioni per visualizzare il nome della coda e il gestore code a cui è stato inviato il messaggio. Se si richiede che il gestore code generi un valore univoco per l'identificativo del messaggio che si sta immettendo (specificando zero binario nel campo *MsgId* della struttura MQMD), la chiamata inserisce il valore nel campo *MsgId* prima di restituire questa struttura all'utente.

Nota: Non è possibile utilizzare MQPUT1 con un nome coda modello; tuttavia, una volta aperta una coda modello, è possibile immettere MQPUT1 per la coda dinamica.

I sei parametri di input per MQPUT1 sono:

Hconn
Questo è un handle di connessione. Per le applicazioni CICS , è possibile specificare la costante MQHC_DEF_HCONN (che ha il valore zero) oppure utilizzare l'handle di connessione restituito dalla

chiamata MQCONN o MQCONNX. Per altri programmi, utilizzare sempre l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX.

ObjDesc
Questa è una struttura descrittore oggetto (MQOD).

Nei campi *ObjectName* e *ObjectQMgrName* , specificare il nome della coda in cui si desidera inserire un messaggio e il nome del gestore code proprietario di questa coda.

Il campo *DynamicQName* viene ignorato per la chiamata MQPUT1 perché non può utilizzare le code modello.

Utilizzare il campo *AlternateUserId* se si desidera denominare un identificativo utente alternativo da utilizzare per verificare l'autorizzazione ad aprire la coda.

MsgDesc
Questa è una struttura del descrittore del messaggio (MQMD). Come con la chiamata MQPUT, utilizzare questa struttura per definire il messaggio che si sta inserendo sulla coda.

PutMsgOpts
Si tratta di una struttura di opzioni di inserimento messaggi (MQPMO). Utilizzarlo come per la chiamata MQPUT (consultare “Specifica delle opzioni utilizzando la struttura MQPMO” a pagina 755).

Quando il campo *Options* è impostato a 0, il gestore code utilizza il proprio ID utente quando esegue verifiche per l'autorizzazione ad accedere alla coda. Inoltre, il gestore code ignora qualsiasi identificativo utente alternativo fornito nel campo *AlternateUserId* della struttura MQOD.

BufferLength
Questa è la lunghezza del messaggio.

Buffer
Questa è l'area di buffer che contiene il testo del messaggio.

Quando si utilizzano i cluster, MQPUT1 funziona come se MQOO_BIND_NOT_FIXED fosse attivo. Le applicazioni devono utilizzare i campi risolti nella struttura MQPMO piuttosto che la struttura MQOD per determinare dove è stato inviato il messaggio. Per ulteriori informazioni, consultare Configurazione di un cluster di gestori code .

Esiste una descrizione della chiamata MQPUT1 in MQPUT1.

Liste di distribuzione

Non supportato su IBM MQ for z/OS. Gli elenchi di distribuzione consentono di inserire un messaggio in più destinazioni in una singola chiamata MQPUT o MQPUT1 .Una singola chiamata MQOPEN può aprire più code e una singola chiamata MQPUT può quindi inserire un messaggio in ognuna di queste code. Alcune informazioni generiche dalle strutture MQI utilizzate per questo processo possono essere sostituite da informazioni specifiche relative alle singole destinazioni incluse nell'elenco di distribuzione.

Attenzione: Gli elenchi di distribuzione non supportano l'utilizzo di code alias che puntano agli oggetti argomento. Se una coda alias punta a un oggetto argomento in un elenco di distribuzione, IBM MQ restituisce MQRC_ALIAS_BASE_Q_TYPE_ERROR.

Quando viene emessa una chiamata MQOPEN, le informazioni generiche vengono prese da MQOD (Object Descriptor). Se si specifica MQOD_VERSION_2 nel campo *Version* e un valore maggiore di zero nel campo *RecsPresent* , *Hobj* può essere definito come un handle di un elenco (di una o più code) piuttosto che di una coda. In questo caso, le informazioni specifiche vengono fornite tramite i record oggetto (MQORs), che forniscono i dettagli della destinazione (ovvero, *ObjectName* e *ObjectQMgrName*).

L'handle dell'oggetto (*Hobj*) viene passato alla chiamata MQPUT, consentendo di inserire in un elenco piuttosto che in una coda singola.

Quando un messaggio viene inserito nelle code (MQPUT), le informazioni generiche vengono prese dalla struttura MQPMO (Put Message Option) e MQMD (Message Descriptor). Le informazioni specifiche vengono fornite sotto forma di MQPMR (Put Message Records).

I record di risposta (MQRR) possono ricevere un codice di completamento e un codice motivo specifici per ogni coda di destinazione.

La Figura 56 a pagina 764 mostra il funzionamento degli elenchi di distribuzione.

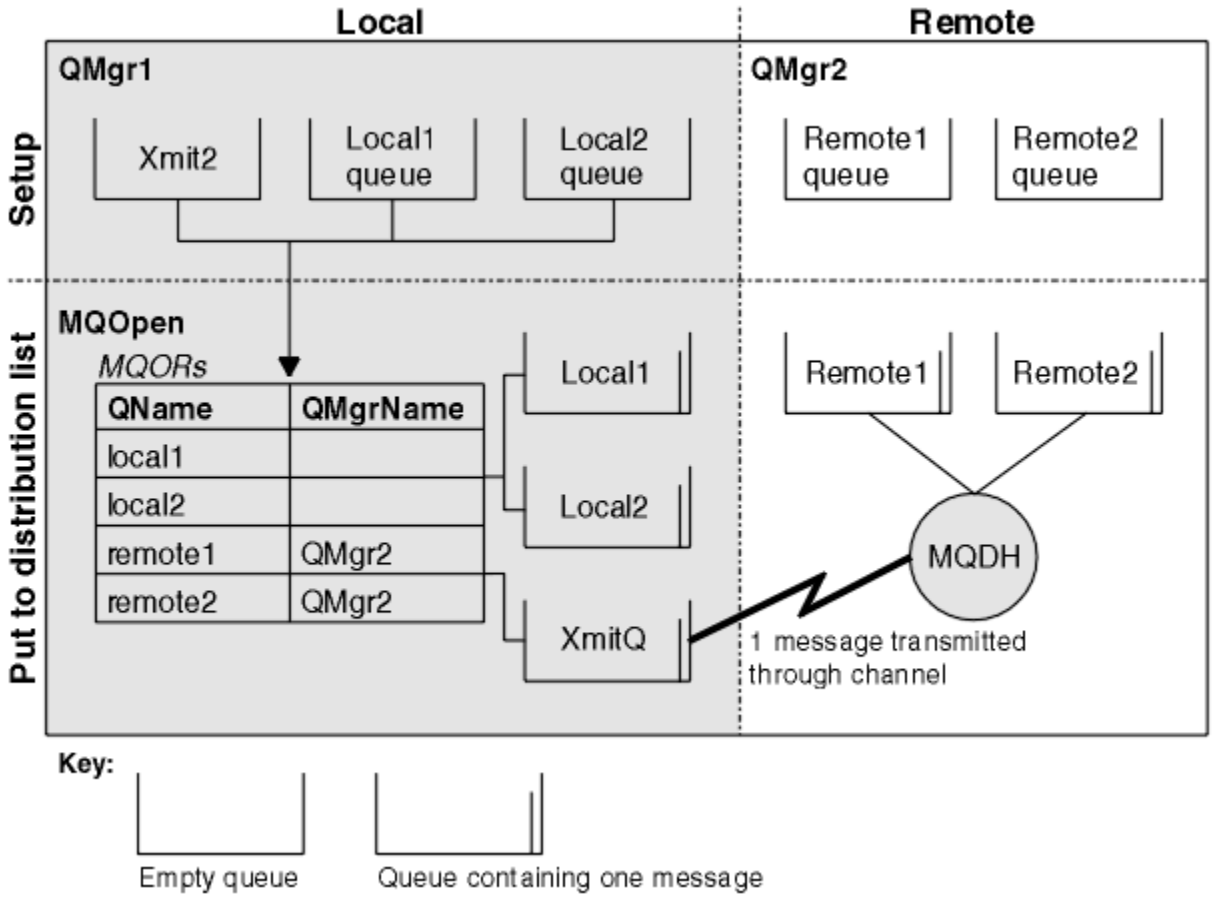


Figura 56. Funzionamento degli elenchi di distribuzione

Apertura degli elenchi di distribuzione
Utilizzare la chiamata MQOPEN per aprire un elenco di distribuzione e utilizzare le opzioni della chiamata per specificare cosa si desidera fare con l'elenco.

Come input per MQOPEN, è necessario fornire:

- Un handle di connessione (consultare “Inserimento di messaggi in una coda” a pagina 753 per una descrizione)
- Informazioni generiche in MQOD (Object Descriptor structure)
- Il nome di ciascuna coda che si desidera aprire, utilizzando MQOR (Object Record structure)

L'output di MQOPEN è:

- Una gestione oggetto che rappresenta l'accesso dell'utente all'elenco di distribuzione
- Un codice di completamento generico
- Un codice di errore generico
- Record di risposta (facoltativo), che contengono un codice di completamento e il motivo per ciascuna destinazione

Utilizzo della struttura MQOD

Utilizzare la struttura MQOD per identificare le code che si desidera aprire.

Per definire un elenco di distribuzione, è necessario specificare MQOD_VERSION_2 nel campo *Version* , un valore maggiore di zero nel campo *RecsPresent* e MQOT_Q nel campo *ObjectType* . Consultare MQOD per una descrizione di tutti i campi della struttura MQOD.

Utilizzo della struttura MQOR

Fornire una struttura MQOR per ogni destinazione.

La struttura contiene i nomi della coda di destinazione e del gestore code. I campi *ObjectName* e *ObjectQMgrName* in MQOD non vengono utilizzati per gli elenchi di distribuzione. Ci devono essere uno o più record oggetto. Se il *ObjectQMgrName* viene lasciato vuoto, viene utilizzato il gestore code locale. Per ulteriori informazioni su questi campi, consultare ObjectName e ObjectQMgrName .

È possibile specificare le code di destinazione in due modi:

- Utilizzando il campo offset *ObjectRecOffset*.

  In questo caso, l'applicazione deve dichiarare la sua struttura contenente una struttura MQOD, seguita dall'array di record MQOR (con tutti gli elementi dell'array necessari) e impostare *ObjectRecOffset* sull'offset del primo elemento dell'array dall'inizio dell'MQOD. Verificare che questo offset sia corretto.

  Si consiglia di utilizzare le funzioni integrate fornite dal linguaggio di programmazione, se sono disponibili in tutti gli ambienti in cui viene eseguita l'applicazione. Il seguente codice illustra questa tecnica per il linguaggio di programmazione COBOL:

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  In alternativa, utilizzare la costante MQOD_CURRENT_LENGTH se il linguaggio di programmazione non supporta le funzioni integrate necessarie in tutti gli ambienti interessati. Il seguente codice illustra questa tecnica:

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  Tuttavia, ciò funziona correttamente solo se la struttura MQOD e l'array dei record MQOR sono contigui; se il compilatore inserisce byte ignorati tra l'MQOD e l'array MQOR, questi devono essere aggiunti al valore memorizzato in *ObjectRecOffset*.

  L'utilizzo di *ObjectRecOffset* è consigliato per i linguaggi di programmazione che non supportano il tipo di dati puntatore o che implementano il tipo di dati puntatore in un modo non portabile in ambienti differenti (ad esempio, il linguaggio di programmazione COBOL).

- Utilizzando il campo puntatore *ObjectRecPtr*.

  In questo caso, l'applicazione può dichiarare l'array delle strutture MQOR separatamente dalla struttura MQOD e impostare *ObjectRecPtr* sull'indirizzo dell'array. Il seguente codice illustra questa tecnica per il linguaggio di programmazione C:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  L'uso di *ObjectRecPtr* è consigliato per i linguaggi di programmazione che supportano il tipo di dati del puntatore in un modo portabile in ambienti diversi (ad esempio, il linguaggio di programmazione C).

Qualunque tecnica si scelga, è necessario utilizzare uno tra *ObjectRecOffset* e *ObjectRecPtr* ; la chiamata ha esito negativo con codice motivo MQRC_OBJECT_RECORDS_ERROR se entrambi sono zero o entrambi sono diversi da zero.

Utilizzo della struttura MQRR

Queste strutture sono specifiche della destinazione; ogni record di risposta contiene un campo *CompCode* e *Reason* per ogni coda di un elenco di distribuzione. È necessario utilizzare questa struttura per consentire di distinguere i problemi.

Ad esempio, se si riceve un codice di errore MQRC_MULTIPLE_REASON e l'elenco di distribuzione contiene cinque code di destinazione, non si sa a quali code si applicano i problemi se non si utilizza questa struttura. Tuttavia, se si dispone di un codice di completamento e di un codice motivo per ogni destinazione, è possibile individuare gli errori più facilmente.

Consultare MQRR per ulteriori informazioni sulla struttura MQRR.

Figura 57 a pagina 766 mostra come aprire un elenco di distribuzione in C.

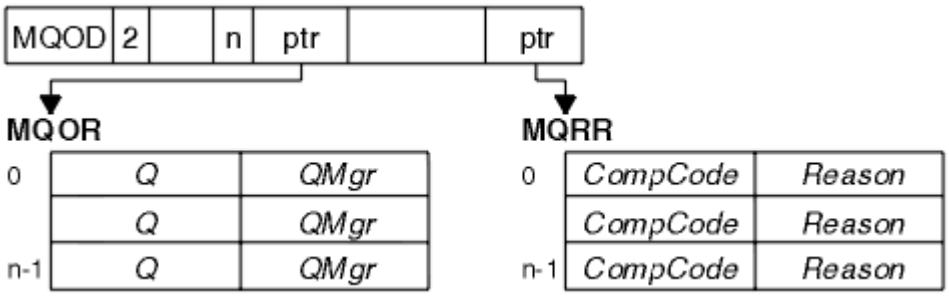


Figura 57. Apertura di un elenco di distribuzione in C

Figura 58 a pagina 766 mostra come è possibile aprire un elenco di distribuzione in COBOL.

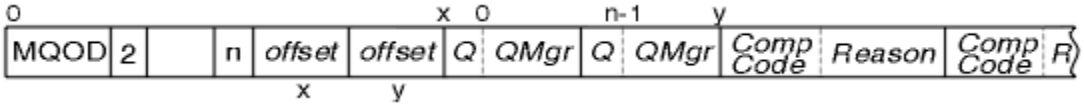


Figura 58. Apertura di un elenco di distribuzione in COBOL

Utilizzo delle opzioni MQOPEN

È possibile specificare le opzioni seguenti quando si apre un elenco di distribuzioni:

- OUTPUT MQOO
- MQOO_FAIL_IF_QUIESCING (facoltativo)
- MQOO_ALTERNATE_USER_AUTHORITY (facoltativo)
- MQOO_ * _CONTEXT (facoltativo)

Consultare “Apertura e chiusura di oggetti” a pagina 742 per una descrizione di queste opzioni.

Inserimento di messaggi in un elenco di distribuzione
Per inserire i messaggi in un elenco di distribuzione, è possibile utilizzare MQPUT o MQPUT1.

Come input, è necessario fornire:

- Un handle di connessione (consultare “Inserimento di messaggi in una coda” a pagina 753 per una descrizione).
- Un handle di oggetto. Se un elenco di distribuzione viene aperto utilizzando MQOPEN, *Hobj* consente solo di inserirlo nell'elenco.
- Una struttura descrittore del messaggio (MQMD). Consultare MQMD per una descrizione di questa struttura.
- Informazioni di controllo nel formato di una struttura di opzioni di inserimento messaggi (MQPMO). Consultare “Specifica delle opzioni utilizzando la struttura MQPMO” a pagina 755 per informazioni sul completamento dei campi della struttura MQPMO.
- Informazioni di controllo nel formato MQPMR (Put Message Records).

- La lunghezza dei dati contenuti nel messaggio (MQLONG).
- I dati del messaggio.

L'output è:

- Un codice di completamento
- Un codice di errore
- Record di risposta (facoltativo)

Utilizzo della struttura MQPMR

Questa struttura è facoltativa e fornisce informazioni specifiche di destinazione per alcuni campi che è possibile identificare in modo diverso da quelli già identificati in MQMD.

Per una descrizione di questi campi, consultare MQPMR.

Il contenuto di ciascun record dipende dalle informazioni fornite nel campo *PutMsgRecFields* di MQPM. Ad esempio, nel programma di esempio AMQSPTL0.C (consultare "Programma di esempio Elenco di distribuzione" a pagina 1093 per una descrizione) che mostra l'utilizzo degli elenchi di distribuzione, l'esempio sceglie di fornire i valori per *MsgId* e *CorrelId* in MQPMR. Questa sezione del programma di esempio è simile alla seguente:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

Ciò implica che *MsgId* e *CorrelId* vengono forniti per ciascuna destinazione di un elenco di distribuzione. I record di inserimento messaggio vengono forniti come un array.

Figura 59 a pagina 767 mostra come è possibile inserire un messaggio in un elenco di distribuzione in C.

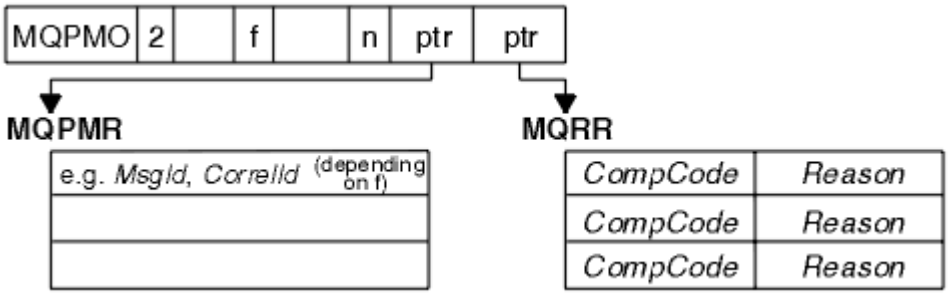


Figura 59. Inserimento di un messaggio in un elenco di distribuzione in C

Figura 60 a pagina 767 mostra come è possibile inserire un messaggio in un elenco di distribuzione in COBOL.

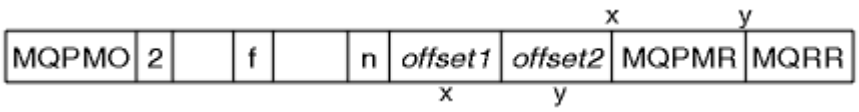


Figura 60. Inserimento di un messaggio in un elenco di distribuzione in COBOL

Utilizzo di MQPUT1

Se si sta utilizzando MQPUT1, considerare i seguenti punti:

1. I valori dei campi *ResponseRecOffset* e *ResponseRecPtr* devono essere null o zero.
2. I record di risposta, se richiesti, devono essere indirizzati da MQOD.

Alcuni casi in cui le chiamate put hanno esito negativo

Se alcuni attributi di una coda vengono modificati utilizzando l'opzione FORCE su un comando durante l'intervallo tra l'emissione di una chiamata MQOPEN e MQPUT, la chiamata MQPUT ha esito negativo e restituisce il codice motivo MQRC_OBJECT_CHANGED.

Il gestore code contrassegna l'handle dell'oggetto come non più valido. Ciò si verifica anche se le modifiche vengono effettuate durante l'elaborazione di una chiamata MQPUT1 o se le modifiche si applicano a qualsiasi coda in cui si risolve il nome della coda. Gli attributi che influenzano l'handle in questo modo sono riportati nella descrizione della chiamata MQOPEN in MQOPEN. Se la chiamata restituisce il codice motivo MQRC_OBJECT_CHANGED, chiudere la coda, riaprirla, quindi provare a inserire nuovamente un messaggio.

Se le operazioni di inserimento sono inibite per una coda in cui si sta tentando di inserire i messaggi (o per qualsiasi coda in cui il nome della coda si risolve), la chiamata MQPUT o MQPUT1 ha esito negativo e restituisce il codice di errore MQRC_PUT_INIITED. Potrebbe essere possibile inserire un messaggio correttamente se si tenta la chiamata in un secondo momento, se la progettazione dell'applicazione è tale che altri programmi modificano regolarmente gli attributi delle code.

Inoltre, se la coda in cui si sta tentando di inserire il messaggio è piena, la chiamata MQPUT o MQPUT1 ha esito negativo e restituisce MQRC_Q_FULL.

Se una coda dinamica (temporanea o permanente) è stata eliminata, le chiamate MQPUT che utilizzano un handle di oggetto precedentemente acquisito hanno esito negativo e restituiscono il codice motivo MQRC_Q_DELETED. In questa situazione, si consiglia di chiudere l'handle dell'oggetto poiché non è più utile.

Nel caso di elenchi di distribuzione, più codici di completamento e codici motivo possono verificarsi in una singola richiesta. Non possono essere gestiti utilizzando solo i campi di output *CompCode* e *Reason* su MQOPEN e MQPUT.

Quando si utilizzano gli elenchi di distribuzione per inserire i messaggi in più destinazioni, i record di risposta contengono gli specifici *CompCode* e *Reason* per ciascuna destinazione. Se si riceve un codice di completamento di MQCC_FAILED, nessun messaggio viene inserito correttamente in una coda di destinazione. Se il codice di completamento è MQCC_WARNING, il messaggio viene inserito correttamente su una o più code di destinazione. Se si riceve un codice di ritorno MQRC_MULTIPLE_REASON, i codici di errore non sono tutti uguali per ogni destinazione. Si consiglia pertanto di utilizzare la struttura MQRR in modo da poter determinare la coda o le code che hanno causato un errore e le relative cause.

Richiamo dei messaggi da una coda

Utilizzare queste informazioni per ottenere messaggi da una coda.

È possibile richiamare i messaggi da una coda in due modi:

1. È possibile eliminare un messaggio dalla coda in modo che altri programmi non possano più visualizzarlo.
2. È possibile copiare un messaggio, lasciando il messaggio originale sulla coda. Questa operazione è nota come *esplorazione*. È possibile rimuovere il messaggio una volta esplorato.

In entrambi i casi, si utilizza la chiamata MQGET, ma prima l'applicazione deve essere connessa al gestore code e si deve utilizzare la chiamata MQOPEN per aprire la coda (per l'input, la ricerca o entrambi). Queste operazioni sono descritte in “Connessione e disconnessione da un gestore code” a pagina 735 e “Apertura e chiusura di oggetti” a pagina 742.

Una volta aperta la coda, è possibile utilizzare la chiamata MQGET ripetutamente per sfogliare o rimuovere i messaggi sulla stessa coda. Richiamare MQCLOSE una volta terminato di richiamare tutti i messaggi desiderati dalla coda.

Utilizzare i seguenti collegamenti per ulteriori informazioni sul richiamo dei messaggi da una coda:

- “Come ottenere i messaggi da una coda utilizzando la chiamata MQGET” a pagina 769
- “L'ordine in cui i messaggi vengono richiamati da una coda” a pagina 774

- “Ricezione di un messaggio particolare” a pagina 786
- “Miglioramento delle prestazioni dei messaggi non persistenti” a pagina 787
- z/OS “Tipo di indice” a pagina 791
- “Gestione dei messaggi con lunghezza superiore a 4 MB” a pagina 792
- “In attesa di messaggi” a pagina 798
- z/OS “Segnaletica” a pagina 798
- “Backout ignorato” a pagina 800
- “Conversione dati applicazione” a pagina 802
- “Visualizzazione dei messaggi su una coda” a pagina 803
- “Alcuni casi in cui la chiamata MQGET non riesce” a pagina 809

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Come ottenere i messaggi da una coda utilizzando la chiamata MQGET

La chiamata MQGET richiama un messaggio da una coda locale aperta. Non può richiamare un messaggio da una coda su un altro sistema.

Come input per la chiamata MQGET, è necessario fornire:

- Un handle di connessione.
- Un handle di coda.
- Una descrizione del messaggio che si desidera ottenere dalla coda. Si tratta di una struttura MQMD (Message Descriptor).
- Controllare le informazioni sotto forma di una struttura MQGMO (Get Message Options).
- La dimensione del buffer che è stato assegnato per conservare il messaggio (MQLONG).
- L'indirizzo della memoria in cui inserire il messaggio.

L'output di MQGET è:

- Un codice di errore
- Un codice di completamento
- Il messaggio nell'area di buffer specificato, se la chiamata viene completata correttamente
- La struttura delle opzioni, modificata per mostrare il nome della coda da cui è stato richiamato il messaggio
- La struttura del descrittore del messaggio, con il contenuto dei campi modificati per descrivere il messaggio richiamato
- La lunghezza del messaggio (MQLONG)

C'è una descrizione della chiamata MQGET in MQGET.

Le seguenti sezioni descrivono le informazioni che è necessario fornire come input alla chiamata MQGET.

- “Specifica degli handle di connessione” a pagina 770
- “Descrizione dei messaggi utilizzando la struttura di MQMD e la chiamata MQGET” a pagina 770
- “Specifica delle opzioni MQGET utilizzando la struttura MQGMO” a pagina 770
- “Specifica della dimensione dell'area buffer” a pagina 773

Specifica degli handle di connessione

z/OS Per applicazioni CICS su z/OS , è possibile specificare la costante MQHC_DEF_HCONN (che ha il valore zero) oppure utilizzare l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX. Per altre applicazioni, utilizzare sempre l'handle di connessione restituito dalla chiamata MQCONN o MQCONNX.

Utilizzare il gestore code (*Hobj*) restituito quando si richiama MQOPEN.

Descrizione dei messaggi utilizzando la struttura di MQMD e la chiamata MQGET

Per identificare il messaggio che si desidera ottenere da una coda, utilizzare la struttura del descrittore del messaggio (MQMD).

Si tratta di un parametro di input / output per la chiamata MQGET. È disponibile un'introduzione alle proprietà del messaggio descritte da MQMD in “IBM MQ messaggi” a pagina 18e una descrizione della struttura stessa in MQMD.

Se si conosce il messaggio che si desidera richiamare dalla coda, consultare “Ricezione di un messaggio particolare” a pagina 786.

Se non si specifica un particolare messaggio, MQGET richiama il *primo* messaggio nella coda. “L'ordine in cui i messaggi vengono richiamati da una coda” a pagina 774 descrive in che modo la priorità di un messaggio, l'attributo **MsgDeliverySequence** della coda e l'opzione MQGMO_LOGICAL_ORDER determinano l'ordine dei messaggi nella coda.

Nota: Se si desidera utilizzare MQGET più di una volta (ad esempio, per esaminare i messaggi nella coda), è necessario impostare i campi *MsgId* e *CorrelId* di questa struttura su null dopo ogni chiamata. Questo elimina questi campi degli identificatori del messaggio che è stato richiamato.

Tuttavia, se si desidera raggruppare i messaggi, il *GroupId* deve essere lo stesso per i messaggi nello stesso gruppo, in modo che la chiamata ricerca un messaggio con gli stessi identificativi del messaggio precedente per costituire l'intero gruppo.

Specifica delle opzioni MQGET utilizzando la struttura MQGMO

La struttura MQGMO è una variabile di input / output per passare le opzioni alla chiamata MQGET. Le seguenti sezioni consentono di completare alcuni dei campi di questa struttura.

È presente una descrizione della struttura MQGMO in MQGMO.

StrucId

StrucId è un campo di 4 caratteri utilizzato per identificare la struttura come una struttura di opzioni get - message. Specificare sempre MQGMO_STRUC_ID.

Version

*Version*descrive il numero di versione della struttura. MQGMO_VERSION_1 è il valore predefinito. Se si desidera utilizzare i campi della Versione 2 o richiamare i messaggi in ordine logico, specificare MQGMO_VERSION_2. Se si desidera utilizzare i campi della Versione 3 o richiamare i messaggi in ordine logico, specificare MQGMO_VERSION_3. MQGMO_CURRENT_VERSION imposta l'applicazione per utilizzare il livello più recente.

Options

All'interno del codice, è possibile selezionare le opzioni in qualsiasi ordine; ogni opzione è rappresentata da un bit nel campo *Options* .

Il campo *Options* controlla:

- Se la chiamata MQGET attende l'arrivo di un messaggio sulla coda prima del completamento (consultare "In attesa di messaggi" a pagina 798)
- Se l'operazione di acquisizione è inclusa in un'unità di lavoro.
- Se un messaggio non persistente viene richiamato al di fuori del punto di sincronizzazione, consentendo la messaggistica rapida
- z/OS Su IBM MQ for z/OS, se il messaggio richiamato è contrassegnato come backout ignorato (consultare "Backout ignorato" a pagina 800)
- Se il messaggio viene rimosso dalla coda o semplicemente esplorato
- Indica se selezionare un messaggio utilizzando un cursore di ricerca o altri criteri di selezione
- Se la chiamata ha esito positivo anche se il messaggio è più lungo del buffer
- z/OS Su IBM MQ for z/OS, indica se consentire il completamento della chiamata. Questa opzione imposta anche un segnale per indicare che si desidera ricevere una notifica quando arriva un messaggio
- Indica se la chiamata ha esito negativo se il gestore code è in stato di sospensione
- z/OS Su IBM MQ for z/OS, se la chiamata non riesce se la connessione è in stato di inattività
- Se è richiesta la conversione dei dati del messaggio dell'applicazione (consultare "Conversione dati applicazione" a pagina 802)
- L'ordine in cui i messaggi e i segmenti vengono richiamati da una coda z/OS (tranne IBM MQ for z/OS)
- Se i messaggi logici completi sono richiamabili solo z/OS (ad eccezione di IBM MQ for z/OS)
- Se i messaggi in un gruppo possono essere richiamati solo quando *tutti* i messaggi nel gruppo sono disponibili
- Se i segmenti in un messaggio logico possono essere richiamati solo quando *tutti* i segmenti nel messaggio logico sono disponibili z/OS (ad eccezione di IBM MQ for z/OS)

Se si lascia il campo *Options* impostato sul valore predefinito (MQGMO_NO_WAIT), la chiamata MQGET funziona in questo modo:

- Se non c'è alcun messaggio che corrisponde ai criteri di selezione sulla coda, la chiamata non attende l'arrivo di un messaggio, ma viene completata immediatamente. z/OS Inoltre, in IBM MQ for z/OS, la chiamata non imposta un segnale che richiede la notifica quando arriva un messaggio di questo tipo.
- Il modo in cui la chiamata opera con i punti di sincronizzazione è determinato dalla piattaforma:

Piattaforma	Sotto il controllo del punto di sincronizzazione
IBM i	No
Sistemi AIX and Linux	No
z/OS z/OS z/OS	Sì
Sistemi Windows	No

- z/OS Su IBM MQ for z/OS, il messaggio richiamato non è contrassegnato come backout ignorato.
- Il messaggio selezionato viene rimosso dalla coda (non esplorato).
- Non è richiesta alcuna conversione dei dati del messaggio dell'applicazione.
- La chiamata ha esito negativo se il messaggio è più lungo del buffer.

WaitInterval
Il campo *WaitInterval* specifica il tempo massimo (in millesimi di secondo) che la chiamata MQGET attende per l'arrivo di un messaggio sulla coda quando si utilizza l'opzione MQGMO_WAIT. Se nessun messaggio arriva entro il periodo di tempo specificato in *WaitInterval*, la chiamata viene completata e restituisce un codice motivo che indica che non c'era alcun messaggio che corrispondesse ai criteri di selezione sulla coda.

z/OS Su IBM MQ for z/OS, se si utilizza l'opzione MQGMO_SET_SIGNAL, il campo *WaitInterval* specifica l'ora per cui è impostato il segnale.

Per ulteriori informazioni su queste opzioni, consultare "In attesa di messaggi" a pagina 798 z/OS e "Segnaletica" a pagina 798 .

z/OS ***Signal1***
Signal1 è supportato solo su IBM MQ for z/OS.

Se si utilizza l'opzione MQGMO_SET_SIGNAL per richiedere che l'applicazione venga avvisata quando arriva un messaggio adatto, specificare il tipo di segnale nel campo *Signal1* . In IBM MQ su altre piattaforme, il campo *Signal1* è riservato e il suo valore non è significativo.

z/OS Per ulteriori informazioni, consultare "Segnaletica" a pagina 798.

Signal2
Il campo *Signal2* è riservato su tutte le piattaforme e il suo valore non è significativo.

z/OS Per ulteriori informazioni, consultare "Segnaletica" a pagina 798.

ResolvedQName
ResolvedQName è un campo di output in cui il gestore code restituisce il nome della coda (dopo la risoluzione di qualsiasi alias) da cui è stato richiamato il messaggio.

MatchOptions
MatchOptions controlla i criteri di selezione per MQGET.

GroupStatus
GroupStatus indica se il messaggio richiamato si trova in un gruppo.

SegmentStatus
SegmentStatus indica se l'elemento richiamato è un segmento di un messaggio logico.

Segmentation
Segmentation indica se la segmentazione è consentita per il messaggio richiamato.

MsgToken

MsgToken identifica in modo univoco un messaggio.

ReturnedLength
ReturnedLength è un campo di output in cui il gestore code restituisce la lunghezza dei dati del messaggio restituiti (in byte).

MsgHandle
L'handle per un messaggio che deve essere popolato con le proprietà del messaggio richiamato dalla coda. L'handle è stato precedentemente creato da una chiamata MQCRTMH. Tutte le proprietà già associate all'handle vengono eliminate prima di richiamare un messaggio.

Specifica della dimensione dell'area buffer

Nel parametro **BufferLength** della chiamata MQGET, specificare le dimensioni dell'area di buffer per contenere i dati del messaggio che si richiamano. Si decide quanto grande dovrebbe essere in tre modi:

1. È possibile che si conosca già la lunghezza dei messaggi da prevedere da questo programma. In tal caso, specificare un buffer di questa dimensione.

   Tuttavia, è possibile utilizzare l'opzione MQGMO_ACCEPT_TRUNCATED_MSG nella struttura MQGMO se si desidera che la chiamata MQGET venga completata anche se il messaggio è troppo grande per il buffer. In questo caso:

   - Il buffer è riempito con la maggior parte del messaggio che può contenere
   - La chiamata restituisce un codice di completamento di avvertenza
   - Il messaggio viene rimosso dalla coda (eliminando il resto del messaggio) oppure il cursore di ricerca è avanzato (se si sta sfogliando la coda)
   - La lunghezza reale del messaggio viene restituita in *DataLength*

   Senza questa opzione, la chiamata viene ancora completata con un'avvertenza, ma non rimuove il messaggio dalla coda (o non fa avanzare il cursore di ricerca).
2. Stimare una dimensione per il buffer (o anche specificare una dimensione di zero byte) e *non* utilizzare l'opzione MQGMO_ACCEPT_TRUNCATED_MSG. Se la chiamata MQGET ha esito negativo (ad esempio, perché il buffer è troppo piccolo), la lunghezza del messaggio viene restituita nel parametro **DataLength** della chiamata. (Il buffer è ancora riempito con la quantità di messaggi che può contenere, ma l'elaborazione della chiamata non viene completata.) Memorizzare il *MsgId* di questo messaggio, quindi ripetere la chiamata MQGET, specificando un'area buffer della dimensione corretta e il *MsgId* annotato dalla prima chiamata.

   Se il programma sta servendo una coda che è servita anche da altri programmi, uno di questi programmi potrebbe rimuovere il messaggio che si desidera prima che il programma possa emettere un'altra chiamata MQGET. Il programma potrebbe perdere tempo nella ricerca di un messaggio che non esiste più. Per evitare ciò, sfogliare prima la coda fino a trovare il messaggio desiderato, specificando un *BufferLength* di zero e utilizzando l'opzione MQGMO_ACCEPT_TRUNCATED_MSG. Questo posiziona il cursore di ricerca sotto il messaggio desiderato. È quindi possibile richiamare il messaggio richiamando nuovamente MQGET, specificando l'opzione MQGMO_MSG_UNDER_CURSOR. Se un altro programma rimuove il messaggio tra le chiamate di ricerca e rimozione, il secondo MQGET non riesce immediatamente (senza ricercare l'intera coda), perché non c'è alcun messaggio sotto il cursore di ricerca.
3. L'attributo *MaxMsgLength* Coda determina la lunghezza massima dei messaggi accettati per tale coda; l'attributo *MaxMsgLength* Gestore code determina la lunghezza massima dei messaggi accettati per tale gestore code. Se non si conosce la lunghezza del messaggio prevista, è possibile richiedere informazioni sull'attributo **MaxMsgLength** (utilizzando la chiamata MQINQ), quindi specificare un buffer di questa dimensione.

   Provare a rendere la dimensione del buffer il più possibile vicina alla dimensione reale del messaggio per evitare prestazioni ridotte.

   Per ulteriori informazioni sull'attributo **MaxMsgLength** , consultare “Aumento della lunghezza massima del messaggio” a pagina 792.

L'ordine in cui i messaggi vengono richiamati da una coda

È possibile controllare l'ordine in cui richiamare i messaggi da una coda. Questa sezione esamina le opzioni.

Priorità

Un programma può assegnare una priorità a un messaggio quando inserisce il messaggio su una coda (consultare “Priorità dei messaggi” a pagina 27). I messaggi di uguale priorità vengono memorizzati in una coda in ordine di arrivo, non in ordine di commit.

Il gestore code gestisce le code nella sequenza FIFO (first in, first out) o in FIFO all'interno della sequenza di priorità. Ciò dipende dall'impostazione dell'attributo **MsgDeliverySequence** della coda. Quando un messaggio arriva su una coda, viene inserito immediatamente dopo l'ultimo messaggio con la stessa priorità.

I programmi possono richiamare il primo messaggio da una coda oppure possono richiamare un determinato messaggio da una coda, ignorando la priorità di tali messaggi. Ad esempio, un programma potrebbe voler elaborare la risposta a un particolare messaggio inviato in precedenza. Per ulteriori informazioni, consultare “Ricezione di un messaggio particolare” a pagina 786.

Se un'applicazione inserisce una sequenza di messaggi in una coda, un'altra applicazione può richiamare tali messaggi nello stesso ordine in cui sono stati inseriti, a condizione che:

- I messaggi hanno tutti la stessa priorità
- I messaggi sono stati tutti inseriti all'interno della stessa unità di lavoro o all'esterno di un'unità di lavoro
- La coda è locale per l'applicazione di inserimento

Se queste condizioni non vengono soddisfatte e le applicazioni dipendono dai messaggi richiamati in un certo ordine, le applicazioni devono includere le informazioni sulla sequenza nei dati del messaggio o stabilire un mezzo per confermare la ricezione di un messaggio prima che venga inviato il messaggio successivo.

z/OS Su IBM MQ for z/OS, è possibile utilizzare l'attributo della coda, *IndexType*, per incrementare la velocità delle operazioni MQGET sulla coda. Per ulteriori informazioni, consultare “Tipo di indice” a pagina 791.

Ordinamento logico e fisico

I messaggi sulle code possono verificarsi (all'interno di ogni livello di priorità) in ordine *fisico* o *logico* .

L'ordine fisico è l'ordine in cui i messaggi arrivano su una coda. L'ordine logico è quando tutti i messaggi e i segmenti all'interno di un gruppo si trovano nella loro sequenza logica, uno accanto all'altro, nella posizione determinata dalla posizione fisica del primo elemento appartenente al gruppo.

Per una descrizione di gruppi, messaggi e segmenti, consultare “Gruppi di messaggi” a pagina 45. Questi ordini fisici e logici possono essere diversi perché:

- I gruppi possono arrivare a una destinazione in momenti simili da applicazioni diverse, perdendo quindi qualsiasi ordine fisico distinto.
- Anche all'interno di un singolo gruppo, i messaggi possono essere disordinati a causa del reinstradamento o del ritardo di alcuni dei messaggi nel gruppo.

Ad esempio, l'ordine logico potrebbe essere simile alla figura Figura 61 a pagina 775:

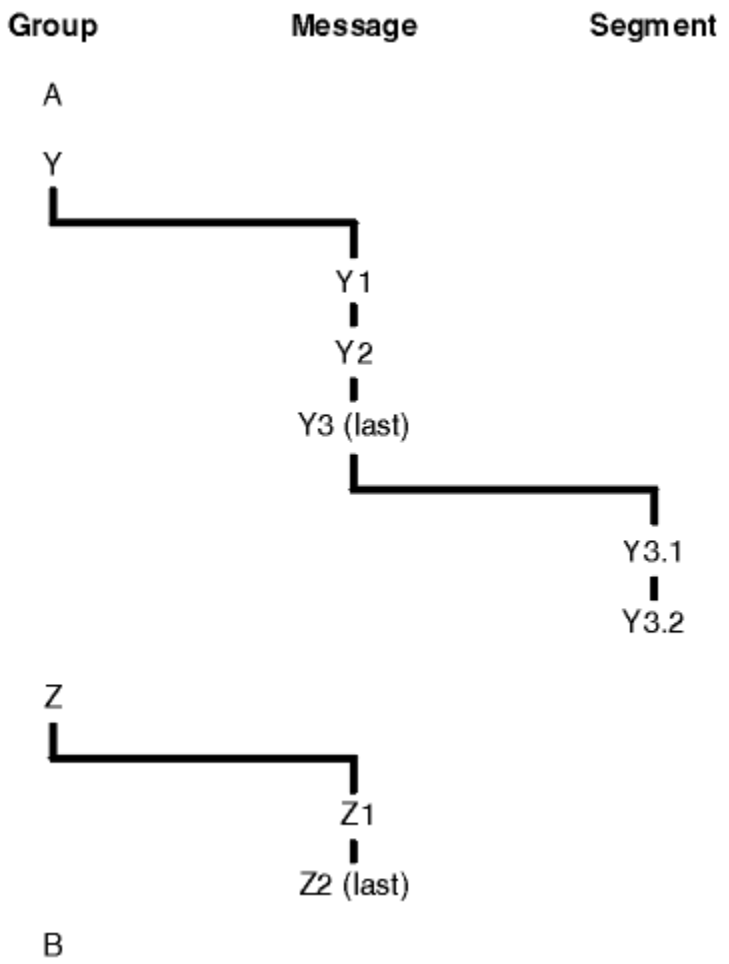


Figura 61. Ordine logico su una coda

Questi messaggi si verificano nel seguente ordine logico su una coda:

1. Messaggio A (non in un gruppo)
2. Messaggio logico 1 del gruppo Y
3. Messaggio logico 2 del gruppo Y
4. Segmento 1 di (ultimo) messaggio logico 3 del gruppo Y
5. (Ultimo) segmento 2 del (ultimo) messaggio logico 3 del gruppo Y
6. Messaggio logico 1 del gruppo Z
7. (Ultimo) messaggio logico 2 del gruppo Z
8. Messaggio B (non in un gruppo)

L'ordine fisico, tuttavia, potrebbe essere completamente diverso. La posizione fisica del *primo* elemento all'interno di ciascun gruppo determina la posizione logica dell'intero gruppo. Ad esempio, se i gruppi Y e Z sono arrivati a orari simili e il messaggio 2 del gruppo Z ha superato il messaggio 1 dello stesso gruppo, l'ordine fisico sarà simile alla figura Figura 62 a pagina 776:

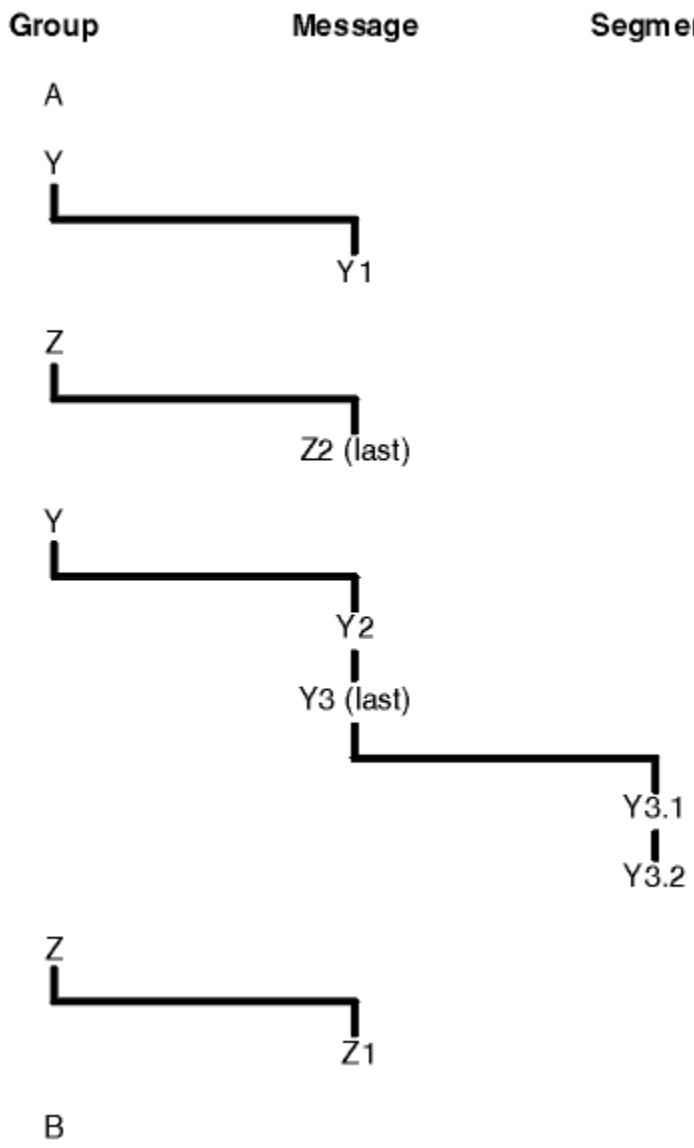


Figura 62. Ordine fisico su una coda

Questi messaggi si verificano nel seguente ordine fisico sulla coda:

1. Messaggio A (non in un gruppo)
2. Messaggio logico 1 del gruppo Y
3. Messaggio logico 2 del gruppo Z
4. Messaggio logico 2 del gruppo Y
5. Segmento 1 di (ultimo) messaggio logico 3 del gruppo Y
6. (Ultimo) segmento 2 del (ultimo) messaggio logico 3 del gruppo Y
7. Messaggio logico 1 del gruppo Z
8. Messaggio B (non in un gruppo)

Nota: Su IBM MQ for z/OS, l'ordine fisico dei messaggi sulla coda non è garantito se la coda è indicizzata da GROUPID.

Quando si ricevono i messaggi, è possibile specificare MQGMO_LOGICAL_ORDER per richiamare i messaggi in ordine logico piuttosto che in ordine fisico.

Se si immette una chiamata MQGET con MQGMO_BROWSE_FIRST e MQGMO_LOGICAL_ORDER, le successive chiamate MQGET con MQGMO_BROWSE_NEXT devono specificare anche

MQGMO_LOGICAL_ORDER. Al contrario, se MQGET con MQGMO_BROWSE_FIRST non specifica MQGMO_LOGICAL_ORDER, né le seguenti MQGET con MQGMO_BROWSE_NEXT.

Le informazioni sul gruppo e sul segmento che il gestore code conserva per le chiamate MQGET che sfogliano i messaggi sulla coda sono separate dal gruppo e le informazioni sul segmento che il gestore code conserva per le chiamate MQGET che rimuovono i messaggi dalla coda. Quando si specifica MQGMO_BROWSE_FIRST, il gestore code ignora le informazioni sul gruppo e sul segmento per la ricerca e esegue la scansione della coda come se non vi fossero gruppi correnti e messaggi logici correnti.

Nota: non utilizzare una chiamata MQGET per esplorare *oltre la fine* di un gruppo di messaggi (o di un messaggio logico non presente in un gruppo) senza specificare MQGMO_LOGICAL_ORDER. Ad esempio, se l'ultimo messaggio nel gruppo *precede* il primo messaggio nel gruppo sulla coda, utilizzando MQGMO_BROWSE_NEXT per esplorare oltre la fine del gruppo, specificando MQMO_MATCH_MSG_SEQ_NUMBER con *MsgSeqNumber* impostato su 1 (per trovare il primo messaggio del gruppo successivo) restituisce nuovamente il primo messaggio nel gruppo già esplorato. Ciò potrebbe verificarsi immediatamente o in un numero di chiamate MQGET successive (se sono presenti gruppi che intervengono).

Evitare la possibilità di un loop infinito aprendo la coda *due volte* per la ricerca:

- Utilizzare il primo handle per ricercare solo il primo messaggio in ciascun gruppo.
- Utilizzare il secondo handle per ricercare solo i messaggi all'interno di un determinato gruppo.
- Utilizzare le opzioni MQMO_ * per spostare il secondo cursore di ricerca nella posizione del primo cursore di ricerca, prima di esaminare i messaggi nel gruppo.
- Non utilizzare la ricerca MQGMO_BROWSE_NEXT oltre la fine di un gruppo.

Per ulteriori informazioni su questo argomento, consultare MQGET, MQMDe Regole per la convalida delle opzioni MQI.

Per la maggior parte delle applicazioni si sceglierà probabilmente l'ordinamento logico o fisico durante la navigazione. Tuttavia, se si desidera passare da una modalità all'altra, tenere presente che quando si emette per la prima volta una ricerca con MQGMO_LOGICAL_ORDER, viene stabilita la propria posizione all'interno della sequenza logica.

Se il primo elemento all'interno del gruppo non è presente in questo momento, il gruppo in cui ci si trova non viene considerato come parte della sequenza logica.

Una volta che il cursore di ricerca si trova all'interno di un gruppo, è possibile continuare all'interno dello stesso gruppo, anche se il primo messaggio viene rimosso. Inizialmente, tuttavia, non è possibile spostarsi in un gruppo utilizzando MQGMO_LOGICAL_ORDER in cui il primo elemento non è presente.

ORDER MQPMO_LOGICAL_

L'opzione MQPMO indica al gestore code in che modo l'applicazione inserisce i messaggi in gruppi e segmenti di messaggi logici. Può essere specificato solo nella chiamata MQPUT; non è valido nella chiamata MQPUT1 .

Se MQPMO_LOGICAL_ORDER è specificato, indica che l'applicazione utilizza le chiamate MQPUT successive per:

1. Inserire i segmenti in ciascun segmento logico nell'ordine di offset crescente dei segmenti, a partire da 0, senza intervalli.
2. Inserire tutti i segmenti in un unico messaggio logico prima di inserirli nel successivo messaggio logico.
3. Inserire i messaggi logici in ciascun gruppo di messaggi nell'ordine crescente del numero di sequenza dei messaggi, a partire da 1, senza intervalli. IBM MQ aumenta il numero di sequenza dei messaggi automaticamente.
4. Inserire tutti i messaggi logici in un unico gruppo di messaggi prima di inserirli nel successivo gruppo di messaggi.

Poiché l'applicazione ha indicato al gestore code il modo in cui inserisce i messaggi in gruppi e segmenti di messaggi logici, non è necessario che l'applicazione mantenga e aggiorni le informazioni sul gruppo e sul segmento relative a ciascuna chiamata MQPUT, poiché il gestore code conserva e

aggiorna queste informazioni. In particolare, significa che l'applicazione non deve impostare i campi *GroupId*, *MsgSeqNumbe*r e *Offset* in MQMD, poiché il gestore code imposta questi campi sui valori appropriati. L'applicazione deve impostare solo il campo *MsgFlags* in MQMD, per indicare quando i messaggi appartengono a gruppi o sono segmenti di messaggi logici e per indicare l'ultimo messaggio in un gruppo o l'ultimo segmento di un messaggio logico.

Una volta avviato un gruppo di messaggi o un messaggio logico, le successive chiamate MQPUT devono specificare gli indicatori MQMF_ * appropriati in *MsgFlags* in MQMD. Se l'applicazione tenta di inserire un messaggio che non si trova in un gruppo quando è presente un gruppo di messaggi non terminato o un messaggio che non è un segmento quando è presente un messaggio logico non terminato, la chiamata non riesce con il codice di errore MQRC_INCOMPLETE_GROUP o MQRC_INCOMPLETE_MSG, come appropriato. Tuttavia, il gestore code conserva le informazioni sul gruppo di messaggi corrente o sul messaggio logico corrente e l'applicazione può terminarli inviando un messaggio (possibilmente senza dati del messaggio dell'applicazione) specificando MQMF_LAST_MSG_IN_GROUP o MQMF_LAST_SEGMENT come appropriato, prima di emettere nuovamente la chiamata MQPUT per inserire il messaggio che non si trova nel gruppo o non in un segmento.

Figura 62 a pagina 776 mostra le combinazioni di opzioni e indicatori validi e i valori dei campi *GroupId*, *MsgSeqNumbe*r e *Offset* utilizzati dal gestore code in ciascun caso. Le combinazioni di opzioni e indicatori non mostrate nella tabella non sono valide. Le colonne nella tabella hanno i seguenti significati; significa Sì o No:

ORD LOG
Se l'opzione MQPMO_LOGICAL_ORDER è specificata sulla chiamata.

MIG
Se l'opzione MQMF_MSG_IN_GROUP o MQMF_LAST_MSG_IN_GROUP è specificata nella chiamata.

SEG
Se l'opzione MQMF_SEGMENT o MQMF_LAST_SEGMENT è specificata nella chiamata.

SEG - OK
Indica se l'opzione MQMF_SEGMENTATION_ALLOWED è specificata nella chiamata.

Grp corrente
Se esiste un gruppo di messaggi corrente prima della chiamata.

Messaggio di log corrente
Se esiste un messaggio logico corrente prima della chiamata.

Altre colonne
Mostra i valori utilizzati dal gestore code. Precedente indica il valore utilizzato per il campo nel precedente messaggio per la gestione della coda.

Tabella 116. Opzioni MQPUT relative a messaggi in gruppi e segmenti di messaggi logici

Opzioni specificate	Opzioni specificate	Opzioni specificate	Opzioni specificate	Stato messaggio di log e gruppo prima della chiamata	Stato messaggio di log e gruppo prima della chiamata	Valori utilizzati dal gestore code	Valori utilizzati dal gestore code	Valori utilizzati dal gestore code
ORD LOG	MIG	SEG	SEG - OK	Grp corrente	Messaggio di log corrente	**GroupI**d	**MsgSeqNumber**	**Offset**
Sì	No	No	No	No	No	MQGI_NONE	1	0
Sì	No	No	Sì	No	No	Nuovo ID gruppo	1	0
Sì	No	Sì	Entrambi	No	No	Nuovo ID gruppo	1	0
Sì	No	Sì	Entrambi	No	Sì	ID gruppo precedente	1	Offset precedente + lunghezza segmento precedente
Sì	Sì	Entrambi	Entrambi	No	No	Nuovo ID gruppo	1	0
Sì	Sì	Entrambi	Entrambi	Sì	No	ID gruppo precedente	Numero di sequenza precedente + 1	0
Sì	Sì	Sì	Entrambi	Sì	Sì	ID gruppo precedente	Numero sequenza precedente	Offset precedente + lunghezza segmento precedente
No	No	No	No	Entrambi	Entrambi	MQGI_NONE	1	0
No	No	No	Sì	Entrambi	Entrambi	Nuovo ID gruppo se MQGI_NONE, altrimenti valore nel campo	1	0
No	No	Sì	Entrambi	Entrambi	Entrambi	Nuovo ID gruppo se MQGI_NONE, altrimenti valore nel campo	1	Valore nel campo
No	Sì	No	Entrambi	Entrambi	Entrambi	Nuovo ID gruppo se MQGI_NONE, altrimenti valore nel campo	Valore nel campo	0

Tabella 116. Opzioni MQPUT relative a messaggi in gruppi e segmenti di messaggi logici (Continua)

Opzioni specificate	Opzioni specificate	Opzioni specificate	Opzioni specificate	Stato messaggio di log e gruppo prima della chiamata	Stato messaggio di log e gruppo prima della chiamata	Valori utilizzati dal gestore code	Valori utilizzati dal gestore code	Valori utilizzati dal gestore code
No	Sì	Sì	Entrambi	Entrambi	Entrambi	Nuovo ID gruppo se MQGI_NONE, altrimenti valore nel campo	Valore nel campo	Valore nel campo

Nota:

- MQPMO_LOGICAL_ORDER non è valido nella chiamata MQPUT1 .
- Per il campo *MsgId* , il gestore code genera un nuovo identificativo del messaggio se MQPMO_NEW_MSG_ID o MQMI_NONE è specificato e utilizza il valore nel campo in caso contrario.
- Per il campo *CorrelId* , il gestore code genera un nuovo identificativo di correlazione se viene specificato MQPMO_NEW_CORREL_ID e utilizza il valore nel campo in caso contrario.

Quando si specifica MQPMO_LOGICAL_ORDER, il gestore code richiede che tutti i messaggi in un gruppo e i segmenti in un messaggio logico vengano inseriti con lo stesso valore nel campo *Persistence* in MQMD, ossia tutti devono essere persistenti o tutti devono essere non persistenti. Se questa condizione non viene soddisfatta, la chiamata MQPUT ha esito negativo con codice motivo MQRC_INCONSISTENT_PERSISTENCE.

L'opzione MQPMO_LOGICAL_ORDER influisce sulle unità di lavoro come segue:

- Se il primo messaggio fisico in un gruppo o in un messaggio logico viene inserito all'interno di un'unità di lavoro, tutti gli altri messaggi fisici nel gruppo o nel messaggio logico devono essere inseriti all'interno di un'unità di lavoro, se viene utilizzato lo stesso gestore code. Tuttavia, non è necessario inserirli all'interno della stessa unità di lavoro, consentendo a un gruppo di messaggi o a un messaggio logico costituito da molti messaggi fisici di essere suddiviso in due o più unità di lavoro consecutive per l'handle della coda.
- Se il primo messaggio fisico in un gruppo o un messaggio logico non viene inserito all'interno di un'unità di lavoro, nessuno degli altri messaggi fisici nel gruppo o messaggio logico può essere inserito all'interno di un'unità di lavoro, se viene utilizzato lo stesso gestore code.

Se queste condizioni non vengono soddisfatte, la chiamata MQPUT ha esito negativo con codice motivo MQRC_INCONSISTENT_UOW.

Quando viene specificato MQPMO_LOGICAL_ORDER, l'MQMD fornito nella chiamata MQPUT non deve essere minore di MQMD_VERSION_2. Se questa condizione non viene soddisfatta, la chiamata ha esito negativo con codice motivo MQRC_WRONG_MD_VERSION.

Se MQPMO_LOGICAL_ORDER non è specificato, i messaggi in gruppi e segmenti di messaggi logici possono essere inseriti in qualsiasi ordine e non è necessario inserire gruppi di messaggi completi o messaggi logici completi. È responsabilità dell'applicazione assicurarsi che i campi *GroupId*, *MsgSeqNumber*, *Offset*e *MsgFlags* abbiano valori appropriati.

Utilizzare questa tecnica per riavviare un gruppo di messaggi o un messaggio logico nel mezzo, dopo che si è verificato un malfunzionamento del sistema. Quando il sistema viene riavviato, l'applicazione può impostare i campi *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags*e *Persistence* sui valori appropriati, quindi emettere la chiamata MQPUT con MQPMO_SYNCPOINT o

MQPMO_NO_SYNCPOINT impostati come richiesto, ma senza specificare MQPMO_LOGICAL_ORDER. Se questa chiamata ha esito positivo, il gestore code conserva le informazioni sul gruppo e sul segmento e le successive chiamate MQPUT che utilizzano tale handle di coda possono specificare MQPMO_LOGICAL_ORDER come normale.

Le informazioni sul gruppo e sul segmento che il gestore code conserva per la chiamata MQPUT sono separate dal gruppo e le informazioni sul segmento che conserva per la chiamata MQGET.

Per qualsiasi handle di coda fornito, l'applicazione può combinare chiamate MQPUT che specificano MQPMO_LOGICAL_ORDER con chiamate MQPUT che non lo fanno, ma notare i seguenti punti:

- Se MQPMO_LOGICAL_ORDER non viene specificato, ogni chiamata MQPUT eseguita correttamente fa sì che il gestore code imposti le informazioni sul gruppo e sul segmento per l'handle di coda sui valori specificati dall'applicazione, sostituendo le informazioni sul segmento e sul gruppo esistenti conservate dal gestore code per l'handle di coda.
- Se MQPMO_LOGICAL_ORDER non è specificato, la chiamata non ha esito negativo se è presente un messaggio logico o un gruppo di messaggi corrente; la chiamata potrebbe avere esito positivo con un codice di completamento MQCC_WARNING. Tabella 117 a pagina 781 mostra i vari casi che possono verificarsi. In questi casi, se il codice di completamento non è MQCC_OK, il codice di errore è uno dei seguenti (come appropriato):
  - GRUPPO_INCOMPLE_MQRC
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSIST_PERSISTENZA
  - UOW MQRC_INCONSISTENT_

  Nota: Il gestore code non controlla le informazioni sul gruppo e sul segmento per la chiamata MQPUT1 .

Tabella 117. Risultato quando la chiamata MQPUT o MQCLOSE non è congruente con le informazioni sul gruppo e sul segmento

La chiamata corrente è	La chiamata precedente era MQPUT con MQPMO_LOGICAL_ORDER	La chiamata precedente era MQPUT senza MQPMO_LOGICAL_ORDER
MQPUT con MQPMO_LOGICAL_ORDER	MQCC_NON RIUSCITO	MQCC_NON RIUSCITO
MQPUT senza MQPMO_LOGICAL_ORDER	MQCC_AVVERTENZA	MQCC_OK
MQCLOSE con un gruppo non terminato o un messaggio logico	MQCC_AVVERTENZA	MQCC_OK

Per le applicazioni che inserano i messaggi e i segmenti in ordine logico, specificare MQPMO_LOGICAL_ORDER, poiché è l'opzione più semplice da utilizzare. Questa opzione allevia l'applicazione della necessità di gestire le informazioni sul gruppo e sul segmento, poiché il gestore code gestisce tali informazioni. Tuttavia, le applicazioni specializzate potrebbero richiedere un controllo maggiore rispetto a quello fornito dall'opzione MQPMO_LOGICAL_ORDER, che può essere ottenuto non specificando tale opzione; in caso contrario, è necessario assicurarsi che i campi *GroupId*, *MsgSeqNumber*, *Offset*e *MsgFlags* in MQMD siano impostati correttamente, prima di ogni chiamata MQPUT o MQPUT1 .

Ad esempio, un'applicazione che desidera inoltrare i messaggi fisici che riceve, indipendentemente dal fatto che tali messaggi si trovino in gruppi o segmenti di messaggi logici, non deve specificare MQPMO_LOGICAL_ORDER, per due motivi:

- Se i messaggi vengono richiamati e messi in ordine, specificando MQPMO_LOGICAL_ORDER si assegna un nuovo identificativo di gruppo ai messaggi, il che potrebbe rendere difficile o impossibile

per il creatore dei messaggi correlare eventuali messaggi di risposta o di report che risultano dal gruppo di messaggi.

- In una rete complessa con più percorsi tra i gestori code di invio e di ricezione, i messaggi fisici potrebbero arrivare fuori ordine. Non specificando MQPMO_LOGICAL_ORDER e MQGMO_LOGICAL_ORDER nella chiamata MQGET, l'applicazione di inoltro può richiamare e inoltrare ogni messaggio fisico non appena arriva, senza attendere il successivo in ordine logico.

Le applicazioni che generano messaggi di report per messaggi in gruppi o segmenti di messaggi logici non devono specificare MQPMO_LOGICAL_ORDER durante l'inserimento del messaggio di report.

MQPMO_LOGICAL_ORDER può essere specificato con una delle altre opzioni MQPMO_ *.

Inserimento di gruppi ordinati in modo logico in una coda cluster (MQOO_BIND_ON_GROUP)

L'opzione MQOO_BIND_ON_OPEN garantisce che tutti i messaggi da questa applicazione, e quindi tutti i gruppi, vengano instradati a una singola istanza. Questo ha lo svantaggio che il traffico dell'applicazione non è bilanciato sul carico tra più istanze di una coda cluster. Per abilitare il bilanciamento del carico di lavoro mantenendo intatti i gruppi di messaggi, è necessario impostare le seguenti opzioni:

- La chiamata MQPUT deve specificare MQPMO_LOGICAL_ORDER
- La chiamata MQOPEN deve specificare una delle seguenti opzioni:
  - Gruppo_BIND_MQOO
  - MQOO_BIND_AS_Q_DEF e la definizione della coda devono specificare DEFBIND (GROUP)

Il bilanciamento del carico di lavoro viene quindi gestito *tra gruppi* di messaggi senza richiedere MQCLOSE e MQOPEN della coda. *Tra gruppi* significa che MQMF_MSG_IN_GROUP è impostato in MQMD (v2) o MQMDE e che non vi è alcun gruppo parzialmente completo in corso. Quando un gruppo è in corso, il gestore code risolto e il nome coda nell'handle dell'oggetto vengono riutilizzati.

Se il messaggio precedente era MQPMO_LOGICAL_ORDER e / o MQMF_MSG_IN_GROUP era impostato, ma il messaggio corrente non fa parte del gruppo, la chiamata PUT ha esito negativo con MQRC_INCOMPLETE_GROUP.

Se un singolo MQPUT non specifica MQPMO_LOGICAL_ORDER e non è attivo alcun gruppo corrente, il bilanciamento del carico di lavoro viene guidato per quel messaggio (come se la chiamata MQOPEN avesse specificato MQOO_BIND_NOT_FIXED).

Non viene eseguita alcuna riassegnazione per i messaggi collegati a una destinazione utilizzando MQOO_BIND_ON_GROUP. Per ulteriori informazioni sulla riassegnazione, consultare “Gruppi di messaggi” a pagina 45.

Raggruppamento di messaggi logici

Esistono due motivi principali per utilizzare i messaggi logici in un gruppo:

- Potrebbe essere necessario elaborare i messaggi in un ordine particolare.
- Potrebbe essere necessario elaborare ogni messaggio in un gruppo in modo correlato.

In entrambi i casi, richiamare l'intero gruppo con la stessa istanza dell'applicazione di richiamo.

Ad esempio, si supponga che il gruppo sia composto da quattro messaggi logici. L'applicazione di inserimento è simile alla seguente:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

L'applicazione di richiamo specifica l'opzione MQGMO_ALL_MSGS_AVAILABLE per il primo messaggio nel gruppo. Ciò garantisce che l'elaborazione non venga avviata fino all'arrivo di tutti i messaggi all'interno del gruppo. L'opzione MQGMO_ALL_MSGS_AVAILABLE viene ignorata per i messaggi successivi all'interno del gruppo.

Quando viene richiamato il primo messaggio logico del gruppo, è possibile utilizzare MQGMO_LOGICAL_ORDER per garantire che i rimanenti messaggi logici del gruppo vengano richiamati in ordine.

Quindi, l'applicazione di acquisizione si presenta come segue:

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

Per ulteriori esempi di raggruppamento dei messaggi, consultare “Segmentazione dell'applicazione dei messaggi logici” a pagina 795 e “Mettere e ottenere un gruppo che si estende alle unità di lavoro” a pagina 783.

Attenzione: Quando si utilizza la pubblicazione / sottoscrizione per inviare messaggi a un argomento (o per inserire i messaggi in un alias dell'argomento), il raggruppamento e la segmentazione dei messaggi non sono consentiti.

Poiché le sottoscrizioni possono essere create e rimosse indipendentemente dall'attività di pubblicazione, non è possibile garantire che un sottoscrittore riceva un gruppo di messaggi completo o tutti i segmenti di un messaggio; consultare RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Per informazioni su come consentire a un'applicazione di richiedere che un gruppo di messaggi sia assegnato alla stessa istanza di destinazione per code cluster, consultare DefBind.

Mettere e ottenere un gruppo che si estende alle unità di lavoro

Nel caso precedente, i messaggi o i segmenti non possono iniziare a lasciare il nodo (se la sua destinazione è remota) o ad essere richiamati fino a quando l'intero gruppo non è stato inserito e non è stato eseguito il commit dell'unità di lavoro. Questo potrebbe non essere quello che si desidera se si impiega molto tempo per inserire l'intero gruppo o se lo spazio della coda è limitato sul nodo. Per superare questo problema, mettere il gruppo in diverse unità di lavoro.

Se il gruppo viene inserito all'interno di più unità di lavoro, è possibile che parte del gruppo esegua il commit anche quando l'applicazione di inserimento non riesce. L'applicazione deve quindi salvare le informazioni sullo stato, sottoposte a commit con ogni unità di lavoro, che può utilizzare dopo un riavvio per riprendere un gruppo incompleto. La posizione più semplice per registrare queste informazioni si trova in una coda STATUS. Se un gruppo completo è stato inserito correttamente, la coda STATUS è vuota.

Se la segmentazione è coinvolta, la logica è simile. In questo caso, **StatusInfo** deve includere *Offset*.

Di seguito viene riportato un esempio di inserimento del gruppo in diverse unità di lavoro:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
```

```
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

Se tutte le unità di lavoro sono state sottoposte a commit, l'intero gruppo è stato inserito correttamente e la coda STATUS è vuota. In caso contrario, il gruppo deve essere ripreso nel punto indicato dalle informazioni sullo stato. MQPMO_LOGICAL_ORDER non può essere utilizzato per la prima immissione, ma può essere utilizzato successivamente.

L'elaborazione del riavvio è simile alla seguente:

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

Dall'applicazione di acquisizione, è possibile che si desideri avviare l'elaborazione dei messaggi in un gruppo prima dell'arrivo dell'intero gruppo. Ciò migliora i tempi di risposta sui messaggi all'interno del gruppo e significa anche che la memoria non è richiesta per l'intero gruppo. Per realizzare i vantaggi, utilizzare diverse unità di lavoro per ciascun gruppo di messaggi. Per motivi di correzione, è necessario richiamare ogni messaggio all'interno di un'unità di lavoro.

Come per la corrispondente applicazione di inserimento, ciò richiede che le informazioni di stato vengano registrate automaticamente quando viene eseguito il commit di ciascuna unità di lavoro. Anche in questo caso, la posizione più semplice per registrare queste informazioni è in una coda STATUS. Se un gruppo completo è stato elaborato correttamente, la coda STATUS è vuota.

Nota: Per le unità di lavoro intermedie, è possibile evitare le richiami MQGET dalla coda STATUS specificando che ogni MQPUT nella coda di stato è un segmento di un messaggio (ovvero, impostando l'indicatore MQMF_SEGMENT), invece di inserire un nuovo messaggio completo per ogni unità di lavoro. Nell'ultima unità di lavoro, un segmento finale viene inserito nella coda di stato specificando MQMF_LAST_SEGMENT e le informazioni di stato vengono cancellate con un MQGET che specifica MQGMO_COMPLETE_MSG.

Durante l'elaborazione del riavvio, anziché utilizzare un singolo MQGET per ottenere un possibile messaggio di stato, sfogliare la coda di stato con MQGMO_LOGICAL_ORDER fino a raggiungere l'ultimo segmento (ossia, fino a quando non viene restituito alcun ulteriore segmento). Nella prima unità di lavoro dopo il riavvio, specificare esplicitamente l'offset quando si colloca il segmento di stato.

Nel seguente esempio, vengono considerati solo i messaggi all'interno di un gruppo, supponendo che il buffer dell'applicazione sia sempre abbastanza grande da contenere l'intero messaggio, indipendentemente dal fatto che il messaggio sia stato segmentato o meno. MQGMO_COMPLETE_MSG

viene quindi specificato su ogni MQGET. Gli stessi principi si applicano se è coinvolta la segmentazione (in questo caso, StatusInfo deve includere *Offset*).

Per semplicità, si presume che un massimo di 4 messaggi vengano richiamati all'interno di una singola UOW:

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

Se è stato eseguito il commit di tutte le unità di lavoro, l'intero gruppo è stato richiamato correttamente e la coda STATUS è vuota. In caso contrario, il gruppo deve essere ripreso nel punto indicato dalle informazioni sullo stato. MQGMO_LOGICAL_ORDER non può essere utilizzato per il primo richiamo, ma può essere utilizzato successivamente.

L'elaborazione del riavvio è simile alla seguente:

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   /* The next message on the group must be retrieved by matching
      the sequence number and group ID with those retrieved from the
      status information. */
   GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
   MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
         MQMD.GroupId      = value from Status message,
         MQMD.MsgSeqNumber = value from Status message plus 1
   msgs = 1
   /* Process this message */
   ...

   /* Now normal processing is resumed */
   /* Retrieve remaining messages in the group */
   do while ( GroupStatus == MQGS_MSG_IN_GROUP )

      /* Process up to 4 messages in the group */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                  | MQGMO_LOGICAL_ORDER
      do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
         MQGET
         msgs = msgs + 1
         /* Process this message */
         ...

      /* Have retrieved last message or 4 messages  */
      /* Update status message if not last in group */
```

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if ( GroupStatus == MQGS_MSG_IN_GROUP )
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT
msgs = 0
```

Ricezione di un messaggio particolare

Esistono diversi modi per ottenere un determinato messaggio da una coda. Questi sono: selezionando MsgId e CorrelId, selezionando GroupId, MsgSeqNumber e Offset e selezionando MsgToken. È anche possibile utilizzare una stringa di selezione quando si apre la coda.

Per ottenere un particolare messaggio da una coda, utilizzare il campo `MsgId` e `CorrelId` della struttura MQMD. Tuttavia, le applicazioni possono impostare esplicitamente questi campi, in modo che i valori specificati potrebbero non identificare un messaggio univoco. Tabella 118 a pagina 786 mostra quale messaggio viene richiamato per le possibili impostazioni di questi campi. Questi campi vengono ignorati all'immissione se si specifica MQGMO_MSG_UNDER_CURSOR nel parametro **GetMsgOpts** della chiamata MQGET.

Tabella 118. Utilizzo degli identificatori di correlazione e di messaggio

Per richiamare ...	**MsgId**	**CorrelId**
Primo messaggio nella coda	MQMI_NONE	MQCI_NONE
Primo messaggio che corrisponde a *MsgId*	Diverso da zero	MQCI_NONE
Primo messaggio che corrisponde a *CorrelId*	MQMI_NONE	Diverso da zero
Primo messaggio che corrisponde a *MsgId* e *CorrelId*	Diverso da zero	Diverso da zero

In ogni caso, *primo* indica il primo messaggio che soddisfa il criterio di selezione (a meno che non venga specificato MQGMO_BROWSE_NEXT, quando indica il *successivo* messaggio nella sequenza che soddisfa il criterio di selezione).

Al ritorno, la chiamata MQGET imposta i campi *MsgId* e *CorrelId* sugli identificativi del messaggio e della correlazione del messaggio restituito, se presenti.

Se si imposta il campo *Version* della struttura MQMD su 2, è possibile utilizzare i campi *GroupId*, *MsgSeqNumbe*r *Offset* . Tabella 119 a pagina 786 mostra quale messaggio viene richiamato per le possibili impostazioni di questi campi.

Tabella 119. Utilizzo dell'identificativo del gruppo

Per richiamare ...	**opzioni di corrispondenza**
Primo messaggio nella coda	MQMO_NONE
Primo messaggio che corrisponde a `MsgId`	ID MQMO_MATCH_MSG_ID
Primo messaggio che corrisponde a `CorrelId`	ID CORREL_MQMO_MATCH_
Primo messaggio che corrisponde a `GroupId`	ID_GROUP_MATCH_MQMO
Primo messaggio che corrisponde a `MsgSeqNumber`	NUMERO SEQ MQMO_MATCH_MSG_
Primo messaggio che corrisponde a `MsgToken`	MQMO_MATCH_MSG_TOKEN
Primo messaggio che corrisponde a `Offset`	MQMO_MATCH_OFFSET

Note:

1. MQMO_MATCH_XXX implica che il campo *XXX* nella struttura MQMD sia impostato sul valore da mettere in corrispondenza.

2. Gli indicatori MQMO possono essere utilizzati in combinazione. Ad esempio, MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER e MQMO_MATCH_OFFSET possono essere utilizzati insieme per fornire il segmento identificato dai campi `GroupId`, `MsgSeqNumbere Offset` .
3. Se si specifica MQGMO_LOGICAL_ORDER, il messaggio che si sta tentando di richiamare è interessato perché l'opzione dipende dalle informazioni di stato controllate per l'handle della coda. Per informazioni, consultare "Ordinamento logico e fisico" a pagina 774 e Opzioni.

La chiamata MQGET di solito richiama il primo messaggio da una coda. Se si specifica un particolare messaggio quando si utilizza la chiamata MQGET, il gestore code deve ricercare la coda fino a quando non trova quel messaggio. Ciò può influire sulle prestazioni dell'applicazione.

Se si utilizza la versione 2 o successiva della struttura MQGMO e non si specificano gli indicatori MQMO_MATCH_MSG_ID o MQMO_MATCH_CORREL_ID, non è necessario reimpostare i campi `MsgId` o `CorrelId` tra le MQGET.

z/OS Su IBM MQ for z/OS, l'attributo della coda `IndexType` può essere utilizzato per incrementare la velocità delle operazioni MQGET sulla coda. Per ulteriori informazioni, consultare "Tipo di indice" a pagina 791.

È possibile ottenere un messaggio specifico da una coda specificando MsgToken e MatchOption MQMO_MATCH_MSG_TOKEN nella struttura MQGMO. Il MsgToken viene restituito dalla chiamata MQPUT che in origine ha inserito tale messaggio nella coda o dalle precedenti operazioni MQGET e rimane costante a meno che il gestore code non venga riavviato.

Se si è interessati solo a un sottoinsieme di messaggi sulla coda, è possibile specificare quali messaggi si desidera elaborare utilizzando una stringa di selezione con la chiamata MQOPEN o MQSUB. MQGET richiama quindi il messaggio successivo che soddisfa tale stringa di selezione. Per ulteriori informazioni sulle stringhe di selezione, consultare "Selettori" a pagina 31.

Miglioramento delle prestazioni dei messaggi non persistenti

Quando un client richiede un messaggio da un server, invia una richiesta al server. Invia una richiesta separata per ciascuno dei messaggi che utilizza. Per migliorare le prestazioni di un client che utilizza messaggi non persistenti evitando di dover inviare questi messaggi di richiesta, è possibile configurare un client per utilizzare la *lettura anticipata*. La lettura anticipata consente l'invio di messaggi a un cliente senza che un'applicazione debba richiederli.

Quando la lettura anticipata è abilitata, i messaggi vengono inviati a un buffer di memoria sul client denominato *buffer di lettura anticipata*. Il client disporrà di un buffer di lettura anticipata per ogni coda aperta con lettura anticipata abilitata. I messaggi nel buffer di lettura anticipata non sono persistenti. Il client aggiorna periodicamente il server con informazioni sulla quantità di dati che ha utilizzato.

Quando si richiama MQOPEN con MQOO_READ_AHEAD, il client IBM MQ abilita la lettura anticipata solo se sono soddisfatte determinate condizioni. Queste condizioni includono:

- L'applicazione client deve essere compilata e collegata rispetto alle librerie client MQI IBM MQ in thread.
- Il canale del client deve utilizzare il protocollo TCP/IP
- Il canale deve avere un'impostazione SharingConversations (SHARECNV) diversa da zero nelle definizioni di canale del client e del server.

L'utilizzo della lettura anticipata può migliorare le prestazioni quando si utilizzano messaggi non persistenti da un'applicazione client. Questo miglioramento delle prestazioni è disponibile per le applicazioni MQI e JMS . Le applicazioni client che utilizzano MQGET o il consumo asincrono beneficeranno dei miglioramenti delle prestazioni quando si utilizzano messaggi non persistenti.

Non tutte le progettazioni di applicazioni client sono adatte per l'utilizzo della lettura anticipata poiché non tutte le opzioni sono supportate per l'utilizzo con la lettura anticipata e alcune opzioni sono richieste per essere congruenti tra le chiamate MQGET quando la lettura anticipata è abilitata. Se un client modifica i propri criteri di selezione tra le chiamate MQGET, i messaggi memorizzati nel buffer di lettura anticipata rimarranno bloccati nel buffer di lettura anticipata del client.

Se un backlog di messaggi bloccati con i criteri di selezione precedenti non è più richiesto, è possibile impostare un intervallo di eliminazione configurabile sul client per eliminare automaticamente tali messaggi dal client. L'intervallo di eliminazione è uno di un gruppo di opzioni di ottimizzazione lettura anticipata determinato dal client. È possibile regolare queste opzioni per soddisfare le vostre esigenze.

Se un'applicazione client viene riavviata, i messaggi nel buffer di lettura anticipata possono essere persi. Al contrario, un messaggio spostato in un buffer di lettura anticipata potrebbe essere eliminato dalla coda sottostante; ciò non ne comporta la rimozione dal buffer, quindi una chiamata MQGET che utilizza la lettura anticipata può restituire un messaggio che non esiste più.

La lettura anticipata viene eseguita solo per i collegamenti client. L'attributo viene ignorato per tutti gli altri bind.

La lettura anticipata non ha alcun effetto sull'attivazione. Nessun messaggio trigger viene generato quando un messaggio viene letto in anticipo dal client. La lettura anticipata non genera informazioni statistiche e account quando è abilitata.

Utilizzo della lettura anticipata con la messaggistica di pubblicazione - sottoscrizione

Quando un'applicazione di sottoscrizione specifica una coda di destinazione a cui vengono inviate le pubblicazioni, il valore DEFREADA della coda specificata viene utilizzato come valore di lettura anticipata predefinito.

Quando un'applicazione di sottoscrizione richiede che IBM MQ gestisca la destinazione a cui vengono inviate le pubblicazioni, una coda gestita viene creata come una coda dinamica basata su una coda modello predefinita. È il valore DEFREADA della coda modello utilizzato come valore di lettura anticipata predefinito. Le code modello predefinite SYSTEM.DURABLE.PUBLICATIONS.MODEL o SYSTEM.NONDURABLE.PUBLICATIONS.MODEL vengono utilizzati a meno che non venga definita una coda modello per questo o per un argomento principale.

Concetti correlati

“Ottimizzazione delle prestazioni per i messaggi non persistenti su AIX” a pagina 790
Se si utilizza AIX V5.3 o versioni successive, considerare l'impostazione del parametro di ottimizzazione per utilizzare le prestazioni complete per i messaggi non persistenti.

Attività correlate

“Abilitazione e disabilitazione della lettura anticipata” a pagina 790
Per impostazione predefinita la lettura anticipata è disabilitata. È possibile abilitare la lettura anticipata a livello di coda o di applicazione.

Riferimenti correlati

“Opzioni MQGET e lettura anticipata” a pagina 788
Non tutte le opzioni MQGET sono supportate quando la lettura anticipata è abilitata; alcune opzioni sono richieste per essere congruenti tra le chiamate MQGET.

Opzioni MQGET e lettura anticipata
Non tutte le opzioni MQGET sono supportate quando la lettura anticipata è abilitata; alcune opzioni sono richieste per essere congruenti tra le chiamate MQGET.

Quando si richiama MQOPEN con MQOO_READ_AHEAD, il client IBM MQ abilita la lettura anticipata solo se sono soddisfatte determinate condizioni. Queste condizioni includono:

- L'applicazione client deve essere compilata e collegata rispetto alle librerie client MQI IBM MQ in thread.
- Il canale del client deve utilizzare il protocollo TCP/IP
- Il canale deve avere un'impostazione SharingConversations (SHARECNV) diversa da zero nelle definizioni di canale del client e del server.

La seguente tabella indica quali opzioni sono supportate per l'utilizzo con la lettura anticipata e se possono essere modificate tra le chiamate MQGET.

Tabella 120. Opzioni MQGET e lettura anticipata			
Opzioni e valori MQGET	**Consentito quando la lettura anticipata è abilitata e può essere modificato tra chiamate MQGET [5]**	**Consentito quando la lettura anticipata è abilitata ma non può essere modificato tra chiamate MQGET [1]**	**Opzioni MQGET non consentite quando la lettura anticipata è abilitata [2]**
valori MQGET MQMD	MsgId [3] CorrelId [3]	Codifica CodedCharSetId	
Opzioni MQGET MQGMO	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _SOTTO_CURSORE • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_SUCCESSIVO • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENTE • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED _MSG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • SYNCPOINT MQGMO • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSORE 4 • LOCK_MQGMO • MQGMO_UNLOCK • ORDER LOGICAL_MQGMO_ • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ DISPONIBILE

Note:

1. Se queste opzioni vengono modificate tra le chiamate MQGET, viene restituito un codice motivo MQRC_OPTIONS_CHANGED.
2. Se queste opzioni vengono specificate nella prima chiamata MQGET, il read ahead è disabilitato. Se queste opzioni vengono specificate in una successiva chiamata MQGET, viene restituito il codice motivo MQRC_OPTIONS_ERROR.
3. Se un'applicazione client altera i valori MsgId e CorrelId tra le chiamate MQGET, i messaggi con i valori precedenti potrebbero già essere stati inviati al client e rimarranno nel buffer di lettura anticipata del client fino a quando non saranno utilizzati (o eliminati automaticamente).
4. MQGMO_MSG_UNDER_CURSOR non è consentito con il read ahead. La lettura anticipata è disabilitata quando vengono specificate entrambe le opzioni MQOO_BROWSE e MQOO_INPUT_SHARED o MQOO_INPUT_EXCLUSIVE durante l'apertura della coda.
5. Quando la lettura anticipata è abilitata, il primo MQGET determina se i messaggi devono essere esplorati o ricevuti da una coda. Se l'applicazione client utilizza MQGET con le opzioni modificate, come ad esempio il tentativo di esplorazione dopo una lettura iniziale o il tentativo di acquisizione dopo una lettura iniziale, viene restituito un codice motivo MQRC_OPTIONS_CHANGED.

Se un client modifica i propri criteri di selezione tra chiamate MQGET, i messaggi memorizzati nel buffer di lettura anticipata che corrispondono ai criteri di selezione iniziali non vengono utilizzati dall'applicazione client e rimangono bloccati nel buffer di lettura anticipata del client. In situazioni in cui il buffer di lettura anticipata del client contiene molti messaggi bloccati, i vantaggi associati alla lettura anticipata vengono persi e viene richiesta una richiesta separata al server per ogni messaggio utilizzato. Per determinare se la lettura anticipata viene utilizzata in modo efficiente, è possibile utilizzare il parametro di stato della connessione READA.

La lettura anticipata può essere inibita quando richiesta da un'applicazione a causa di opzioni incompatibili specificate sulla prima chiamata MQGET. In questa situazione lo stato della connessione mostra che la lettura anticipata è inibita.

Se, a causa di queste limitazioni su MQGET, si decide che una progettazione dell'applicazione client non è adatta per la lettura anticipata, specificare l'opzione MQOPEN MQOO_READ_AHEAD_NO. In alternativa, impostare il valore di lettura anticipata predefinito della coda che si sta aprendo su NO o su DISABLED.

Abilitazione e disabilitazione della lettura anticipata
Per impostazione predefinita la lettura anticipata è disabilitata. È possibile abilitare la lettura anticipata a livello di coda o di applicazione.

Informazioni su questa attività

Quando si richiama MQOPEN con MQOO_READ_AHEAD, il client IBM MQ abilita la lettura anticipata solo se sono soddisfatte determinate condizioni. Queste condizioni includono:

- L'applicazione client deve essere compilata e collegata rispetto alle librerie client MQI IBM MQ in thread.
- Il canale del client deve utilizzare il protocollo TCP/IP
- Il canale deve avere un'impostazione SharingConversations (SHARECNV) diversa da zero nelle definizioni di canale del client e del server.

Per abilitare la lettura anticipata:

- Per configurare la lettura anticipata a livello della coda, impostare l'attributo della coda DEFREADA su YES.
- Per configurare la lettura anticipata a livello di applicazione:
  - per utilizzare la lettura anticipata laddove possibile, utilizzare l'opzione MQOO_READ_AHEAD sulla chiamata della funzione MQOPEN. Non è possibile per l'applicazione client utilizzare la lettura anticipata se l'attributo della coda DEFREADA è stato impostato su DISABLED.
  - per utilizzare la sola lettura anticipata quando la lettura anticipata è abilitata su una coda, utilizzare l'opzione MQOO_READ_AHEAD_AS_Q_DEF sulla chiamata alla funzione MQOPEN.

Se la progettazione di un'applicazione client non è adatta per la lettura anticipata, è possibile disabilitarla:

- al livello della coda impostando l'attributo della coda, DEFREADA su NO se non si desidera che venga utilizzata la lettura anticipata a meno che non sia richiesta da un'applicazione client o DISABLED se non si desidera che venga utilizzata la lettura anticipata indipendentemente dal fatto che la lettura anticipata sia richiesta da una applicazione client.
- a livello dell'applicazione utilizzando l'opzione MQOO_NO_READ_AHEAD sulla chiamata della funzione MQOPEN.

Due opzioni MQCLOSE consentono di configurare cosa accade ai messaggi memorizzati nel buffer di lettura anticipata se la coda è chiusa.

- Utilizzare MQCO_IMMEDIATE per eliminare i messaggi nel buffer di lettura anticipata.
- Utilizzare MQCO_QUIESCE per garantire che i messaggi nel buffer di lettura anticipata vengano utilizzati dall'applicazione prima che la coda venga chiusa. Quando viene emesso MQCLOSE con MQCO_QUIESCE e ci sono messaggi rimanenti nel buffer di lettura anticipata, MQRC_READ_AHEAD_MSGS restituisce MQCC_WARNING.

Ottimizzazione delle prestazioni per i messaggi non persistenti su AIX
Se si utilizza AIX V5.3 o versioni successive, considerare l'impostazione del parametro di ottimizzazione per utilizzare le prestazioni complete per i messaggi non persistenti.

Per impostare il parametro di ottimizzazione in modo che diventi immediatamente effettivo, immettere il seguente comando come utente root:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Per impostare il parametro di ottimizzazione in modo che abbia effetto immediato e che persista durante i riavvii, immettere il seguente comando come utente root:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Di solito, i messaggi non persistenti vengono conservati solo in memoria, ma ci sono circostanze in cui AIX può pianificare i messaggi non persistenti da scrivere sul disco. I messaggi pianificati per essere scritti su disco non sono disponibili per MQGET fino a quando non viene completata la scrittura su disco. Il comando di ottimizzazione suggerito varia questa soglia; invece di pianificare i messaggi da scrivere su disco quando vengono accodati 16 kilobyte di dati, la scrittura su disco si verifica solo quando la memoria reale sulla macchina si avvicina al riempimento. Si tratta di una modifica globale che potrebbe interessare altri componenti software.

Su AIX, quando si utilizzano applicazioni multithread e soprattutto quando si eseguono su macchine con più processori, si consiglia di impostare AIXTHREAD_SCOPE=S nell'ID mqm `.profile` o di impostare AIXTHREAD_SCOPE=S nell'ambiente prima di avviare l'applicazione, per prestazioni migliori e una pianificazione più solida. Ad esempio:

```
export AIXTHREAD_SCOPE=S
```

L'impostazione di AIXTHREAD_SCOPE=S indica che i thread utente creati con gli attributi predefiniti vengono inseriti in un ambito di conflitto a livello di sistema. Se un thread utente viene creato con un ambito di conflitto a livello di sistema, è collegato a un thread kernel ed è pianificato dal kernel. Il thread del kernel sottostante non è condiviso con nessun altro thread utente.

Descrittori file

Quando si esegue un processo a più thread, ad esempio il processo agent, è possibile raggiungere il limite soft per i descrittori file. Questo limite fornisce il IBM MQ codice di errore `MQRC_UNEXPECTED_ERROR (2195)` e, se ci sono descrittori file sufficienti, un file IBM MQ FFST™ .

Per evitare questo problema, è possibile aumentare il limite di processo per il numero di descrittori file. Per effettuare questa operazione, modificare l'attributo `nofiles` in `/etc/security/limits` in 10.000 per l'ID utente mqm o nella stanza predefinita.

Limiti risorse di sistema

Impostare il limite di risorse di sistema per il segmento dati e il segmento stack su illimitato utilizzando i seguenti comandi in un prompt dei comandi:

```
ulimit -d unlimited
ulimit -s unlimited
```

Tipo di indice

L'attributo coda, *IndexType*, specifica il tipo di indice che il gestore code gestisce per aumentare la velocità delle operazioni MQGET sulla coda.

Nota: Supportato solo su IBM MQ for z/OS.

Sono disponibili cinque opzioni:

Valore	Descrizione
Nessuna	Nessun indice viene conservato. Utilizzarlo quando si richiamano i messaggi in modo sequenziale (consultare “Priorità” a pagina 774).
GroupID	Viene conservato un indice di identificativi di gruppo. È necessario utilizzare questo tipo di indice se si desidera un ordinamento logico dei gruppi di messaggi (consultare “Ordinamento logico e fisico” a pagina 774).
MSGID	Viene conservato un indice di identificativi messaggio. Utilizzare questa opzione quando si richiamano i messaggi utilizzando il campo *MsgId* come criterio di selezione sulla chiamata MQGET (consultare “Ricezione di un messaggio particolare” a pagina 786).
MsgToken	Viene conservato un indice di token di messaggi.

Valore	Descrizione
CorrelID	Viene conservato un indice di identificativi di correlazione. Utilizzare questa opzione quando si richiamano i messaggi utilizzando il campo *CorrelId* come criterio di selezione sulla chiamata MQGET (consultare "Ricezione di un messaggio particolare" a pagina 786).

Nota:

1. Se si sta effettuando l'indicizzazione utilizzando l'opzione MSGID o CORRELID, impostare i parametri relativi **MsgId** o **CorrelId** in MQMD. Non è vantaggioso impostare entrambi.
2. Sfoglia utilizza il meccanismo dell'indice per trovare un messaggio se una coda corrisponde a tutte le condizioni seguenti:
   - Ha il tipo di indice MSGID, CORRELID o GROUPID
   - Viene esplorato con lo stesso tipo di id
   - Ha messaggi di una sola priorit ...
3. Evitare le code (indicizzate da *MsgId* o *CorrelId*) contenenti migliaia di messaggi poiché ciò influisce sul tempo di riavvio. (Questo non si applica ai messaggi non persistenti in quanto vengono eliminati al riavvio.)
4. MSGTOKEN viene utilizzato per definire le code gestite dal gestore del carico di lavoro z/OS .

Per una descrizione completa dell'attributo **IndexType** , consultare IndexType. Per ulteriori informazioni sull'attributo **IndexType** , consultare "Considerazioni sulla progettazione e sulle prestazioni per le applicazioni z/OS" a pagina 67.

Gestione dei messaggi con lunghezza superiore a 4 MB

I messaggi possono essere troppo grandi per l'applicazione, la coda o il gestore code. A seconda dell'ambiente, IBM MQ fornisce una serie di modi per gestire i messaggi più lunghi di 4 MB.

È possibile aumentare l'attributo **MaxMsgLength** fino a 100 MB su tutti i sistemi IBM MQ alla versione V6 o successiva. Impostare questo valore per riflettere la dimensione dei messaggi che utilizzano la coda. Su sistemi IBM MQ diversi da IBM MQ for z/OS, è anche possibile:

1. Utilizzare messaggi segmentati. (I messaggi possono essere segmentati dall'applicazione o dal gestore code.)
2. Utilizzare i messaggi di riferimento.

Ciascuno di questi approcci è descritto nel resto di questa sezione.

Aumento della lunghezza massima del messaggio

L'attributo Gestore code **MaxMsgLength** definisce la lunghezza massima di un messaggio che può essere gestito da un gestore code. Allo stesso modo, l'attributo della coda **MaxMsgLength** è la lunghezza massima di un messaggio che può essere gestita da una coda. La lunghezza massima del messaggio predefinita supportata dipende dall'ambiente in cui si sta lavorando.

Se si gestiscono messaggi di grandi dimensioni, è possibile modificare questi attributi indipendentemente su piattaforme diverse da z/OS. È possibile impostare il valore dell'attributo gestore code in un intervallo tra 32768 byte e 100 MB.

Attenzione: Su IBM MQ for z/OS , l'attributo **MaxMsgLength** del gestore code è codificato a 100 MB.

Su tutte le piattaforme, è possibile impostare il valore dell'attributo della coda in un intervallo compreso tra 0 e 100 MB.

Dopo aver modificato uno o entrambi gli attributi **MaxMsgLength** , riavviare le applicazioni e i canali per rendere effettive le modifiche.

Quando vengono apportate queste modifiche, la lunghezza del messaggio deve essere inferiore o uguale sia alla coda che agli attributi **MaxMsgLength** del gestore code. Tuttavia, i messaggi esistenti potrebbero essere più lunghi di entrambi gli attributi.

Se il messaggio è troppo grande per la coda, viene restituito MQRC_MSG_TOO_BIG_FOR_Q.
Allo stesso modo, se il messaggio è troppo grande per il gestore code, viene restituito MQRC_MSG_TOO_BIG_FOR_Q_MGR.

Questo metodo di gestione di messaggi di grandi dimensioni è facile e conveniente. Tuttavia, considerare i seguenti fattori prima di utilizzarlo:

- L'uniformità tra i gestori code è ridotta. La dimensione massima dei dati del messaggio è determinata dal *MaxMsgLength* per ogni coda (incluse le code di trasmissione) in cui verrà inserito il messaggio. Questo valore è spesso impostato sul valore predefinito *MaxMsgLength*del gestore code, specialmente per le code di trasmissione. Ciò rende difficile prevedere se un messaggio è troppo grande quando deve essere inviato a un gestore code remoto.
- L'utilizzo delle risorse di sistema è aumentato. Ad esempio, le applicazioni necessitano di buffer più grandi e, su alcune piattaforme, potrebbe verificarsi un maggiore utilizzo della memoria condivisa. La memoria della coda dovrebbe essere interessata solo se effettivamente richiesta per i messaggi più grandi.
- Il batch del canale è interessato. Un messaggio di grandi dimensioni conta ancora come un solo messaggio per il conteggio batch, ma ha bisogno di più tempo per la trasmissione, aumentando così i tempi di risposta per gli altri messaggi.

Multi *Segmentazione del messaggio*

Utilizzare queste informazioni per informazioni sulla segmentazione dei messaggi. Questa funzione non è supportata su IBM MQ for z/OS o dalle applicazioni che utilizzano IBM MQ classes for JMS.

Aumentare la lunghezza massima del messaggio come descritto nell'argomento “Aumento della lunghezza massima del messaggio” a pagina 792 ha alcune implicazioni negative. Inoltre, è ancora possibile che il messaggio sia troppo grande per la coda o il gestore code. In questi casi, è possibile segmentare un messaggio. Per informazioni sui segmenti, consultare “Gruppi di messaggi” a pagina 45.

Le sezioni successive esaminano gli usi comuni per la segmentazione dei messaggi. Per l'inserimento e il richiamo in modo distruttivo, si presuppone che le chiamate MQPUT o MQGET *sempre* operino all'interno di un'unità di lavoro. Prendere sempre in considerazione l'utilizzo di questa tecnica per ridurre la possibilità che gruppi incompleti siano presenti nella rete. Il commit a fase singola viene assunto dal gestore code, ma altre tecniche di coordinamento sono ugualmente valide.

Inoltre, nelle applicazioni di richiamo, si presume che se più server elaborano la stessa coda, ogni server esegue un codice simile, in modo che un server non riesca mai a trovare un messaggio o un segmento che prevede di essere presente (poiché aveva specificato MQGMO_ALL_MSGS_AVAILABLE o MQGMO_ALL_SEGMENTS_AVAILABLE in precedenza).

Attenzione: Quando si utilizza la pubblicazione / sottoscrizione per inviare messaggi a un argomento (o per inserire i messaggi in un alias dell'argomento), il raggruppamento e la segmentazione dei messaggi non sono consentiti.

Poiché le sottoscrizioni possono essere create e rimosse indipendentemente dall'attività di pubblicazione, non è possibile garantire che un sottoscrittore riceva un gruppo di messaggi completo o tutti i segmenti di un messaggio; consultare RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Inserimento e ricezione di un messaggio segmentato che abbraccia le unità di lavoro

È possibile inserire e ottenere un messaggio segmentato che si estende su un'unità di lavoro in modo simile a “Mettere e ottenere un gruppo che si estende alle unità di lavoro” a pagina 783.

Tuttavia, non è possibile inserire o ottenere messaggi segmentati in un'unità di lavoro globale.

Multi *Segmentazione e riassemblaggio per gestore code*

Questo è lo scenario più semplice, in cui un'applicazione inserisce un messaggio che deve essere richiamato da un altro. Il messaggio potrebbe essere grande: non troppo grande per l'applicazione di inserimento o di richiamo da gestire in un singolo buffer, ma troppo grande per il gestore code o una coda in cui deve essere inserito il messaggio.

Le uniche modifiche necessarie per queste applicazioni sono che l'applicazione di inserimento autorizzi il gestore code ad eseguire la segmentazione, se necessario:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

e per l'applicazione di richiamo per richiedere al gestore code di riassemblare il messaggio se è stato segmentato:

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

In questo scenario più semplice, l'applicazione deve reimpostare il campo GroupId su MQGI_NONE prima della chiamata MQPUT, in modo che il gestore code possa generare un identificativo gruppo univoco per ogni messaggio. Se questa operazione non viene eseguita, i messaggi non correlati possono avere lo stesso identificativo del gruppo, che potrebbe successivamente portare a un'elaborazione non corretta.

Il buffer dell'applicazione deve essere abbastanza grande da contenere il messaggio riassemblato (a meno che non si includa l'opzione MQGMO_ACCEPT_TRUNCATED_MSG).

Se l'attributo MAXMSGLEN di una coda deve essere modificato per adattare la segmentazione del messaggio, considerare:

- Il segmento di messaggi minimo supportato su una coda locale è 16 byte.
- Per una coda di trasmissione, MAXMSGLEN deve includere anche lo spazio richiesto per le intestazioni. Considerare l'utilizzo di un valore di almeno 4000 byte superiore alla lunghezza massima prevista dei dati utente in qualsiasi segmento di messaggio che potrebbe essere inserito in una coda di trasmissione.

Se la conversione dei dati è necessaria, l'applicazione di richiamo potrebbe dover eseguire tale operazione specificando MQGMO_CONVERT. Ciò dovrebbe essere semplice perché l'uscita di conversione dati viene presentata con il messaggio completo. Non tentare di convertire i dati in un canale mittente se il messaggio è segmentato e il formato dei dati è tale che l'uscita di conversione dati non può eseguire la conversione su dati incompleti.

Multi *Segmentazione applicazione*

La segmentazione dell'applicazione viene utilizzata quando la segmentazione del gestore code non è adeguata o quando le applicazioni richiedono la conversione dei dati con limiti di segmento specifici.

La segmentazione dell'applicazione viene utilizzata per due motivi principali:

1. La sola segmentazione del gestore code non è sufficiente perché il messaggio è troppo grande per essere gestito in un singolo buffer dalle applicazioni.
2. La conversione dei dati deve essere eseguita dai canali del mittente e il formato è tale che l'applicazione di inserimento deve stabilire dove devono essere i confini del segmento affinché sia possibile la conversione di un singolo segmento.

Tuttavia, se la conversione dei dati non è un problema o se l'applicazione di richiamo utilizza sempre MQGMO_COMPLETE_MSG, la segmentazione del gestore code può essere consentita anche specificando MQMF_SEGMENTATION_ALLOWED. In questo esempio, l'applicazione segmenta il messaggio in quattro segmenti:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

Se non si utilizza MQPMO_LOGICAL_ORDER, l'applicazione deve impostare *Offset* e la lunghezza di ciascun segmento. In questo caso, lo stato logico non viene mantenuto automaticamente.

L'applicazione di richiamo non può garantire di avere un buffer abbastanza grande da contenere qualsiasi messaggio riassemblato. Deve quindi essere pronta a elaborare i segmenti singolarmente.

Per i messaggi che sono segmentati, questa applicazione non desidera avviare l'elaborazione di un segmento finché non sono presenti tutti i segmenti che costituiscono il messaggio logico. MQGMO_ALL_SEGMENTS_AVAILABLE è quindi specificato per il primo segmento. Se si specifica MQGMO_LOGICAL_ORDER ed è presente un messaggio logico corrente, MQGMO_ALL_SEGMENTS_AVAILABLE viene ignorato.

Dopo che il primo segmento di un messaggio logico è stato richiamato, utilizzare MQGMO_LOGICAL_ORDER per assicurarsi che i restanti segmenti del messaggio logico vengano richiamati in ordine.

Non viene data alcuna considerazione ai messaggi all'interno di gruppi differenti. Se tali messaggi si verificano, vengono elaborati nell'ordine in cui il primo segmento di ciascun messaggio si verifica sulla coda.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi *Segmentazione dell'applicazione dei messaggi logici*

I messaggi devono essere conservati in ordine logico in un gruppo e alcuni o tutti potrebbero essere così grandi da richiedere la segmentazione dell'applicazione.

In questo esempio, è necessario inserire un gruppo di quattro messaggi logici. Tutti i messaggi tranne il terzo sono grandi e richiedono la segmentazione, che viene eseguita dall'applicazione di inserimento:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

Nell'applicazione di richiamo, MQGMO_ALL_MSGS_AVAILABLE è specificata sul primo MQGET. Ciò significa che nessun messaggio o segmento di un gruppo viene richiamato fino a quando l'intero gruppo non è disponibile. Una volta richiamato il primo messaggio fisico di un gruppo, MQGMO_LOGICAL_ORDER viene utilizzato per garantire che i segmenti e i messaggi del gruppo vengano richiamati in ordine:

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
           (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

Nota: Se si specifica MQGMO_LOGICAL_ORDER e c'è un gruppo corrente, MQGMO_ALL_MSGS_AVAILABLE viene ignorato.

Messaggi di riferimento
Utilizzare queste informazioni per ulteriori informazioni sui messaggi di riferimento.

Nota: Non supportato in IBM MQ for z/OS.

Questo metodo consente ad un oggetto di grandi dimensioni di essere trasferito da un nodo ad un altro senza memorizzare l'oggetto nelle code IBM MQ sui nodi di origine o di destinazione. Ciò è particolarmente utile quando i dati esistono in un altro formato, ad esempio per le applicazioni di posta.

Per fare ciò, specificare un'exit dei messaggi ad entrambe le estremità di un canale. Per informazioni su come svolgere questa procedura, consultare “Programmi di uscita messaggi canale” a pagina 981.

IBM MQ definisce il formato di un'intestazione del messaggio di riferimento (MQRMH). Fare riferimento a MQRMH per una descrizione. Questo viene riconosciuto con un nome formato definito e potrebbe essere seguito da dati effettivi.

Per avviare un trasferimento di un LOB (large object), un'applicazione può inserire un messaggio costituito da un'intestazione del messaggio di riferimento senza dati che lo seguano. Quando questo messaggio lascia il nodo, l'exit dei messaggi richiama l'oggetto in modo appropriato e lo accoda al messaggio di riferimento. Restituisce quindi il messaggio (ora più grande di prima) all'MCA (Message Channel Agent) di invio per la trasmissione all'MCA di ricezione.

Un'altra uscita del messaggio è configurata sull'MCA di ricezione. Quando questa uscita messaggio riceve uno di questi messaggi, crea l'oggetto utilizzando i dati oggetto accodati e trasmette il messaggio di riferimento *senza* . Il messaggio di riferimento può ora essere ricevuto da un'applicazione e questa applicazione sa che l'oggetto (o almeno la parte di esso rappresentata da questo messaggio di riferimento) è stato creato in questo nodo.

La quantità massima di dati oggetto che un'uscita del messaggio di invio può accodare al messaggio di riferimento è limitata dalla lunghezza massima negoziata del messaggio per il canale. L'uscita può restituire solo un singolo messaggio all'MCA per ogni messaggio che viene passato, in modo che l'applicazione di inserimento possa inserire diversi messaggi per causare il trasferimento di un oggetto. Ogni messaggio deve identificare la lunghezza *logica* e lo scostamento dell'oggetto che deve essere accodato ad esso. Tuttavia, nei casi in cui non è possibile conoscere la dimensione totale dell'oggetto o la dimensione massima consentita dal canale, progettare l'uscita del messaggio di invio in modo che l'applicazione di inserimento immetta un singolo messaggio e l'uscita stessa immetta il messaggio successivo nella coda di trasmissione quando ha accodato più dati possibile al messaggio che è stato passato.

Prima di utilizzare questo metodo per gestire messaggi di grandi dimensioni, considerare i seguenti punti:

- L'MCA e l'uscita messaggio vengono eseguiti con un ID utente IBM MQ . L'uscita del messaggio (e quindi l'ID utente) deve accedere all'oggetto per richiamarlo all'estremità di invio o crearlo all'estremità di ricezione; ciò potrebbe essere possibile solo nei casi in cui l'oggetto è universalmente accessibile. Ciò pone un problema di sicurezza.
- Se il messaggio di riferimento con i dati di massa accodati deve viaggiare attraverso diversi gestori code prima di raggiungere la destinazione, i dati di massa sono presenti nelle code IBM MQ nei nodi intermedi. Tuttavia, in questi casi non è necessario fornire alcun supporto o uscite speciali.

- La progettazione dell'uscita del messaggio è resa difficile se è consentito il reinstradamento o l'accodamento di messaggi non recapitabili. In questi casi, le porzioni dell'oggetto potrebbero non essere in ordine.
- Quando un messaggio di riferimento arriva a destinazione, l'uscita del messaggio di ricezione crea un oggetto. Tuttavia, questo non è sincronizzato con l'unità di lavoro dell'MCA, quindi se viene eseguito il backout del batch, un altro messaggio di riferimento contenente questa stessa parte dell'oggetto arriverà in un batch successivo e l'uscita del messaggio potrebbe tentare di ricreare la stessa parte dell'oggetto. Se l'oggetto è, ad esempio, una serie di aggiornamenti del database, ciò potrebbe essere inaccettabile. In tal caso, l'exit dei messaggi deve conservare un log di cui sono stati applicati gli aggiornamenti; ciò potrebbe richiedere l'utilizzo di una coda IBM MQ .
- A seconda delle caratteristiche del tipo di oggetto, le uscite del messaggio e le applicazioni potrebbero dover cooperare per mantenere i conteggi di utilizzo, in modo che l'oggetto possa essere eliminato quando non è più necessario. Potrebbe essere richiesto anche un identificativo di istanza; viene fornito un campo per questo nell'intestazione del messaggio di riferimento (consultare MQRMH).
- Se un messaggio di riferimento viene inserito come elenco di distribuzione, l'oggetto deve essere richiamabile per ogni elenco di distribuzione risultante o per ogni singola destinazione in quel nodo. Potrebbe essere necessario mantenere i conteggi di utilizzo. Considerare anche la possibilità che un nodo possa essere il nodo finale per alcune delle destinazioni nell'elenco, ma un nodo intermedio per altre.
- I dati di massa non vengono generalmente convertiti. Ciò è dovuto al fatto che la conversione avviene *prima* che venga richiamata l'uscita messaggio. Per questo motivo, la conversione non deve essere richiesta sul canale mittente di origine. Se il messaggio di riferimento passa attraverso un nodo intermedio, i dati di massa vengono convertiti quando vengono inviati dal nodo intermedio, se richiesto.
- I messaggi di riferimento non possono essere segmentati.

Utilizzo delle strutture di MQRMH e MQMD

Consultare MQRMH e MQMD per una descrizione dei campi nell'intestazione del messaggio di riferimento e nel descrizione del messaggio.

Nella struttura MQMD, impostare il campo *Format* su MQFMT_REF_MSG_HEADER. Il formato MQHREF, quando richiesto in MQGET, viene convertito automaticamente da IBM MQ insieme ai dati di massa che seguono.

Di seguito è riportato un esempio dell'utilizzo dei campi di *DataLogicalOffset* e *DataLogicalLength* di MQRMH:

Un'applicazione di inserimento potrebbe inserire un messaggio di riferimento con:

- Nessun dato fisico
- *DataLogicalLength* = 0 (questo messaggio rappresenta l'intero oggetto)
- *DataLogicalOffset* = 0.

Supponendo che l'oggetto sia lungo 70 000 byte, l'uscita del messaggio di invio invia i primi 40 000 byte lungo il canale in un messaggio di riferimento contenente:

- 40 000 byte di dati fisici dopo MQRMH
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (dall'inizio dell'oggetto).

Posiziona quindi un altro messaggio sulla coda di trasmissione contenente:

- Nessun dato fisico
- *DataLogicalLength* = 0 (alla fine dell'oggetto). È possibile specificare un valore di 30 000.
- *DataLogicalOffset* = 40000 (a partire da questo punto).

Quando questa uscita del messaggio viene visualizzata dall'uscita del messaggio di invio, i rimanenti 30.000 byte di dati vengono aggiunti e i campi vengono impostati su:

- 30.000 byte di dati fisici dopo MQRMH
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (a partire da questo punto).

Viene impostato anche l'indicatore MQRMHF_LAST.

Per una descrizione dei programmi di esempio forniti per l'utilizzo dei messaggi di riferimento, consultare "Utilizzo dei programmi di esempio su Multiplatforms" a pagina 1061.

In attesa di messaggi

Se si desidera che un programma attenda che un messaggio arrivi su una coda, specificare l'opzione MQGMO_WAIT nel campo *Options* della struttura MQGMO.

Utilizzare il campo *WaitInterval* della struttura MQGMO per specificare il tempo massimo (in millisecondi) durante il quale si desidera che una chiamata MQGET attenda l'arrivo di un messaggio su una coda.

Se il messaggio non arriva entro questo periodo di tempo, la chiamata MQGET viene completata con il codice motivo MQRC_NO_MSG_AVAILABLE.

È possibile specificare un intervallo di attesa illimitato utilizzando la costante MQWI_UNLIMITED nel campo *WaitInterval* . Tuttavia, gli eventi al di fuori del controllo potrebbero causare un'attesa prolungata del programma, quindi utilizzare questa costante con cautela. Le applicazioni IMS non devono specificare un intervallo di attesa illimitato perché ciò impedirebbe la chiusura del sistema IMS . (Quando IMS termina, richiede la fine di tutte le regioni dipendenti). Invece, le applicazioni IMS possono specificare un periodo di attesa limitato; quindi, se la chiamata viene completata senza richiamare un messaggio dopo tale intervallo, emettere un'altra chiamata MQGET con l'opzione di attesa.

Nota: Se più di un programma è in attesa sulla stessa coda condivisa di *rimuovere* un messaggio, solo un programma viene attivato da un messaggio in arrivo. Tuttavia, se più di un programma è in attesa di visualizzare un messaggio, tutti i programmi possono essere attivati. Per ulteriori informazioni, consultare la descrizione del campo *Options* della struttura MQGMO in MQGMO.

Se lo stato della coda o del gestore code cambia prima della scadenza dell'intervallo di attesa, si verificano le azioni riportate di seguito:

- Se il gestore code entra nello stato di inattività ed è stata utilizzata l'opzione MQGMO_FAIL_IF_QUIESCING, l'attesa viene annullata e la chiamata MQGET viene completata con il codice motivo MQRC_Q_MGR_QUIESCING. Senza questa opzione, la chiamata rimane in attesa.
- z/OS Su z/OS, se la connessione (per un'applicazione CICS o IMS) entra nello stato di inattività ed è stata utilizzata l'opzione MQGMO_FAIL_IF_QUIESCING, l'attesa viene annullata e la chiamata MQGET viene completata con il codice motivo MQRC_CONN_QUIESCING. Senza questa opzione, la chiamata rimane in attesa.
- Se il gestore code viene arrestato o annullato, la chiamata MQGET viene completata con il codice motivo MQRC_Q_MGR_STOPPING o MQRC_CONNECTION_BROKEN.
- Se gli attributi della coda (o una coda in cui si risolve il nome della coda) vengono modificati in modo che le richieste get siano ora inibite, l'attesa viene annullata e la chiamata MQGET viene completata con il codice motivo MQRC_GET_INHIBITED.
- Se gli attributi della coda (o una coda in cui si risolve il nome della coda) vengono modificati in modo che l'opzione FORCE sia obbligatoria, l'attesa viene annullata e la chiamata MQGET viene completata con il codice motivo MQRC_OBJECT_CHANGED.

z/OS Se si desidera che l'applicazione attenda più di una coda, utilizzare la funzione di segnale IBM MQ for z/OS (consultare "Segnaletica" a pagina 798). Per ulteriori informazioni sulle circostanze in cui si verificano tali azioni, consultare MQGMO.

Segnaletica

La segnalazione è supportata solo su IBM MQ for z/OS.

La segnalazione è un'opzione sulla chiamata MQGET per consentire al sistema operativo di notificare (o *segnalare*) un programma quando un messaggio previsto arriva su una coda. Questa funzione è simile alla funzione *get with wait* descritta nell'argomento "In attesa di messaggi" a pagina 798 perché consente al programma di continuare con altro lavoro mentre si attende il segnale. Tuttavia, se si utilizza la segnalazione, è possibile liberare il thread dell'applicazione e fare affidamento sul sistema operativo per notificare il programma quando arriva un messaggio.

Per impostare un segnale

Per impostare un segnale, effettuare quanto segue nella struttura MQGMO che si utilizza nella chiamata MQGET:

1. Impostare l'opzione MQGMO_SET_SIGNAL nel campo *Options* .
2. Impostare la durata massima del segnale nel campo *WaitInterval* . Ciò imposta il periodo di tempo (in millisecondi) per cui si desidera che IBM MQ controlli la coda. Utilizzare il valore MQWI_UNLIMITED per specificare una durata illimitata.

   Nota: Le applicazioni IMS non devono specificare un intervallo di attesa illimitato in quanto ciò impedirebbe la chiusura del sistema IMS . (Quando IMS termina, richiede la fine di tutte le regioni dipendenti). Invece, le applicazioni IMS possono esaminare lo stato della BCE a intervalli regolari (cfr. fase 3). Un programma può avere segnali impostati su diversi handle di coda contemporaneamente:
3. Specificare l'indirizzo di *Event Control Block* (ECB) nel campo *Signal1* . Questo ti notifica il risultato del tuo segnale. La memoria della BCE deve restare disponibile fino alla chiusura della coda.

Nota: Non è possibile utilizzare l'opzione MQGMO_SET_SIGNAL con l'opzione MQGMO_WAIT.

Quando arriva il messaggio

Quando arriva un messaggio appropriato, viene restituito un codice di completamento alla BCE.

Il codice di completamento descrive uno dei seguenti:

- Il messaggio per cui si imposta il segnale è arrivato sulla coda. Il messaggio non è riservato per il programma che ha richiesto un segnale, quindi il programma deve emettere nuovamente una chiamata MQGET per richiamare il messaggio.

  Nota: Un'altra applicazione potrebbe ricevere il messaggio nel tempo che intercorre tra la ricezione del segnale e l'emissione di un'altra chiamata MQGET.
- L'intervallo di attesa impostato è scaduto e il messaggio per cui è stato impostato il segnale non è arrivato sulla coda. IBM MQ ha annullato il segnale.
- Il segnale è stato annullato. Ciò si verifica, ad esempio, se il gestore code si arresta o se l'attributo della coda viene modificato, in modo che le chiamate MQGET non siano più consentite.

Quando un messaggio adatto è già sulla coda, la chiamata MQGET viene completata nello stesso modo di una chiamata MQGET senza segnalazione. Inoltre, se viene rilevato un errore immediatamente, la chiamata viene completata e vengono impostati i codici di ritorno.

Quando la chiamata viene accettata e nessun messaggio è immediatamente disponibile, il controllo viene restituito al programma in modo che possa continuare con altro lavoro. Non è stato impostato alcun campo di output nel descrittore del messaggio, ma il parametro **CompCode** è stato impostato su MQCC_WARNING e il parametro **Reason** è stato impostato su MQRC_SIGNAL_REQUEST_ACCEPTED.

Per informazioni su ciò che IBM MQ può restituire alla tua applicazione quando effettua una chiamata MQGET utilizzando il segnale, vedi MQGET.

Se il programma non ha altro lavoro da fare mentre è in attesa che la BCE venga inviata, può attendere che la BCE utilizzi:

- Per un programma CICS Transaction Server per z/OS , il comando EXEC CICS WAIT EXTERNAL
- Per programmi batch e IMS , la macro z/OS WAIT

Se lo stato della coda o del gestore code cambia mentre il segnale è impostato (ovvero, la BCE non è ancora stata inviata), si verificano le seguenti azioni:

- Se il gestore code entra nello stato di sospensione e si utilizza l'opzione MQGMO_FAIL_IF_QUIESCING, il segnale viene annullato. La BCE viene inserita con il codice di completamento MQEC_Q_MGR_QUIESCING. Senza questa opzione, il segnale rimane impostato.
- Se il gestore code viene arrestato o annullato, il segnale viene annullato. Il segnale viene consegnato con il codice di completamento MQEC_WAIT_ANNULLATO.
- Se gli attributi della coda (o una coda in cui il nome della coda si risolve) vengono modificati in modo che le richieste di ricezione siano ora inibite, il segnale viene annullato. Il segnale viene consegnato con il codice di completamento MQEC_WAIT_ANNULLATO.

Nota:

1. Se più di un programma ha impostato un segnale sulla stessa coda condivisa per rimuovere un messaggio, solo un programma viene attivato da un messaggio in arrivo. Tuttavia, se più di un programma è in attesa di visualizzare un messaggio, tutti i programmi possono essere attivati. Le regole che il gestore code segue quando decide quali applicazioni attivare sono le stesse delle applicazioni in attesa: per ulteriori informazioni, consultare la descrizione del campo *Options* della struttura MQGMO in MQGMO - Get - message options.
2. Se è presente più di una chiamata MQGET in attesa dello stesso messaggio, con una combinazione di opzioni di attesa e di segnale, ogni chiamata in attesa viene considerata allo stesso modo. Per ulteriori informazioni, consultare la descrizione del campo *Options* della struttura MQGMO in MQGMO - Opzioni Get - message.
3. In alcune condizioni, è possibile sia per una chiamata MQGET richiamare un messaggio che per un segnale (risultante dall'arrivo dello stesso messaggio) da consegnare. Ciò significa che quando il tuo programma emette un'altra chiamata MQGET (perché il segnale è stato consegnato), potrebbe non esserci alcun messaggio disponibile. Progetta il tuo programma per testare questa situazione.

Per informazioni su come impostare un segnale, fare riferimento alla descrizione dell'opzione MQGMO_SET_SIGNAL e al campo *Signal1* in Signal1.

Backout ignorato

È possibile impedire a un programma di applicazione di immettere un loop *MQGET - error - backout* specificando l'opzione **MQGMO_MARK_SKIP_BACKOUT** nella chiamata MQGET.

Nota: Supportato solo su IBM MQ for z/OS.

Come parte di un'unità di lavoro, un programma applicativo può emettere una o più chiamate MQGET per richiamare i messaggi da una coda. Se il programma applicativo rileva un errore, può eseguire il backout dell'unità di lavoro. Ciò ripristina tutte le risorse aggiornate durante tale unità di lavoro allo stato in cui si trovavano prima dell'avvio dell'unità di lavoro e reinstalla i messaggi richiamati dalle chiamate MQGET.

Una volta ripristinati, questi messaggi sono disponibili per le successive chiamate MQGET emesse dal programma applicativo. In molti casi, ciò non causa un problema per il programma applicativo. Tuttavia, nei casi in cui l'errore che porta al backout non può essere aggirato, la reintegrazione del messaggio nella coda può far sì che il programma di applicazione immetta un loop *MQGET - error - backout* .

Per evitare questo problema, specificare l'opzione MQGMO_MARK_SKIP_BACKOUT nella chiamata MQGET. Ciò contrassegna la richiesta MQGET come non implicata nel backout avviato dall'applicazione; vale a dire, non deve essere ripristinata. L'utilizzo di questa opzione indica che quando si verifica un backout, gli aggiornamenti ad altre risorse vengono ripristinati come richiesto, ma il messaggio contrassegnato viene considerato come se fosse stato richiamato in una nuova unità di lavoro.

Il programma applicativo deve emettere una chiamata IBM MQ per eseguire il commit della nuova unità di lavoro o per eseguire il backout della nuova unità di lavoro. Ad esempio, il programma può eseguire la gestione delle eccezioni, ad esempio informare il creatore che il messaggio è stato eliminato ed eseguire il commit dell'unità di lavoro in modo da rimuovere il messaggio dalla coda. Se la nuova unità di lavoro viene ripristinata (per qualsiasi motivo), il messaggio viene ripristinato sulla coda.

All'interno di un'unità di lavoro, può esistere una sola richiesta MQGET contrassegnata come backout ignorato; tuttavia, possono essere presenti diversi altri messaggi che non sono contrassegnati come backout ignorato. Una volta che un messaggio è stato contrassegnato come backout ignorato, eventuali ulteriori chiamate MQGET all'interno dell'unità di lavoro che specificano MQGMO_MARK_SKIP_BACKOUT non riescono con codice motivo MQRC_SECOND_MARK_NOT_ALLOWED.

Nota:

1. Il messaggio contrassegnato ignora il backout solo se l'unità di lavoro che lo contiene è terminata da una richiesta dell'applicazione di backout. Se l'unità di lavoro viene ripristinata per qualsiasi altro motivo, il messaggio viene ripristinato sulla coda nello stesso modo in cui lo sarebbe se non fosse stato contrassegnato per ignorare il backout.
2. Ignora backout non è supportato all'interno delle procedure memorizzate Db2 che partecipano alle unità di lavoro controllate da RRS. Ad esempio, una chiamata MQGET con l'opzione MQGMO_MARK_SKIP_BACKOUT avrà esito negativo con il codice motivo MQRC_OPTION_ENVIRONMENT_ERROR.

Figura 63 a pagina 801 illustra una sequenza tipica di passi che un programma applicativo potrebbe contenere quando è richiesta una richiesta MQGET per ignorare il backout.

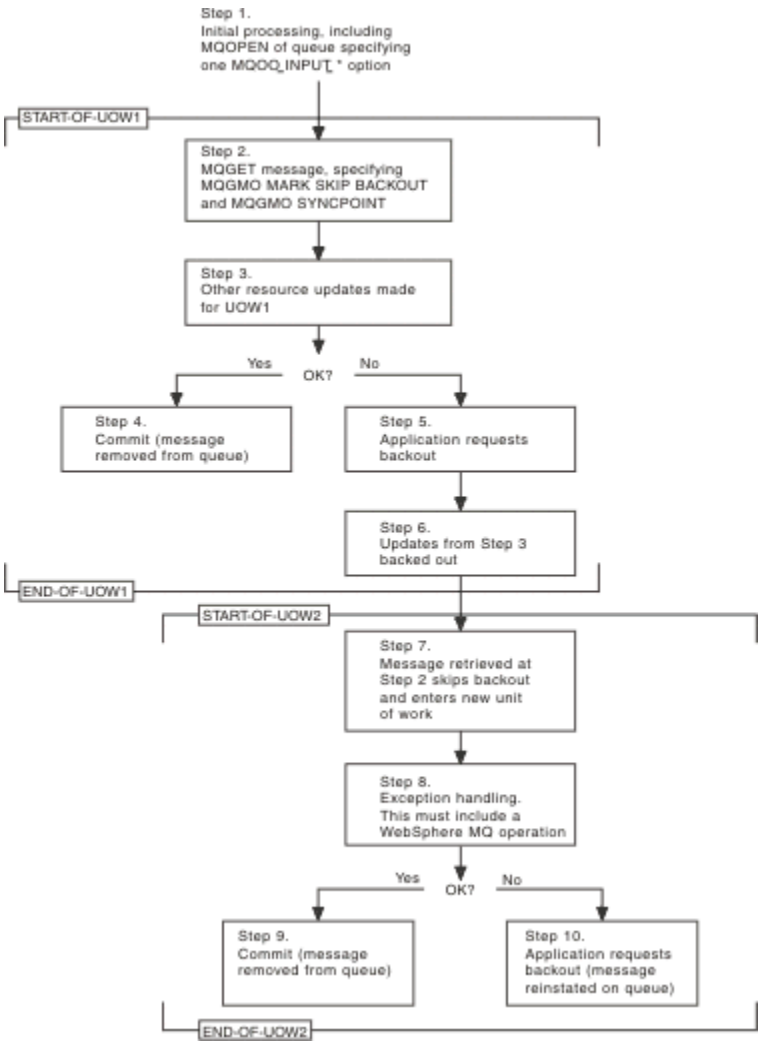


Figura 63. Il backout viene ignorato utilizzando MQGMO_MARK_SKIP_BACKOUT

I passaggi in Figura 63 a pagina 801 sono:

Passo 1
L'elaborazione iniziale si verifica all'interno della transazione, inclusa una chiamata MQOPEN per aprire una coda (specificando una delle opzioni MQOO_INPUT_ * per richiamare i messaggi dalla coda nel passo 2).

Passo 2
MQGET viene richiamato, con MQGMO_SYNCPOINT e MQGMO_MARK_SKIP_BACKOUT. MQGMO_SYNCPOINT è richiesto perché MQGET deve essere all'interno di un'unità di lavoro affinché MQGMO_MARK_SKIP_BACKOUT sia effettivo. In Figura 63 a pagina 801 questa unità di lavoro viene indicata come UOW1.

Passo 3
Altri aggiornamenti di risorse vengono effettuati come parte di UOW1. Queste possono includere ulteriori chiamate MQGET (emesse senza MQGMO_MARK_SKIP_BACKOUT).

Passo 4
Tutti gli aggiornamenti dei passi 2 e 3 vengono completati come richiesto. Il programma applicativo esegue il commit degli aggiornamenti e UOW1 termina. Il messaggio richiamato nel passo 2 viene rimosso dalla coda.

Passo 5
Alcuni degli aggiornamenti dei passi 2 e 3 non vengono completati come richiesto. Il programma applicativo richiede il backout degli aggiornamenti effettuati durante questi passi.

Passo 6
Gli aggiornamenti effettuati nel passo 3 vengono ripristinati.

Fase 7
La richiesta MQGET effettuata al passo 2 ignora il backout e diventa parte di una nuova unità di lavoro, UOW2.

Fase 8
UOW2 esegue la gestione delle eccezioni in risposta al backout di UOW1 . (Ad esempio, una chiamata MQPUT a un altra coda, che indica che si è verificato un problema che ha causato il backout di UOW1 .)

Fase 9
Il passo 8 viene completato come richiesto, il programma applicativo esegue il commit dell'attività e UOW2 termina. Poiché la richiesta MQGET fa parte di UOW2 (vedere il Passo 7), questo commit causa la rimozione del messaggio dalla coda.

Passo 10
Il passo 8 non viene completato come richiesto e il programma applicativo esegue il backout di UOW2. Poiché la richiesta di richiamo del messaggio fa parte di UOW2 (vedere il passo 7), viene eseguito il backout e reintegrato nella coda. È ora disponibile per ulteriori chiamate MQGET emesse da questo o da un altro programma applicativo (allo stesso modo di qualsiasi altro messaggio sulla coda).

Conversione dati applicazione
Quando è necessario, gli MCA convertono il descrittore del messaggio e i dati di intestazione nella serie di caratteri e nella codifica richiesti. L'estremità del collegamento (ovvero, l'MCA locale o l'MCA remoto) può eseguire la conversione.

Quando un'applicazione inserisce i messaggi in una coda, il gestore code locale aggiunge le informazioni di controllo ai descrittori di messaggi per semplificare il controllo dei messaggi quando vengono elaborati dai gestori code e dagli MCA. A seconda dell'ambiente, i campi dei dati di intestazione del messaggio vengono creati nella serie di caratteri e nella codifica del sistema locale.

Quando si spostano i messaggi tra i sistemi, a volte è necessario convertire i dati dell'applicazione nella serie di caratteri e nella codifica richieste dal sistema ricevente. Questa operazione può essere eseguita dall'interno dei programmi applicativi sul sistema di ricezione o dagli MCA sul sistema di invio. Se la conversione dei dati è supportata sul sistema ricevente, utilizzare i programmi applicativi per convertire i dati dell'applicazione, piuttosto che in base alla conversione che si è già verificata sul sistema di invio.

I dati dell'applicazione vengono convertiti all'interno di un programma applicativo quando si specifica l'opzione MQGMO_CONVERT nel campo *Options* della struttura MQGMO passata a una chiamata MQGET e quando *tutte* le seguenti istruzioni sono vere:

- I campi *CodedCharSetId* o *Encoding* impostati nella struttura MQMD associata al messaggio sulla coda differiscono dai campi *CodedCharSetId* o *Encoding* impostati nella struttura MQMD specificata sulla chiamata MQGET.
- Il campo *Format* nella struttura MQMD associata al messaggio non è MQFMT_NONE.
- Il valore *BufferLength* specificato nella chiamata MQGET non è zero.
- La lunghezza dei dati del messaggio è diversa da zero.
- Il gestore code supporta la conversione tra i campi *CodedCharSetId* e *Encoding* specificati nelle strutture MQMD associate al messaggio e alla chiamata MQGET. Consultare CodedCharSetId e Codifica per dettagli sugli identificativi della serie di caratteri codificati e sulle codifiche di macchina supportate.
- Il gestore code supporta la conversione del formato del messaggio. Se il campo *Format* della struttura MQMD associata al messaggio è uno dei formati integrati, il gestore code può convertire il messaggio. Se *Format* non è uno dei formati integrati, è necessario scrivere un'uscita di conversione dati per convertire il messaggio.

Se l'MCA di invio deve convertire i dati, specificare la parola chiave CONVERT (YES) sulla definizione di ogni canale mittente o server per cui è necessaria la conversione. Se la conversione dei dati non riesce, il messaggio viene inviato alla DLQ sul gestore code di invio e il campo *Feedback* della struttura MQDLH indica il motivo. Se il messaggio non può essere inserito nella DLQ, il canale si chiude e il messaggio non convertito rimane nella coda di trasmissione. La conversione dei dati all'interno delle applicazioni piuttosto che all'invio di MCA evita questa situazione.

Come regola, i dati nel messaggio descritti come dati *carattere* dal formato integrato o dall'uscita di conversione dati vengono convertiti dalla serie di caratteri codificati utilizzata dal messaggio a quella richiesta e i campi *numerici* vengono convertiti nella codifica richiesta.

Per ulteriori dettagli sulle convenzioni di elaborazione della conversione utilizzate durante la conversione dei formati integrati e per informazioni sulla scrittura delle proprie uscite di conversione dati, consultare “Scrittura delle uscite di conversione dati” a pagina 985. Consultare anche Lingue nazionali e Codici macchina per informazioni sulle tabelle di supporto lingua e sulle codifiche macchina supportate.

Conversione dei caratteri di nuova riga EBCDIC

Se è necessario assicurarsi che i dati inviati da una piattaforma EBCDIC a una ASCII siano identici ai dati che si ricevono di nuovo, è necessario controllare la conversione dei caratteri di nuova riga EBCDIC.

È possibile eseguire questa operazione utilizzando uno switch dipendente dalla piattaforma che forza IBM MQ ad utilizzare le tabelle di conversione non modificate, ma è necessario essere consapevoli del comportamento incongruente che potrebbe risultare.

Il problema si verifica perché il carattere di nuova riga EBCDIC non viene convertito in maniera congruente tra piattaforme o tabelle di conversione. Di conseguenza, se i dati vengono visualizzati su una piattaforma ASCII, la formattazione potrebbe non essere corretta. Ciò renderebbe difficile, ad esempio, gestire un sistema IBM i in remoto da una piattaforma ASCII utilizzando RUNMQSC.

Consultare Conversione dati per ulteriori informazioni sulla conversione dei dati in formato EBCDIC in formato ASCII.

Visualizzazione dei messaggi su una coda

Utilizzare queste informazioni per informazioni sull'esplorazione dei messaggi su una coda utilizzando la chiamata MQGET.

Per utilizzare la chiamata MQGET per sfogliare i messaggi su una coda:

1. Richiamare MQOPEN per aprire la coda per la ricerca, specificando l'opzione MQOO_BROWSE.

2. Per esaminare il primo messaggio sulla coda, richiamare MQGET con l'opzione MQGMO_BROWSE_FIRST. Per individuare il messaggio desiderato, richiamare ripetutamente MQGET con l'opzione MQGMO_BROWSE_NEXT per esaminare molti messaggi.

   È necessario impostare i campi *MsgId* e *CorrelId* della struttura MQMD su null dopo ogni chiamata MQGET per visualizzare tutti i messaggi.
3. Richiamare MQCLOSE per chiudere la coda.

Il cursore di ricerca
Quando si apre (MQOPEN) una coda per la ricerca, la chiamata stabilisce un cursore di ricerca da utilizzare con le chiamate MQGET che utilizzano una delle opzioni di ricerca. È possibile considerare il cursore di ricerca come un puntatore logico posizionato prima del primo messaggio sulla coda.

È possibile avere più di un cursore di ricerca attivo (da un singolo programma) emettendo diverse richieste MQOPEN per la stessa coda.

Quando si richiama MQGET per l'esplorazione, utilizzare una delle seguenti opzioni nella struttura MQGMO:

MQGMO_BROWSE_FIRST
Richiama una copia del primo messaggio che soddisfa le condizioni specificate nella struttura MQMD.

MQGMO_BROWSE_SUCCESSIVO
Richiama una copia del successivo messaggio che soddisfa le condizioni specificate nella propria struttura MQMD.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Richiama una copia del messaggio attualmente puntato dal cursore, ovvero quello che è stato richiamato l'ultima volta utilizzando l'opzione MQGMO_BROWSE_FIRST o MQGMO_BROWSE_NEXT.

In tutti i casi, il messaggio rimane nella coda.

Quando si apre una coda, il cursore di ricerca viene posizionato logicamente appena prima del primo messaggio sulla coda. Ciò significa che se si effettua la propria chiamata MQGET immediatamente dopo la chiamata MQOPEN, è possibile utilizzare l'opzione MQGMO_BROWSE_NEXT per sfogliare il primo messaggio; non è necessario utilizzare l'opzione MQGMO_BROWSE_FIRST.

L'ordine di copia dei messaggi dalla coda è determinato dall'attributo **MsgDeliverySequence** della coda. (Per ulteriori informazioni, consultare "L'ordine in cui i messaggi vengono richiamati da una coda" a pagina 774.)

- "Code nella sequenza FIFO (first in, first out)" a pagina 804
- "Code in sequenza di priorità" a pagina 804
- "Messaggi senza commit" a pagina 805
- "Modifica in sequenza coda" a pagina 805
- "Utilizzo dell'indice della coda" a pagina 805

Code nella sequenza FIFO (first in, first out)

Il primo messaggio in una coda in questa sequenza è il messaggio che è stato sulla coda più a lungo.

Utilizzare MQGMO_BROWSE_NEXT per leggere i messaggi in modo sequenziale nella coda. Si vedranno tutti i messaggi inseriti nella coda durante la navigazione, poiché una coda in questa sequenza ha i messaggi posizionati alla fine. Quando il cursore riconosce di aver raggiunto la fine della coda, il cursore di ricerca rimane dove si trova e restituisce con MQRC_NO_MSG_AVAILABLE. È quindi possibile lasciarlo in attesa di ulteriori messaggi o ripristinarlo all'inizio della coda con una chiamata MQGMO_BROWSE_FIRST.

Code in sequenza di priorità

Il primo messaggio in una coda in questa sequenza è il messaggio che è stato sulla coda più a lungo e che ha la priorità più alta al tempo in cui viene emessa la chiamata MQOPEN.

Utilizzare MQGMO_BROWSE_NEXT per leggere i messaggi nella coda.

Il cursore di ricerca punta al messaggio successivo, operando dalla priorità del primo messaggio per terminare con il messaggio con la priorità più bassa. Esamina tutti i messaggi inseriti nella coda durante questo periodo di tempo, purché abbiano una priorità uguale o inferiore al messaggio identificato dal cursore di ricerca corrente.

Qualsiasi messaggio inserito nella coda con priorità più alta può essere sfogliato solo da:

- Apertura della coda per la ricerca di nuovo, a quel punto viene stabilito un nuovo cursore di ricerca
- Utilizzo dell'opzione MQGMO_BROWSE_FIRST

Messaggi senza commit

Un messaggio di cui non è stato eseguito il commit non è mai visibile a una ricerca; il cursore di ricerca lo supera.

I messaggi all'interno di un'unità di lavoro non possono essere esaminati fino a quando non viene eseguito il commit dell'unità di lavoro. I messaggi non cambiano la loro posizione sulla coda quando viene eseguito il commit, quindi i messaggi ignorati e non sottoposti a commit non verranno visualizzati, anche quando *sono* sottoposti a commit, a meno che non si utilizzi l'opzione MQGMO_BROWSE_FIRST e non si lavori di nuovo nella coda.

Modifica in sequenza coda

Se la sequenza di consegna del messaggio viene modificata da priorità a FIFO mentre ci sono messaggi sulla coda, l'ordine dei messaggi che sono già accodati non viene modificato. I messaggi aggiunti alla coda successivamente, assumono la priorità predefinita della coda.

Utilizzo dell'indice della coda

Quando si sfoglia una coda indicizzata che contiene solo messaggi con una singola priorità (persistente o non persistente o entrambi), il gestore code utilizza l'indice per sfogliare quando vengono utilizzate determinate forme di ricerca.

Nota: Supportato solo su IBM MQ for z/OS.

Una delle seguenti forme di ricerca viene utilizzata quando una coda indicizzata contiene solo messaggi con priorità singola:

1. Se la coda è indicizzata da MSGID, le richieste di ricerca che passano un MSGID nella struttura MQMD vengono elaborate utilizzando l'indice per trovare il messaggio di destinazione.
2. Se la coda è indicizzata da CORRELID, le richieste di ricerca che passano un CORRELID nella struttura MQMD vengono elaborate utilizzando l'indice per trovare il messaggio di destinazione.
3. Se la coda è indicizzata da GROUPID, le richieste di esplorazione che passano un GROUPID nella struttura MQMD vengono elaborate utilizzando l'indice per trovare il messaggio di destinazione.

Se la richiesta di ricerca non passa un MSGID, CORRELID o GROUPID nella struttura MQMD, la coda viene indicizzata e viene restituito un messaggio, è necessario trovare la voce di indice per il messaggio e le informazioni al suo interno utilizzate per aggiornare il cursore di ricerca. Se si utilizza una vasta selezione di valori di indice, ciò non aggiunge un'ulteriore elaborazione significativa alla richiesta di ricerca.

Visualizzazione dei messaggi quando la lunghezza del messaggio è sconosciuta

Per sfogliare un messaggio quando non si conosce la dimensione del messaggio e non si desidera utilizzare i campi *MsgId*, *CorrelId*o *GroupId* per individuare il messaggio, è possibile utilizzare l'opzione MQGMO_BROWSE_MSG_UNDER_CURSOR:

1. Emettere un MQGET con:
   - L'opzione MQGMO_BROWSE_FIRST o MQGMO_BROWSE_NEXT
   - L'opzione MQGMO_ACCEPT_TRUNCATED_MSG
   - Lunghezza buffer zero

Nota: Se è probabile che un altro programma ricevi lo stesso messaggio, utilizzare anche l'opzione MQGMO_LOCK. MQRC_TRUNCATED_MSG_ACCEPTED deve essere restituito.

2. Utilizzare il *DataLength* restituito per allocare la memoria necessaria.
3. Emettere MQGET con MQGMO_BROWSE_MSG_UNDER_CURSOR.

Il messaggio puntato è l'ultimo che è stato richiamato; il cursore di ricerca non sarà stato spostato. È possibile scegliere di bloccare il messaggio utilizzando l'opzione MQGMO_LOCK o di sbloccare un messaggio bloccato utilizzando l'opzione MQGMO_UNLOCK.

La chiamata non riesce se nessuna opzione MQGET con le opzioni MQGMO_BROWSE_FIRST o MQGMO_BROWSE_NEXT è stata emessa correttamente da quando è stata aperta la coda.

Rimozione di un messaggio visualizzato
È possibile rimuovere dalla coda un messaggio che è già stato sfogliato, purché sia stata aperta la coda per la rimozione dei messaggi e per la navigazione. (È necessario specificare una delle opzioni MQOO_INPUT_ *, così come l'opzione MQOO_BROWSE, nella chiamata MQOPEN.)

Per rimuovere il messaggio, richiamare di nuovo MQGET, ma nel campo *Options* della struttura MQGMO, specificare MQGMO_MSG_UNDER_CURSOR.In questo caso, la chiamata MQGET ignora i campi *MsgId*, *CorrelId*e *GroupId* della struttura MQMD.

Nel periodo tra la navigazione e la rimozione, un altro programma potrebbe aver rimosso i messaggi dalla coda, incluso il messaggio sotto il cursore di esplorazione. In questo caso, la chiamata MQGET restituisce un codice motivo per indicare che il messaggio non è disponibile.

Visualizzazione dei messaggi in ordine logico

"Ordinamento logico e fisico" a pagina 774 spiega la differenza tra l'ordine logico e fisico dei messaggi su una coda. Questa distinzione è particolarmente importante quando si sfoglia una coda, poiché, in generale, i messaggi non vengono eliminati e le operazioni di ricerca non iniziano necessariamente all'inizio della coda.

Se un'applicazione sfoglia i vari messaggi di un gruppo (utilizzando l'ordine logico), è importante che l'ordine logico venga seguito per raggiungere l'inizio del gruppo successivo, poiché l'ultimo messaggio di un gruppo potrebbe verificarsi fisicamente *dopo* il primo messaggio del gruppo successivo. L'opzione MQGMO_LOGICAL_ORDER garantisce che l'ordine logico venga seguito durante la scansione di una coda.

Utilizzare MQGMO_ALL_MSGS_AVAILABLE (o MQGMO_ALL_SEGMENTS_AVAILABLE) con attenzione per le operazioni di ricerca. Considerare il caso dei messaggi logici con MQGMO_ALL_MSGS_AVAILABLE. L'effetto è che un messaggio logico è disponibile solo se sono presenti anche tutti i restanti messaggi nel gruppo. In caso contrario, il messaggio viene trasmesso. Ciò può significare che quando i messaggi mancanti arrivano in seguito, non vengono notate da un'operazione di ricerca successiva.

Ad esempio, se sono presenti i seguenti messaggi logici,

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

e viene emessa una funzione di ricerca con MQGMO_ALL_MSGS_AVAILABLE, viene restituito il primo messaggio logico del gruppo 456, lasciando il cursore di ricerca su questo messaggio logico. Se arriva il secondo (ultimo) messaggio del gruppo 123:

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
Logical message 1 (not last) of group 456  <=== browse cursor
Logical message 2 (last)     of group 456
```

e viene emessa la stessa funzione browse - next, non si nota che il gruppo 123 è ora completo, perché il primo messaggio di questo gruppo è *prima* del cursore browse.

In alcuni casi (ad esempio, se i messaggi vengono richiamati in modo distruttivo quando il gruppo è presente nella sua interezza), è possibile utilizzare MQGMO_ALL_MSGS_AVAILABLE insieme a

MQGMO_BROWSE_FIRST. In caso contrario, è necessario ripetere la scansione di esplorazione per prendere nota dei messaggi appena arrivati che sono stati persi; l'emissione di MQGMO_WAIT insieme a MQGMO_BROWSE_NEXT e MQGMO_ALL_MSGS_AVAILABLE non ne tiene conto. Ciò si verifica anche per i messaggi con priorità più elevata che potrebbero arrivare dopo che la scansione dei messaggi è stata completata.

Le sezioni successive esaminano esempi di esplorazione che trattano messaggi non segmentati; i messaggi segmentati seguono principi simili.

Visualizzazione dei messaggi in gruppi
In questo esempio, l'applicazione esamina ogni messaggio sulla coda, in ordine logico.

I messaggi sulla coda potrebbero essere raggruppati. Per i messaggi raggruppati, l'applicazione non desidera avviare l'elaborazione di alcun gruppo fino a quando non sono arrivati tutti i messaggi al suo interno. MQGMO_ALL_MSGS_AVAILABLE è quindi specificato per il primo messaggio nel gruppo; per messaggi successivi nel gruppo, questa opzione non è necessaria.

MQGMO_WAIT viene utilizzato in questo esempio. Tuttavia, anche se l'attesa può essere soddisfatta se arriva un nuovo gruppo, per i motivi riportati in "Visualizzazione dei messaggi in ordine logico" a pagina 806, non viene soddisfatta se il cursore di ricerca ha già inoltrato il primo messaggio logico in un gruppo e ora arrivano i restanti messaggi. Tuttavia, l'attesa di un intervallo adeguato assicura che l'applicazione non sia costantemente in loop durante l'attesa di nuovi messaggi o segmenti.

MQGMO_LOGICAL_ORDER viene utilizzato per garantire che la scansione sia in ordine logico.
Ciò è in contrasto con l'esempio distruttivo MQGET, dove poiché ogni gruppo viene rimosso, MQGMO_LOGICAL_ORDER non viene utilizzato quando si cerca il primo (o solo) messaggio in un gruppo.

Si presume che il buffer dell'applicazione sia sempre abbastanza grande da contenere l'intero messaggio, indipendentemente dal fatto che il messaggio sia stato segmentato o meno. MQGMO_COMPLETE_MSG viene quindi specificato su ogni MQGET.

Di seguito viene riportato un esempio di esplorazione dei messaggi logici in un gruppo:

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

Il gruppo viene ripetuto fino a quando non viene restituito MQRC_NO_MSG_AVAILABLE.

Esplorazione e recupero in modo distruttivo
In questo esempio, l'applicazione sfoglia ciascuno dei messaggi logici all'interno di un gruppo, prima di decidere se richiamare tale gruppo in modo distruttivo.

La prima parte di questo esempio è simile alla precedente. Tuttavia, in questo caso, dopo aver sfogliato un intero gruppo, decidiamo di tornare indietro e recuperarlo in modo distruttivo.

Poiché ogni gruppo viene rimosso in questo esempio, MQGMO_LOGICAL_ORDER non viene utilizzato durante la ricerca del primo o unico messaggio in un gruppo.

Di seguito viene riportato un esempio di esplorazione e recupero distruttivo:

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...
```

```
if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

Evitare la consegna ripetuta dei messaggi visualizzati
Utilizzando determinate opzioni di apertura e di ricezione messaggi, è possibile contrassegnare i messaggi come esplorati in modo che non vengano richiamati di nuovo dalle applicazioni correnti o da altre applicazioni correlate. I messaggi possono essere decontrassegnati esplicitamente o automaticamente per renderli nuovamente disponibili per la navigazione.

Se si sfogliano i messaggi su una coda, è possibile richiamarli in un ordine diverso rispetto a quello in cui li si richiamerebbe se li si ottenesse in modo distruttivo. In particolare, è possibile sfogliare lo stesso messaggio più volte, il che non è possibile se viene rimosso dalla coda. Per evitare ciò, è possibile *contrassegnare* i messaggi come visualizzati ed evitare di richiamare i messaggi contrassegnati. A volte viene indicato come *sfoglia con contrassegno*. Per contrassegnare i messaggi sfogliati, utilizzare l'opzione di richiamo del messaggio MQGMO_MARK_BROWSE_HANDLE e per richiamare solo i messaggi non contrassegnati, utilizzare MQGMO_UNMARKED_BROWSE_MSG. Se si utilizza la combinazione delle opzioni MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_HANDLE e si emettono MQGET ripetute, si richiamerà ogni messaggio sulla coda a turno. Ciò impedisce la consegna ripetuta dei messaggi anche se MQGMO_BROWSE_FIRST viene utilizzato per assicurare che i messaggi non vengano ignorati. Questa combinazione di opzioni può essere rappresentata dalla singola costante MQGMO_BROWSE_HANDLE. Quando non ci sono messaggi sulla coda che non sono stati esaminati, viene restituito MQRC_NO_MSG_AVAILABLE.

Se più applicazioni stanno sfogliando la stessa coda, possono aprire la coda con le opzioni MQOO_CO_OP e MQOO_BROWSE. L'handle dell'oggetto restituito da ogni MQOPEN è considerato parte di un gruppo cooperante. Qualsiasi messaggio restituito da una chiamata MQGET che specifichi l'opzione MQGMO_MARK_BROWSE_CO_OP viene considerato come contrassegnato per questa serie di handle cooperante.

Se un messaggio è stato contrassegnato per un certo periodo di tempo, può essere automaticamente annullato dal gestore code e reso disponibile per una nuova esplorazione. L'attributo del gestore code MsgMarkBrowseInterval fornisce il tempo, in millisecondi, per cui un messaggio deve rimanere contrassegnato per la serie di handle correlati. Un MsgMarkBrowseInterval di -1 significa che i messaggi non vengono mai contrassegnati automaticamente.

Quando il singolo processo o la serie di processi cooperativi contrassegnano i messaggi vengono arrestati, tutti i messaggi contrassegnati non vengono contrassegnati.

Esempi di navigazione cooperativa

È possibile eseguire più copie di un'applicazione dispatcher per sfogliare i messaggi su una coda e avviare un consumer in base al contenuto di ciascun messaggio. In ogni dispatcher, aprire la coda con MQOO_CO_OP. Ciò indica che i dispatcher cooperano e saranno a conoscenza dei rispettivi messaggi contrassegnati. Ogni dispatcher effettua quindi chiamate MQGET ripetute, specificando le opzioni MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_CO_OP (è possibile utilizzare la singola costante MQGMO_BROWSE_CO_OP per rappresentare questa combinazione di opzioni). Ogni applicazione dispatcher richiama quindi solo i messaggi che non sono stati già contrassegnati da altri dispatcher cooperanti. Il dispatcher inizializza un consumer e trasmette il MsgToken restituito da MQGET al consumer, che riceve in maniera distruttiva il messaggio dalla coda. Se il consumer esegue il backout di MQGET del messaggio, il messaggio è disponibile per la ridistribuzione da parte di uno dei browser, poiché non è più contrassegnato. Se il consumer non esegue un MQGET sul messaggio, dopo il passaggio di MsgMarkBrowseInterval , il gestore code annulla il contrassegno del messaggio per la serie di handle correlati e può essere ridistribuito.

Piuttosto che più copie della stessa applicazione dispatcher, è possibile disporre di un numero di applicazioni dispatcher differenti che sfogliano la coda, ciascuna adatta per l'elaborazione di un sottoinsieme di messaggi sulla coda. In ogni dispatcher, aprire la coda con MQOO_CO_OP. Ciò indica che i dispatcher cooperano e saranno a conoscenza dei rispettivi messaggi contrassegnati.

- Se l'ordine di elaborazione dei messaggi per un singolo dispatcher è importante, ogni dispatcher effettua chiamate MQGET ripetute, specificando le opzioni MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_HANDLE (o MQGMO_BROWSE_HANDLE). Se il messaggio sfogliato è adatto per l'elaborazione da parte di questo dispatcher, effettua una chiamata MQGET specificando MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP e il MsgToken restituito dalla precedente chiamata MQGET. Se la chiamata ha esito positivo, il dispatcher inizializza il consumer, passando il MsgToken .
- Se l'ordine di elaborazione dei messaggi non è importante e si prevede che il dispatcher elabori la maggior parte dei messaggi rilevati, utilizzare le opzioni MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_CO_OP (o MQGMO_BROWSE_CO_OP). Se il dispatcher sfoglia un messaggio che non può elaborare, annulla il contrassegno del messaggio richiamando MQGET con l'opzione MQMO_MATCH_MSG_TOKEN, MQGMO_UNMARK_BROWSE_CO_OP e il MsgToken precedentemente restituito.

Alcuni casi in cui la chiamata MQGET non riesce

Se alcuni attributi di una coda vengono modificati utilizzando l'opzione FORCE su un comando tra l'emissione di una chiamata MQOPEN e MQGET, la chiamata MQGET ha esito negativo e restituisce il codice motivo MQRC_OBJECT_CHANGED.

Il gestore code contrassegna l'handle dell'oggetto come non più valido. Ciò si verifica anche se le modifiche si applicano a qualsiasi coda in cui il nome della coda si risolve. Gli attributi che influenzano l'handle in questo modo sono riportati nella descrizione della chiamata MQOPEN in MQOPEN. Se la chiamata restituisce il codice motivo MQRC_OBJECT_CHANGED, chiudere la coda, riaprirla, quindi provare a richiamare nuovamente un messaggio.

Se le operazioni di richiamo sono inibite per una coda da cui si sta tentando di richiamare i messaggi (o qualsiasi coda in cui il nome della coda si risolve), la chiamata MQGET ha esito negativo e restituisce il codice motivo MQRC_GET_INHIBITED. Ciò si verifica anche se si sta utilizzando la chiamata MQGET per la navigazione. Potrebbe essere possibile ricevere un messaggio correttamente se si tenta la chiamata MQGET in un secondo momento, se la progettazione dell'applicazione è tale che altri programmi modificano regolarmente gli attributi delle code.

Se una coda dinamica (temporanea o permanente) è stata eliminata, le chiamate MQGET che utilizzano un handle di oggetto precedentemente acquisito hanno esito negativo e restituiscono il codice motivo MQRC_Q_DELETED.

Scrittura di applicazioni di pubblicazione / sottoscrizione

Avviare la scrittura delle applicazioni IBM MQ di pubblicazione / sottoscrizione.

Per una panoramica dei concetti di pubblicazione / sottoscrizione, consultare Messaggistica di pubblicazione / sottoscrizione.

Consultare i seguenti argomenti per informazioni sulla scrittura di diversi tipi di applicazioni di pubblicazione / sottoscrizione:

- “Scrittura delle applicazioni del publisher” a pagina 810
- “Scrittura delle applicazioni del sottoscrittore” a pagina 817
- “Cicli di vita di pubblicazione / sottoscrizione” a pagina 834
- “Proprietà dei messaggi di pubblicazione / sottoscrizione” a pagina 839
- “Ordinamento dei messaggi” a pagina 841
- “Intercettazione delle pubblicazioni” a pagina 841
- “Opzioni di pubblicazione” a pagina 849
- “Opzioni di sottoscrizione” a pagina 849

Concetti correlati

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

“Sviluppo di applicazioni per IBM MQ” a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

“Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51
Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

“Scrittura di un'applicazione procedurale per l'accodamento” a pagina 721
Utilizzare queste informazioni per informazioni sulla scrittura delle applicazioni di accodamento, la connessione e la disconnessione da un gestore code, la pubblicazione / sottoscrizione e l'apertura e la chiusura di oggetti.

“Scrittura di applicazioni procedurali client” a pagina 915
Cosa devi sapere per scrivere le applicazioni client su IBM MQ utilizzando un linguaggio procedurale.

“Creazione di un'applicazione procedurale” a pagina 1002
È possibile scrivere un'applicazione IBM MQ in uno dei diversi linguaggi procedurali ed eseguire l'applicazione su diverse piattaforme.

“Gestione degli errori del programma procedurale” a pagina 1040
Queste informazioni spiegano gli errori associati alle chiamate MQI delle applicazioni quando effettua una chiamata o quando il suo messaggio viene consegnato alla destinazione finale.

Attività correlate

“Utilizzo dei programmi procedurali di esempio IBM MQ” a pagina 1060
Questi programmi di esempio sono scritti in linguaggi procedurali e dimostrano gli usi tipici di MQI (Message Queue Interface). Programmi IBM MQ su diverse piattaforme.

Scrittura delle applicazioni del publisher

Inizia a scrivere applicazioni di pubblicazione studiando due esempi. Il primo è modellato il più strettamente possibile su un'applicazione point to point inserendo i messaggi in una coda e il secondo dimostra la creazione dinamica degli argomenti - un modello più comune per le applicazioni publisher.

La scrittura di una semplice applicazione publisher IBM MQ è come la scrittura di un'applicazione point to point IBM MQ che inserisce i messaggi in una coda (Tabella 121 a pagina 811). La differenza è che i messaggi MQPUT vengono inviati a un argomento, non a una coda.

Tabella 121. Modello di programma di pubblicazione / sottoscrizione IBM MQ da punto a punto.		
Fase	**Chiamata MQ punto a punto**	**Pubblica chiamata MQ**
Connetterti a un gestore code	MQCONN	MQCONN
Apri coda	MQOPEN	
Apri argomento		MQOPEN
Inserisci messaggi	MQPUT	MQPUT
Chiudi argomento		MQCLOSE
Chiudi coda	MQCLOSE	
Disconnetti dal gestore code	MQDISC	MQDISC

Per renderlo concreto, ci sono due esempi di applicazioni per pubblicare i prezzi delle azioni. Nel primo esempio (“Esempio 1: pubblicazione di un argomento fisso” a pagina 811), modellato strettamente sull'inserimento di messaggi in una coda, l'amministratore crea una definizione di argomento in modo simile alla creazione di una coda. Il programmatore codifica MQPUT per scrivere messaggi nell'argomento invece di scriverli in coda. Nel secondo esempio (“Esempio 2: pubblicazione di un argomento variabile” a pagina 814), il pattern di interazione del programma con IBM MQ è simile. La differenza è che il programmatore fornisce l'argomento in cui viene scritto il messaggio, piuttosto che l'amministratore. In pratica, ciò di solito significa che la stringa di argomenti è un contenuto definito o fornito da un'altra origine, come ad esempio l'input umano tramite un browser.

Concetti correlati

“Scrittura delle applicazioni del sottoscrittore” a pagina 817
Iniziare a scrivere le applicazioni sottoscrittore studiando tre esempi: un'applicazione IBM MQ che utilizza i messaggi da una coda, un'applicazione che crea una sottoscrizione e che non richiede alcuna conoscenza dell'accodamento e infine un esempio che utilizza sia l'accodamento che le sottoscrizioni.

Riferimenti correlati

DEFINISCI ARGOMENTO
VISUALIZZAZIONEARGOMENTO
STATOVISUALIZZAZIONEP

Esempio 1: pubblicazione di un argomento fisso

Un programma IBM MQ per illustrare la pubblicazione in un argomento definito amministrativamente.

Nota: Lo stile di codifica compatto è progettato per la leggibilità e non per l'utilizzo in produzione.

Vedere l'output in Figura 65 a pagina 812

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
               publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figura 64. Pubblicazione IBM MQ semplice per un argomento fisso.

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Figura 65. Esempio di output del primo programma di pubblicazione

Le seguenti righe di codice selezionate illustrano gli aspetti della scrittura di un'applicazione publisher per IBM MQ.

char topicNameDefault[] = "IBMSTOCKPRICE";
Nel programma viene definito un nome argomento predefinito. È possibile sovrascriverlo fornendo il nome di un oggetto argomento differente come primo argomento del programma.

MQCHAR resTopicStr[151];
`resTopicStr` è puntato da `td.ResObjectString.VSPtr` e viene utilizzato da `MQOPEN` per restituire la stringa di argomenti risolta. Aumentare la lunghezza di `resTopicStr` di uno rispetto alla lunghezza passata in `td.ResObjectString.VSBufSize` per dare spazio per la terminazione null.

memset (resTopicStr, 0, sizeof(resTopicStr));
Inizializza `resTopicStr` su valori null per garantire che la stringa di argomenti risolta restituita in `MQCHARV` sia terminata con valore null.

td.ObjectType = MQOT_TOPIC
Esiste un nuovo tipo di oggetto per la pubblicazione / sottoscrizione: l' *oggetto argomento*.

td.Version = MQOD_VERSION_4;
Per utilizzare il nuovo tipo di oggetto, è necessario utilizzare almeno la *versione 4* del descrittore oggetto.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
`topicName` è il nome di un oggetto argomento, a volte denominato oggetto argomento di gestione. Nell'esempio, l'oggetto argomento deve essere creato prima, utilizzando IBM MQ Explorer o questo comando MQSC,

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
La stringa di argomenti risolta viene rimandata nel `printf` finale nel programma. Impostare la struttura `MQCHARV ResObjectString` per IBM MQ per restituire la stringa risolta al programma.

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
Aprire l'argomento per l'emissione, proprio come aprire una coda per l'emissione.

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
Si desidera che i nuovi sottoscrittori siano in grado di ricevere la pubblicazione e specificando `MQPMO_RETAIN` nel publisher, quando si avvia un sottoscrittore riceve l'ultima pubblicazione, pubblicata prima dell'avvio del sottoscrittore, come prima pubblicazione corrispondente. L'alternativa consiste nel fornire ai sottoscrittori le pubblicazioni pubblicate solo dopo l'avvio del sottoscrittore. Inoltre un sottoscrittore ha la possibilità di rifiutare di ricevere una pubblicazione conservata specificando `MQSO_NEW_PUBLICATIONS_ONLY` nella propria sottoscrizione.

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
Aggiungere 1 alla lunghezza della stringa passata a `MQPUT` per passare il carattere di terminazione null a IBM MQ come parte del buffer del messaggio.

Che cosa dimostra il primo esempio? L'esempio imita il più strettamente possibile il modello tradizionale provato e testato per scrivere programmi punto a punto IBM MQ . Una funzione importante del modello di programmazione IBM MQ è che il programmatore non è interessato a dove vengono inviati i messaggi. L'attività del programmatore consiste nel connettersi a un gestore code e trasmettergli i messaggi che devono essere distribuiti ai destinatari. Nel paradigma point-to-point, il programmatore apre una coda (probabilmente una coda alias) configurata dall'amministratore. La coda alias instrada i messaggi a una coda di destinazione, sul gestore code locale o su un gestore code remoto. Mentre i messaggi sono in attesa di essere consegnati, vengono memorizzati nelle code tra l'origine e la destinazione.

Nel modello di pubblicazione / sottoscrizione, invece di aprire una coda, il programmatore apre un argomento. In questo esempio, l'argomento è associato a una stringa di argomenti da un amministratore. Il gestore code inoltra la pubblicazione, utilizzando le code, ai sottoscrittori locali o remoti che hanno sottoscrizioni che corrispondono alla stringa argomento della pubblicazione. Se le pubblicazioni sono conservate, il gestore code conserva l'ultima copia della pubblicazione, anche se al momento non dispone di sottoscrittori. La pubblicazione conservata è disponibile per l'inoltro ai sottoscrittori futuri.

L'applicazione publisher non ha alcun ruolo nella selezione o nell'instradamento della pubblicazione verso una destinazione; il suo compito è creare e inserire le pubblicazioni negli argomenti definiti dall'amministratore.

Questo esempio di argomento fisso è atipico di molte applicazioni di pubblicazione / sottoscrizione: è statico. Richiede un amministratore per definire le stringhe degli argomenti e modificare gli argomenti pubblicati. Generalmente, le applicazioni di pubblicazione / sottoscrizione devono conoscere alcune o tutte le strutture ad albero degli argomenti. Forse gli argomenti cambiano frequentemente, o forse anche se gli argomenti non cambiano molto, il numero di combinazioni di argomenti è elevato ed è troppo oneroso per un amministratore definire un nodo di argomenti per ogni stringa di argomenti su cui potrebbe essere necessario pubblicare. Forse le stringhe argomento non sono note prima della pubblicazione; un'applicazione publisher potrebbe utilizzare le informazioni del contenuto della pubblicazione per specificare una stringa argomento oppure potrebbe avere informazioni sulle stringhe argomento da pubblicare da un'altra origine, come ad esempio l'input umano da un browser. Per fornire stili di pubblicazione più dinamici, l'esempio successivo mostra come creare gli argomenti in modo dinamico, come parte dell'applicazione publisher.

Gli argomenti uniscono editori e sottoscrittori. La progettazione delle regole o dell'architettura per la denominazione degli argomenti e la loro organizzazione in strutture ad albero degli argomenti è un passo importante nello sviluppo di una soluzione di pubblicazione / sottoscrizione. Esaminare attentamente la misura in cui l'organizzazione della struttura ad albero degli argomenti si collega ai programmi di pubblicazione e sottoscrizione e li collega al contenuto della struttura ad albero degli argomenti. Porsi la domanda se le modifiche nella struttura ad albero degli argomenti influiscono sulle applicazioni del publisher e del sottoscrittore e su come è possibile ridurre al minimo l'effetto. Integrato nell'architettura del modello di pubblicazione / sottoscrizione IBM MQ è la nozione di un oggetto argomento di gestione che fornisce la parte root, o la struttura ad albero secondaria root, di un argomento. L'oggetto dell'argomento fornisce l'opzione di definire la parte root della struttura ad albero dell'argomento in modo amministrativo che semplifica la programmazione e le operazioni dell'applicazione e, di conseguenza, migliora la manutenibilità. Ad esempio, se si distribuiscono più applicazioni di pubblicazione / sottoscrizione che hanno strutture ad albero degli argomenti isolate, definendo amministrativamente la parte root della struttura ad albero degli argomenti, è possibile garantire l'isolamento delle strutture ad albero degli argomenti, anche se non vi è coerenza nelle convenzioni di denominazione degli argomenti adottate dalle diverse applicazioni.

In pratica, le applicazioni degli editori coprono uno spettro che va dall'utilizzo esclusivo di argomenti fissi, come in questo esempio, e argomenti variabili, come nel prossimo. “Esempio 2: pubblicazione di un argomento variabile” a pagina 814 dimostra anche la combinazione dell'utilizzo di argomenti e stringhe di argomento.

Concetti correlati

“Esempio 2: pubblicazione di un argomento variabile” a pagina 814
Un programma WebSphere MQ per illustrare la pubblicazione in un argomento definito in modo programmatico.

“Scrittura delle applicazioni del sottoscrittore” a pagina 817
Iniziare a scrivere le applicazioni sottoscrittore studiando tre esempi: un'applicazione IBM MQ che utilizza i messaggi da una coda, un'applicazione che crea una sottoscrizione e che non richiede alcuna conoscenza dell'accodamento e infine un esempio che utilizza sia l'accodamento che le sottoscrizioni.

Esempio 2: pubblicazione di un argomento variabile

Un programma WebSphere MQ per illustrare la pubblicazione in un argomento definito in modo programmatico.

Nota: Lo stile di codifica compatto è progettato per la leggibilità e non per l'utilizzo in produzione.

Consultare l'output in Figura 67 a pagina 815.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figura 66. Pubblicazione IBM MQ semplice per un argomento variabile.

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Figura 67. Esempio di output del secondo publisher

Ci sono alcuni punti da notare su questo esempio.

char topicNameDefault[] = "STOCKS";
Il nome argomento predefinito STOCKS definisce parte della stringa di argomenti. È possibile sovrascrivere questo nome argomento fornendolo come primo argomento al programma oppure eliminare l'utilizzo del nome argomento fornendo / come primo parametro.

char topicString[101] = "IBM/PRICE";
IBM/PRICE è la stringa argomento predefinita. È possibile sovrascrivere questa stringa di argomenti fornendolo come secondo argomento al programma.

Il gestore code combina la stringa di argomento fornita dall'oggetto argomento STOCKS , "NYSE", con la stringa di argomento fornita dal programma "IBM/PRICE" e inserisce un "/" tra le due stringhe di argomento. Il risultato è la stringa argomento risolta "NYSE/IBM/PRICE". La stringa di argomenti risultante è uguale a quella definita nell'oggetto argomento IBMSTOCKPRICE e ha esattamente lo stesso effetto.

L'oggetto argomento di gestione associato alla stringa argomento risolta non è necessariamente lo stesso oggetto argomento passato a MQOPEN dal publisher. IBM MQ utilizza la struttura ad albero implicita nella stringa argomento risolta per determinare quale oggetto argomento di gestione definisce gli attributi associati alla pubblicazione.

Si supponga che vi siano due oggetti argomento A e B, e A definisce l'argomento "a"e B definisce l'argomento "a/b" (Figura 68 a pagina 816). Se il programma di pubblicazione fa riferimento all'oggetto argomento A e fornisce la stringa argomento "b", risolvendo l'argomento nella stringa argomento "a/b", la pubblicazione eredita le proprietà dall'oggetto argomento B poiché l'argomento corrisponde alla stringa argomento "a/b" definita per B.

if (strcmp(argv[1],"/"))
argv[1] è il topicNamefornito facoltativamente. "/" non è valido come nome argomento; in questo caso indica che non esiste alcun nome argomento e che la stringa argomento viene fornita interamente dal programma. L'output in Figura 67 a pagina 815 mostra l'intera stringa argomento fornita dinamicamente dal programma.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
Per il caso predefinito, il topicName facoltativo deve essere creato in anticipo, utilizzando IBM MQ Explorer o questo comando MQSC:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
La stringa di argomenti è un campo MQCHARV nel descrittore di argomento

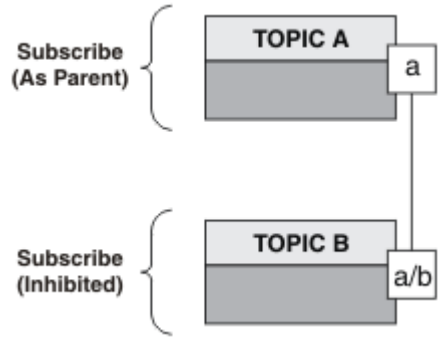


Figura 68. Associazioni oggetto argomento

Che cosa dimostra il secondo esempio? Anche se il codice è molto simile al primo esempio - effettivamente ci sono solo due righe di differenza - il risultato è un programma significativamente diverso dal primo. Il programmatore controlla le destinazioni a cui vengono inviate le pubblicazioni. Insieme all'input dell'amministratore minimo utilizzato per progettare le applicazioni del sottoscrittore,

non è necessario che siano predefiniti argomenti o code per instradare le pubblicazioni dai publisher ai sottoscrittori.

Nel paradigma della messaggistica point-to-point, le code devono essere definite prima che i messaggi siano in grado di fluire. Per la pubblicazione / sottoscrizione, non lo fanno, sebbene IBM MQ implementi la pubblicazione / sottoscrizione utilizzando il sistema di accodamento sottostante; i vantaggi della distribuzione garantita, della transazionalità e dell'associazione debole associati alla messaggistica e all'accodamento sono ereditati dalle applicazioni di pubblicazione / sottoscrizione.

Un progettista deve decidere se i programmi publisher e sottoscrittore devono essere a conoscenza della struttura ad albero degli argomenti sottostante o meno, e anche se i programmi sottoscrittore sono a conoscenza dell'accodamento o meno. Esaminare le applicazioni di esempio del sottoscrittore. Sono progettati per essere utilizzati con gli esempi del publisher, generalmente la pubblicazione e la sottoscrizione a `NYSE/IBM/PRICE`.

Concetti correlati

“Esempio 1: pubblicazione di un argomento fisso” a pagina 811
Un programma IBM MQ per illustrare la pubblicazione in un argomento definito amministrativamente.

“Scrittura delle applicazioni del sottoscrittore” a pagina 817
Iniziare a scrivere le applicazioni sottoscrittore studiando tre esempi: un'applicazione IBM MQ che utilizza i messaggi da una coda, un'applicazione che crea una sottoscrizione e che non richiede alcuna conoscenza dell'accodamento e infine un esempio che utilizza sia l'accodamento che le sottoscrizioni.

Scrittura delle applicazioni del sottoscrittore

Iniziare a scrivere le applicazioni sottoscrittore studiando tre esempi: un'applicazione IBM MQ che utilizza i messaggi da una coda, un'applicazione che crea una sottoscrizione e che non richiede alcuna conoscenza dell'accodamento e infine un esempio che utilizza sia l'accodamento che le sottoscrizioni.

In Tabella 122 a pagina 817 vengono elencati i tre stili di consumer o sottoscrittore, insieme alle sequenze di chiamate di funzione IBM MQ che li caratterizzano.

1. Il primo stile, MQ Publication Consumer, è identico a un programma MQ punto a punto che esegue solo MQGET. L'applicazione non è a conoscenza del fatto che sta consumando pubblicazioni - si tratta semplicemente di leggere i messaggi da una coda. La sottoscrizione che fa in modo che le pubblicazioni vengano instradate alla coda viene creata amministrativamente utilizzando IBM MQ Explorer o un comando.
2. Il secondo stile è il modello preferito per la maggior parte delle applicazioni del sottoscrittore. L'applicazione sottoscrittore crea la sottoscrizione e ottiene le pubblicazioni. La gestione della coda viene eseguita dal gestore code. Questo è noto come *sottoscrittore gestito*.
3. Nel terzo stile, l'applicazione del sottoscrittore è responsabile della specifica della coda che verrà utilizzata per conservare le pubblicazioni, aprire e chiudere tale coda ed emettere sottoscrizioni per riempire la coda con le pubblicazioni. È noto come *sottoscrittore non gestito*.

Un modo per comprendere questi stili è studiare i programmi `C` di esempio elencati in Tabella 122 a pagina 817 per ognuno degli stili. Gli esempi sono progettati per essere eseguiti insieme all'esempio del publisher trovato in “Scrittura delle applicazioni del publisher” a pagina 810.

Tabella 122. Pattern del programma IBM MQ point to point vs. subscribe.

Fase	**Utente del messaggio MQ**	**“Esempio 1: consumer di pubblicazione MQ” a pagina 818**	**“Esempio 2: utente MQ gestito” a pagina 820**	**“Esempio 3: sottoscrittore MQ non gestito” a pagina 825**
Connetterti a un gestore code	MQCONN	MQCONN	MQCONN	MQCONN
Apri coda	MQOPEN	MQOPEN		MQOPEN
Sottoscrivi			MQSUB	MQSUB

Tabella 122. Pattern del programma IBM MQ point to point vs. subscribe. (Continua)				
Fase	**Utente del messaggio MQ**	**"Esempio 1: consumer di pubblicazione MQ" a pagina 818**	**"Esempio 2: utente MQ gestito" a pagina 820**	**"Esempio 3: sottoscrittore MQ non gestito" a pagina 825**
Ottieni messaggi	MQGET	MQGET	MQGET	MQGET
Chiudi coda	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Chiudi sottoscrizione			MQCLOSE	MQCLOSE
Disconnetti dal gestore code	MQDISC	MQDISC	MQDISC	MQDISC

L'utilizzo di MQCLOSE è sempre facoltativo, per rilasciare le risorse, passare le opzioni MQCLOSE o solo per la simmetria con MQOPEN. Poiché è improbabile che sia necessario specificare le opzioni MQCLOSE quando la coda di sottoscrizione viene chiusa nel caso del sottoscrittore (subscriber) MQ gestito e l'argomento della simmetria non è rilevante, la coda di sottoscrizione non viene chiusa esplicitamente in Esempio 2: Sottoscrittore (subscriber) MQ gestito.

Un altro modo per comprendere i modelli di applicazione di pubblicazione / sottoscrizione è esaminare troppo le interazioni tra le diverse entità coinvolte. I diagrammi di sequenza Lifeline o UML sono un buon modo per studiare le interazioni. Tre esempi di lifeline sono descritti in "Cicli di vita di pubblicazione / sottoscrizione" a pagina 834.

Esempio 1: consumer di pubblicazione MQ
Il consumer di pubblicazione di MQ è un consumer di messaggi IBM MQ che non sottoscrive gli argomenti.

Per creare la sottoscrizione e la coda di pubblicazione per questo esempio, eseguire i seguenti comandi oppure definire gli oggetti utilizzando Esplora risorse di IBM MQ .

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

La sottoscrizione IBMSTOCKPRICESUB fa riferimento all'oggetto argomento IBMSTOCK creato per l'esempio del publisher e la coda locale STOCKTICKER. L'oggetto argomento IBMSTOCK definisce la stringa di argomenti utilizzata nella sottoscrizione, NYSE/IBM/PRICE. Si noti che l'oggetto argomento e la coda utilizzata per ricevere le pubblicazioni devono essere definiti prima della creazione della sottoscrizione.

Esistono diversi facet utili per il pattern del consumer della pubblicazione MQ :

1. Multiprocessing: condivisione del lavoro di lettura delle pubblicazioni. Le pubblicazioni vanno tutte nella singola coda associata all'argomento della sottoscrizione. Più utenti possono aprire la coda utilizzando MQOO_INPUT_SHARED.
2. Sottoscrizioni gestite centralmente. Le applicazioni non costruiscono i propri argomenti di sottoscrizione o sottoscrizioni; l'amministratore è responsabile di dove vengono inviate le pubblicazioni.
3. Concentrazione sottoscrizione: più sottoscrizioni differenti possono essere inviate a una singola coda.
4. Durata della sottoscrizione: la coda riceve tutte le pubblicazioni indipendentemente dal fatto che i consumer siano attivi o meno.
5. Migrazione e coesistenza: il codice consumer funziona allo stesso modo per uno scenario point-to-point e di pubblicazione / sottoscrizione.

La sottoscrizione crea una relazione tra la stringa argomento NYSE/IBM/PRICE e la coda STOCKTICKER. Le pubblicazioni, inclusa qualsiasi pubblicazione attualmente conservata, vengono inoltrate a STOCKTICKER dal momento in cui viene creata la sottoscrizione.

Una sottoscrizione creata in modo amministrativo può essere gestita o non gestita. Una sottoscrizione gestita diventa effettiva non appena è stata creata, proprio come una sottoscrizione non gestita. Non tutti i facet pattern sono disponibili per una sottoscrizione gestita. Vedi “Esempio 3: sottoscrittore MQ non gestito” a pagina 825

Nota: Lo stile di codifica compatto è progettato per la leggibilità e non per l'utilizzo in produzione.

I risultati sono riportati in Figura 70 a pagina 820.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figura 69. Consumer della pubblicazione MQ .

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

Figura 70. Output dal consumer della pubblicazione MQ

Ci sono un paio di suggerimenti di programmazione in linguaggio IBM MQ *C* standard di cui tenere conto:

memset(publication, 0, sizeof(publicationBuffer));
Assicurarsi che il messaggio abbia un valore null finale per una formattazione semplice utilizzando `printf`. L'esempio publisher include il valore null finale nel buffer di messaggi passato a `MQPUT` aggiungendo 1 a `strlen(publication)`. L'impostazione dei buffer `MQCHAR` su null è un buon stile di programmazione per i programmi IBM MQ C che utilizzano i buffer per memorizzare le stringhe, garantendo che un valore null segua un array di caratteri che non riempia completamente il buffer.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
Riservare un valore null alla fine del buffer di messaggi per assicurarsi che il messaggio restituito abbia un valore null finale nel caso in cui `if (messlen == strlen(publication));` sia true. Questo suggerimento integra quello precedente e garantisce che sia presente almeno un valore null in `publicationBuffer` non sovrascritto dal contenuto di `publication`.

Concetti correlati

“Esempio 2: utente MQ gestito” a pagina 820
Il sottoscrittore MQ gestito è il modello preferito per la maggior parte delle applicazioni sottoscrittore. Una sottoscrizione gestita è quella in cui IBM MQ gestisce la sottoscrizione ed esegue la registrazione e l'annullamento della registrazione per conto dell'utente. L'esempio richiede *nessuna* definizione amministrativa di code, argomenti o sottoscrizioni.

“Esempio 3: sottoscrittore MQ non gestito” a pagina 825
Il sottoscrittore non gestito è una classe importante dell'applicazione del sottoscrittore. Con esso, si combinano i vantaggi della pubblicazione / sottoscrizione con il *controllo* dell'accodamento e dell'utilizzo delle pubblicazioni. Una sottoscrizione non gestita è il punto in cui l'applicazione è responsabile. per specificare la coda in cui sono memorizzate le sottoscrizioni. L'esempio illustra diversi modi di combinazione di sottoscrizioni e code.

“Scrittura delle applicazioni del publisher” a pagina 810
Inizia a scrivere applicazioni di pubblicazione studiando due esempi. Il primo è modellato il più strettamente possibile su un'applicazione point to point inserendo i messaggi in una coda e il secondo dimostra la creazione dinamica degli argomenti - un modello più comune per le applicazioni publisher.

Esempio 2: utente MQ gestito
Il sottoscrittore MQ gestito è il modello preferito per la maggior parte delle applicazioni sottoscrittore. Una sottoscrizione gestita è quella in cui IBM MQ gestisce la sottoscrizione ed esegue la registrazione e l'annullamento della registrazione per conto dell'utente. L'esempio richiede *nessuna* definizione amministrativa di code, argomenti o sottoscrizioni.

Questo tipo più semplice di sottoscrittore gestito generalmente utilizza una sottoscrizione *non durevole* . L'esempio si concentra su una sottoscrizione non durevole. La sottoscrizione dura solo fino alla durata dell'handle di sottoscrizione da `MQSUB`. Tutte le pubblicazioni che corrispondono alla stringa dell'argomento durante il ciclo di vita della sottoscrizione vengono inviate alla coda della sottoscrizione (e possibilmente una pubblicazione conservata se l'indicatore `MQSO_NEW_PUBLICATIONS_ONLY` non è impostato o predefinito, una pubblicazione precedente corrispondente alla stringa dell'argomento è stata conservata e la pubblicazione era persistente o il gestore code non è terminato, da quando è stata creata la pubblicazione).

Con questo modello è anche possibile utilizzare una sottoscrizione *durevole* . Di solito, se viene utilizzata una sottoscrizione durevole gestita, viene eseguita per motivi di affidabilità, piuttosto che per stabilire una sottoscrizione che, senza che si verifichino errori, sopravviva al sottoscrittore. Per ulteriori informazioni sui diversi cicli di vita associati alle sottoscrizioni gestite, non gestite, durevoli e non durevoli, consultare la sezione degli argomenti correlati.

Le sottoscrizioni durevoli sono spesso associate a pubblicazioni persistenti e a sottoscrizioni non durevoli con pubblicazioni non persistenti, ma non esiste una relazione necessaria tra la durata della sottoscrizione e la persistenza della pubblicazione. Tutte e quattro le combinazioni di persistenza e durata sono possibili.

Per il caso non durevole gestito, il gestore code crea una coda di sottoscrizione che viene eliminata quando la coda viene chiusa. Le pubblicazioni vengono rimosse dalla coda quando la sottoscrizione non durevole viene chiusa.

Di seguito sono riportati gli aspetti importanti del modello non durevole gestito esemplificato da questo codice:

1. Sottoscrizione on demand: la stringa dell'argomento di sottoscrizione è dinamica. Viene fornito dall'applicazione quando viene eseguito.
2. Coda di gestione automatica: la coda di sottoscrizione è di definizione e gestione automatica.
3. Ciclo di vita della sottoscrizione a gestione automatica: le sottoscrizioni *non durevoli* esistono solo per la durata dell'applicazione del sottoscrittore.
   - Se si definisce una sottoscrizione gestita *durevole* , ne risulta una coda di sottoscrizione permanente e le pubblicazioni continuano ad essere memorizzate su di essa senza alcun programma del sottoscrittore attivo. Il gestore code elimina la coda (e cancella da essa tutte le pubblicazioni non richiamate) solo dopo che l'applicazione o l'amministratore ha scelto di eliminare la sottoscrizione. La sottoscrizione può essere eliminata utilizzando un comando di gestione o chiudendo la sottoscrizione con l'opzione `MQCO_REMOVE_SUB` .
   - Considerare l'impostazione di `SubExpiry` per le sottoscrizioni durevoli in modo che le pubblicazioni cessino di essere inviate alla coda e che il sottoscrittore possa utilizzare tutte le pubblicazioni rimanenti prima di rimuovere la sottoscrizione e far sì che il gestore code elimini la coda e le pubblicazioni rimanenti su di essa.
4. Distribuzione flessibile della stringa di argomenti: la gestione dell'argomento di sottoscrizione viene semplificato definendo la parte root della sottoscrizione utilizzando un argomento definito amministrativamente. La parte root della struttura ad albero degli argomenti viene quindi nascosta dall'applicazione. Nascondendo la parte root, è possibile distribuire un'applicazione senza che l'applicazione crei inavvertitamente una struttura ad albero degli argomenti che si sovrapponi a un'altra struttura ad albero degli argomenti creata da un'altra istanza o da un'altra applicazione.
5. Argomenti gestiti: utilizzando una stringa di argomenti in cui la prima parte corrisponde a un oggetto argomento definito in modo amministrativo, le pubblicazioni vengono gestite in base agli attributi dell'oggetto argomento.
   - Ad esempio, se la prima parte della stringa argomento corrisponde alla stringa argomento associata a un oggetto argomento in cluster, la sottoscrizione può ricevere pubblicazioni da altri membri del cluster
   - La corrispondenza selettiva di oggetti argomento definiti in modo amministrativo e sottoscrizioni definite in modo programmatico consente di combinare i vantaggi di entrambi. L'amministratore fornisce gli attributi per gli argomenti e il programmatore definisce dinamicamente i sottoargomenti senza preoccuparsi della gestione degli argomenti.
   - È la stringa di argomenti risultante che viene utilizzata per mettere in corrispondenza l'oggetto argomento che fornisce gli attributi associati all'argomento e non necessariamente l'oggetto argomento denominato in `sd.Objectname`, sebbene in genere si rivelino uguali. Consultare “Esempio 2: pubblicazione di un argomento variabile” a pagina 814.

Rendendo la sottoscrizione durevole nell'esempio, le pubblicazioni continuano ad essere inviate alla coda di sottoscrizione dopo che il sottoscrittore ha chiuso la sottoscrizione con l'opzione `MQCO_KEEP_SUB` . La coda continua a ricevere pubblicazioni quando il sottoscrittore non è attivo. È possibile sovrascrivere questo comportamento creando la sottoscrizione con l'opzione `MQSO_PUBLICATIONS_ON_REQUEST` e utilizzando `MQSUBRQ` per richiedere la pubblicazione conservata.

La sottoscrizione può essere ripresa in un secondo momento aprendo la sottoscrizione con l'opzione `MQCO_RESUME` .

È possibile utilizzare l'handle della coda, `Hobj`, restituito da `MQSUB` in vari modi. L'handle di coda viene utilizzato nell'esempio per analizzare il nome della coda di sottoscrizione. Le code gestite vengono aperte utilizzando le code modello predefinite `SYSTEM.NDURABLE.MODEL.QUEUE` o `SYSTEM.DURABLE.MODEL.QUEUE`. È possibile sovrascrivere i valori predefiniti fornendo le proprie code modello durevoli e non durevoli su un argomento per argomento come proprietà dell'oggetto argomento associato alla sottoscrizione.

Indipendentemente dagli attributi ereditati dalle code modello, non è possibile riutilizzare un handle della coda gestita per creare una sottoscrizione aggiuntiva. Inoltre, non è possibile ottenere un altro handle per la coda gestita aprendo la coda gestita una seconda volta utilizzando il nome della coda restituito. La coda si comporta come se fosse stata aperta per l'immissione esclusiva.

Le code non gestite sono più flessibili delle code gestite. È possibile, ad esempio, condividere code non gestite o definire più sottoscrizioni su una coda. L'esempio successivo, illustra come combinare le sottoscrizioni con una coda di sottoscrizioni non gestite.

Nota: Lo stile di codifica compatto è progettato per la leggibilità e non per l'utilizzo in produzione.

I risultati sono riportati in Figura 73 a pagina 824.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){               /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
}
```

Figura 71. Sottoscrittore MQ gestito - parte 1: dichiarazioni e gestione dei parametri.

Ci sono alcuni commenti aggiuntivi da fare sulle dichiarazioni in questo esempio.

MQHOBJ Hobj = MQHO_NONE;
Non è possibile aprire esplicitamente una coda di sottoscrizioni gestite non durevoli per ricevere le pubblicazioni, ma è necessario assegnare la memoria per l'oggetto che il gestore code restituisce quando apre la coda. È importante inizializzare l'handle in MQHO_OBJECT. Ciò indica al gestore code che deve restituire un handle di coda alla coda di sottoscrizione.

MQSD sd = {MQSD_DEFAULT};
Il nuovo descrittore di sottoscrizione, utilizzato in MQSUB.

MQCHAR48 qName;
Sebbene l'esempio non richieda la conoscenza della coda di sottoscrizione, l'esempio richiede il nome della coda di sottoscrizione - il bind MQINQ è un po' imbarazzante nel linguaggio C, quindi questa parte dell'esempio potrebbe essere utile da studiare.

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

Figura 72. Sottoscrittore MQ gestito - parte 2: corpo del codice.

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020    "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

Figura 73. sottoscrittore MQ

Ci sono alcuni commenti aggiuntivi da fare sul codice in questo esempio.

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
Se topicName è null o vuoto *(valore predefinito)*, il nome dell'argomento non viene utilizzato per calcolare la stringa di argomenti risolta.

sd.ObjectString.VSPtr = topicString;
Invece di utilizzare solo un oggetto argomento predefinito, in questo esempio il programma fornisce un oggetto argomento e una stringa argomento, combinati da MQSUB. Si noti che la stringa di argomenti è una struttura MQCHARV .

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Un'alternativa all'impostazione della lunghezza di un campo MQCHARV .

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
Dopo aver definito la stringa di argomenti, gli indicatori sd.Options richiedono l'attenzione più attenta. Ci sono molte opzioni, l'esempio specifica solo quelle più comunemente utilizzate. Le altre opzioni utilizzano i valori predefiniti.

1. Poiché la sottoscrizione è *non durevole*, ovvero ha una durata della sottoscrizione aperta nell'applicazione, impostare il flag MQSO_CREATE . È anche possibile impostare l'indicatore *(predefinito)* MQSO_NON_DURABLE per la leggibilità.
2. Il completamento di MQSO_CREATE è MQSO_RESUME. Entrambi gli indicatori possono essere impostati insieme; il gestore code crea una nuova sottoscrizione o riprende una sottoscrizione esistente, a seconda dei casi. Tuttavia, se si specifica MQSO_RESUME , è necessario inizializzare anche la struttura MQCHARV per sd.SubName, anche se non è presente alcuna sottoscrizione da riprendere. L'errore di inizializzazione di SubName determina un codice di ritorno di 2440: MQRC_SUB_NAME_ERROR da MQSUB.

   Nota: MQSO_RESUME viene sempre ignorato per una sottoscrizione gestita non durevole: ma specificarlo senza inizializzare la struttura MQCHARV per sd.SubName causa l'errore.
3. Inoltre, è presente un terzo indicatore che influenza la modalità di apertura della sottoscrizione, MQSO_ALTER. Date le autorizzazioni corrette, le proprietà di una sottoscrizione ripresa vengono modificate per corrispondere ad altri attributi specificati in MQSUB.

   Nota: È necessario specificare almeno uno degli indicatori MQSO_CREATE, MQSO_RESUME e MQSO_ALTER . Vedere Opzioni (MQLONG). Esistono esempi di utilizzo di tutti e tre gli indicatori in “Esempio 3: sottoscrittore MQ non gestito” a pagina 825.
4. Impostare MQSO_MANAGED per il gestore code per gestire automaticamente la sottoscrizione.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Facoltativamente, omettere l'impostazione della lunghezza di MQCHARV per le stringhe con terminazione null e utilizzare invece l'indicatore di terminazione null.

sd.ResObjectString.VSPtr = resTopicStr;
La stringa di argomenti risultante viene riportata nel primo printf del programma. Impostare MQCHARV ResObjectString per IBM MQ per restituire la stringa risolta al programma.

Nota: `resTopicStringBuffer` viene inizializzato con valori null in `memset(resTopicStr, 0, sizeof(resTopicStrBuffer))`. Le stringhe argomento restituite non terminano con un valore null finale.

`sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;`
Impostare la dimensione del buffer di `sd.ResObjectString` su un valore inferiore alla dimensione effettiva. Ciò impedisce la sovrascrittura del carattere di terminazione null fornito, nel caso in cui la stringa di argomenti risolta riempia l'intero buffer.

Nota: Non viene restituito alcun errore se la stringa dell'argomento è più lunga di `sizeof(resTopicStrBuffer)-1`. Anche se `VSLength > VSBufSiz` la lunghezza restituita in `sd.ResObjectString.VSLength` è la lunghezza della stringa completa e non necessariamente la lunghezza della stringa restituita. Verificare `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz` per confermare che la stringa dell'argomento sia completa.

`MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);`
La funzione `MQSUB` crea una sottoscrizione. Se non è durevole, probabilmente non si è interessati al suo nome, anche se è possibile esaminarne lo stato in Esplora risorse di IBM MQ . È possibile fornire il parametro `sd.SubName` come input, in modo da sapere quale nome cercare; è ovviamente necessario evitare conflitti di nomi con altre sottoscrizioni.

`MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);`
La chiusura della sottoscrizione e della coda di sottoscrizione è facoltativa. Nell'esempio la sottoscrizione è chiusa, ma non la coda. L'opzione `MQCLOSE` `MQCO_REMOVE_SUB` è comunque l'opzione predefinita in questo caso poiché la sottoscrizione non è durevole. L'utilizzo di `MQCO_KEEP_SUB` è un errore.

Nota: la sottoscrizione *queue* non viene chiusa da `MQSUB`e il relativo handle, `Hobj`, rimane valido fino a quando la coda non viene chiusa da `MQCLOSE` o `MQDISC`. Se l'applicazione termina prematuramente, la coda e la sottoscrizione vengono ripulite dal gestore code qualche tempo dopo la chiusura dell'applicazione.

Concetti correlati

“Esempio 1: consumer di pubblicazione MQ” a pagina 818
Il consumer di pubblicazione di MQ è un consumer di messaggi IBM MQ che non sottoscrive gli argomenti.

“Esempio 3: sottoscrittore MQ non gestito” a pagina 825
Il sottoscrittore non gestito è una classe importante dell'applicazione del sottoscrittore. Con esso, si combinano i vantaggi della pubblicazione / sottoscrizione con il *controllo* dell'accodamento e dell'utilizzo delle pubblicazioni. Una sottoscrizione non gestita è il punto in cui l'applicazione è responsabile. per specificare la coda in cui sono memorizzate le sottoscrizioni. L'esempio illustra diversi modi di combinazione di sottoscrizioni e code.

“Scrittura delle applicazioni del publisher” a pagina 810
Inizia a scrivere applicazioni di pubblicazione studiando due esempi. Il primo è modellato il più strettamente possibile su un'applicazione point to point inserendo i messaggi in una coda e il secondo dimostra la creazione dinamica degli argomenti - un modello più comune per le applicazioni publisher.

Esempio 3: sottoscrittore MQ non gestito
Il sottoscrittore non gestito è una classe importante dell'applicazione del sottoscrittore. Con esso, si combinano i vantaggi della pubblicazione / sottoscrizione con il *controllo* dell'accodamento e dell'utilizzo delle pubblicazioni. Una sottoscrizione non gestita è il punto in cui l'applicazione è responsabile. per specificare la coda in cui sono memorizzate le sottoscrizioni. L'esempio illustra diversi modi di combinazione di sottoscrizioni e code.

Il pattern non gestito è più comunemente associato a sottoscrizioni *durevoli* rispetto a *non durevoli*. In genere, il ciclo di vita di una sottoscrizione creata da un sottoscrittore non gestito è indipendente dal ciclo di vita dell'applicazione di sottoscrizione stessa. Rendendo la sottoscrizione durevole la sottoscrizione riceve le pubblicazioni anche quando non è attiva alcuna applicazione di sottoscrizione.

È possibile creare sottoscrizioni *gestite* durevoli per ottenere lo stesso risultato, ma alcune applicazioni richiedono maggiore flessibilità e controllo su code e messaggi rispetto a quanto è possibile con una sottoscrizione gestita. Per una sottoscrizione gestita durevole, il gestore code crea una coda

permanente per le pubblicazioni che corrispondono all'argomento della sottoscrizione. Elimina la coda e le pubblicazioni associate quando la sottoscrizione viene eliminata.

Generalmente, le sottoscrizioni *gestite* durevoli vengono utilizzate se il ciclo di vita dell'applicazione e la sottoscrizione sono essenzialmente uguali, ma difficili da garantire. Rendendo la sottoscrizione durevole e facendo in modo che il publisher crei pubblicazioni persistenti, non vi sono messaggi persi nel caso in cui il gestore code o il sottoscrittore terminino prematuramente e debbano essere ripristinati.

Per applicazioni non JMS o per applicazioni JMS che non utilizzano una sottoscrizione condivisa, il gestore code aprirà implicitamente la coda di sottoscrizione gestita durevole per un sottoscrittore in modo che l'elaborazione condivisa della coda non sia possibile. Inoltre, a meno che l'applicazione non stia utilizzando le sottoscrizioni condivise JMS, non è possibile creare più di una sottoscrizione per ciascuna coda gestita e le code potrebbero risultare più difficili da gestire poiché si ha un minore controllo sui nomi delle code. Per questi motivi, considerare se il sottoscrittore *non gestito* MQ è più adatto per le applicazioni che richiedono sottoscrizioni durevoli rispetto al sottoscrittore *gestito* MQ .

Il codice in Figura 76 a pagina 831 dimostra un pattern di sottoscrizione durevole non gestito. Per l'illustrazione il codice crea anche sottoscrizioni non gestite e non durevoli. Questo esempio illustra i seguenti facet di pattern:

- Sottoscrizioni on demand: le stringhe dell'argomento di sottoscrizione sono dinamiche. Vengono forniti dall'applicazione quando viene eseguita.
- Gestione degli argomenti di sottoscrizione semplificata: la gestione degli argomenti di sottoscrizione viene semplificata definendo la parte root della stringa degli argomenti di sottoscrizione utilizzando un argomento definito amministrativamente. Questa operazione nasconde la parte root della struttura ad albero dell'argomento dall'applicazione. Nascondendo la parte root, un sottoscrittore può essere distribuito a diverse strutture ad albero degli argomenti.
- Gestione flessibile della sottoscrizione: è possibile definire una sottoscrizione in modo amministrativo o crearla su richiesta in un programma sottoscrittore. Non esiste alcuna differenza tra le sottoscrizioni create in modo amministrativo e in modo programmatico, tranne un attributo che mostra come è stata creata la sottoscrizione. Esiste un terzo tipo di sottoscrizione che viene creato automaticamente dal gestore code per la distribuzione delle sottoscrizioni. Tutte le sottoscrizioni vengono visualizzate in Esplora risorse di IBM MQ .
- Associazione flessibile di sottoscrizioni con code: una coda locale predefinita è associata a una sottoscrizione dalla funzione MQSUB . Esistono diversi modi per utilizzare MQSUB per associare le sottoscrizioni alle code:
  - Associare una sottoscrizione a una coda con *nessuna* sottoscrizione esistente, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Associare una *nuova* sottoscrizione a una coda con sottoscrizioni esistenti, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Spostare una sottoscrizione esistente in una diversa coda, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Riprendere una sottoscrizione esistente associata a una coda esistente, `MQSO_RESUME + (Hobj = MQHO_NONE)` o `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - Combinando `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` in diverse combinazioni, è possibile soddisfare i diversi stati di input della sottoscrizione e della coda senza dover codificare più versioni di MQSUB con valori differenti `sd.Options` .
  - In alternativa, codificando una scelta specifica di `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` il gestore code restituisce un errore (Tabella 123 a pagina 828) se gli stati della sottoscrizione e della coda forniti come input per MQSUB sono incongruenti con il valore di `sd.Options`. Figura 82 a pagina 834 mostra i risultati dell'emissione di MQSUB per la sottoscrizione X con diverse impostazioni individuali dell'indicatore `sd.Options` e della trasmissione di tre diversi handle di oggetti.

  Esplorare diversi input al programma di esempio in Figura 75 a pagina 830 per acquisire familiarità con questi diversi tipi di errore. Un errore comune, `RC = 2440`, che non è incluso nei casi elencati nella

tabella, è un errore del nome della sottoscrizione. è generalmente causato dal passaggio di un nome sottoscrizione null o non valido con MQSO_RESUME o MQSO_ALTER.

- Multiprocessing: è possibile condividere tra molti consumatori il lavoro di lettura delle pubblicazioni. Le pubblicazioni vanno tutte nella singola coda associata all'argomento della sottoscrizione. I consumatori possono scegliere di aprire la coda direttamente utilizzando MQOPEN o di riprendere la sottoscrizione utilizzando MQSUB.
- Concentrazione sottoscrizione: è possibile creare più sottoscrizioni sulla stessa coda. Prestare attenzione con questa capacità in quanto può portare alla sovrapposizione di sottoscrizioni e ricevere la stessa pubblicazione più volte. L'opzione MQSO_GROUP_SUB elimina le pubblicazioni duplicate causate dalla sovrapposizione delle sottoscrizioni.
- Separazione tra sottoscrittore e consumatore: oltre ai tre modelli illustrati negli esempi, un altro modello consiste nel separare il consumatore dal sottoscrittore. Si tratta di una variazione del sottoscrittore di MQ non gestito, ma piuttosto che emettere MQOPEN e MQSUB nello stesso programma, un programma sottoscrive le pubblicazioni e un altro programma le utilizza. Ad esempio, il sottoscrittore potrebbe far parte di un cluster di pubblicazione / sottoscrizione e l'utente collegato a un gestore code esterno al cluster del gestore code. L'utente riceve le pubblicazioni tramite l'accodamento distribuito standard definendo la coda di sottoscrizione come una definizione di coda remota.

Comprendere il comportamento di MQSO_CREATE | MQSO_RESUME | MQSO_ALTER è importante, soprattutto se si intende semplificare il codice utilizzando combinazioni di queste opzioni. Esaminare la tabella Tabella 123 a pagina 828 che mostra i risultati della trasmissione di diversi handle di code a MQSUB e i risultati dell'esecuzione del programma di esempio mostrato in Figura 77 a pagina 832 a Figura 82 a pagina 834.

Lo scenario utilizzato per costruire la tabella ha una sottoscrizione X e due code, A e B. Il parametro del nome sottoscrizione sd.SubName è impostato su X, il nome di una sottoscrizione collegata alla coda A. Alla coda B non è allegata alcuna sottoscrizione.

In Tabella 123 a pagina 828, MQSUB viene passata la sottoscrizione X e l'handle di coda alla coda A. I risultati delle opzioni di sottoscrizione sono i seguenti:

- MQSO_CREATE non riesce perché l'handle di coda corrisponde alla coda A che ha già una sottoscrizione a X. Confronta questo comportamento con la chiamata riuscita. Tale chiamata ha esito positivo perché la coda B non dispone di una sottoscrizione a X ad essa collegata.
- MQSO_RESUME riesce perché l'handle della coda corrisponde alla coda A che ha già una sottoscrizione a X. Al contrario, la chiamata non riesce quando la sottoscrizione X non esiste sulla coda A.
- MQSO_ALTER si comporta in modo simile a MQSO_RESUME rispetto all'apertura della sottoscrizione e della coda. Tuttavia, se gli attributi contenuti nel descrittore della sottoscrizione inoltrato a MQSUB differiscono dagli attributi della sottoscrizione, MQSO_RESUME ha esito negativo, mentre MQSO_ALTER ha esito positivo finché l'istanza di programma dispone dell'autorizzazione per modificare gli attributi. Si noti che non è mai possibile modificare la stringa di argomenti in una sottoscrizione; ma, invece di restituire un errore, MQSUB ignora i valori del nome argomento e della stringa argomento nel descrittore della sottoscrizione e utilizza i valori nella sottoscrizione esistente.

Successivamente, esaminare Tabella 123 a pagina 828 dove MQSUB viene passato alla sottoscrizione X e l'handle di coda alla coda B. I risultati delle opzioni di sottoscrizione sono i seguenti:

- MQSO_CREATE riesce e crea la sottoscrizione X sulla coda B perché si tratta di una nuova sottoscrizione sulla coda B.
- MQSO_RESUME non viene eseguito. MQSUB cerca la sottoscrizione X sulla coda B e non la trova, ma invece di restituire *RC = 2428 - la sottoscrizione X non esiste*, restituisce *RC = 2019 - La coda di sottoscrizione non corrisponde all'handle dell'oggetto coda*. Il comportamento della terza opzione MQSO_ALTER suggerisce il motivo di questo errore imprevisto. MQSUB prevede che l'handle della coda punti a una coda con una sottoscrizione. Lo verifica prima di verificare se la sottoscrizione denominata in sd.SubName esiste.
- MQSO_ALTER riesce e sposta la sottoscrizione dalla coda A alla coda B.

Un caso non visualizzato nella tabella è se il nome della sottoscrizione sulla coda A non corrisponde al nome della sottoscrizione in `sd.SubName`. La chiamata ha esito negativo con un *RC = 2428 - la sottoscrizione X non esiste nella coda A*.

Tabella 123. Errori da MQSUB con diversi handle di coda e combinazioni di sottoscrizioni

Handle di coda	**Coda A** **Sottoscrizione X** **Coda B** **Nessuna sottoscrizione**	**Coda A** **Nessuna sottoscrizione** **Coda B** **Nessuna sottoscrizione**
Hobj per *Coda A* passato a MQSUB	**MQSO_CREATE** RC = 2432 - La sottoscrizione X esiste già nella coda A **MQSO_RESUME** Riprende la sottoscrizione X sulla coda A **MQSO_ALTER** Riprende la sottoscrizione X sulla Coda A e apporta le modifiche consentite	**MQSO_CREATE** Crea la sottoscrizione X sulla coda A **MQSO_RESUME** RC = 2428 - La sottoscrizione X non esiste nella coda A **MQSO_ALTER** RC = 2428 - La sottoscrizione X non esiste nella coda A
Hobj per *Coda B* passato a MQSUB	**MQSO_CREATE** Crea una nuova sottoscrizione X sulla coda B **MQSO_RESUME** RC = 2019 - La coda di sottoscrizione non corrisponde al gestore oggetti della coda **MQSO_ALTER** Spostare la sottoscrizione X dalla coda A alla coda B	**MQSO_CREATE** Crea una nuova sottoscrizione X sulla coda B **MQSO_RESUME** RC = 2428 - la sottoscrizione X non esiste sulla coda B **MQSO_ALTER** RC = 2428 - la sottoscrizione X non esiste sulla coda B
MQHO_NONE passato a MQSUB	**MQSO_CREATE** RC = 2019 - Handle oggetto non valido: impostare l'indicatore `MQSO_MANAGED` per creare una sottoscrizione gestita e creare una coda gestita **MQSO_RESUME** Riprende la sottoscrizione X sulla coda A e restituisce Hobj alla coda A **MQSO_ALTER** Riprende la sottoscrizione X sulla coda A, restituisce Hobj alla coda A e apporta le modifiche consentite	**MQSO_CREATE** RC = 2019 - Handle oggetto non valido: impostare l'indicatore `MQSO_MANAGED` per creare una sottoscrizione gestita e creare una coda gestita **MQSO_RESUME** RC = 2428 - Nessuna sottoscrizione X **MQSO_ALTER** RC = 2019 - Handle oggetto non valido: nessuna coda A o B

Nota: Lo stile di codifica compatto è progettato per la leggibilità e non per l'utilizzo in produzione.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

Figura 74. Sottoscrittore MQ non gestito - parte 1: dichiarazioni.

```
    switch(argc){                  /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

Figura 75. Utente MQ non gestito - parte 2: gestione parametri.

Ulteriori commenti sulla gestione dei parametri in questo esempio sono i seguenti:

switch((argv[5][0]))
È possibile immettere A lter | C reate | R esume nel parametro 5 per verificare l'effetto della sovrascrittura di parte dell'impostazione dell'opzione MQSUB utilizzata per impostazione predefinita nell'esempio. L'impostazione predefinita utilizzata dall'esempio è MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Nota: L'impostazione MQSO_ALTER o MQSO_RESUME senza l'impostazione MQSO_DURABLE è un errore e sd.SubName deve essere impostato e fare riferimento a una sottoscrizione che può essere ripresa o modificata.

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
Se la coda di sottoscrizione predefinita, STOCKTICKER viene sostituita da una stringa nulla, se MQSO_CREATE è impostato, l'esempio imposta l'indicatore MQSO_MANAGED e crea una coda

di sottoscrizione dinamica. Se `Alter or Resume` sono impostati nel quinto parametro, il comportamento dell'esempio dipenderà dal valore di `subscriptionName`.

`*subscriptionName = '\0';`
`sdOptions = sdOptions - MQSO_DURABLE;`

Se la sottoscrizione predefinita, `IBMSTOCKPRICESUB`, viene sostituita da una stringa nulla, l'esempio rimuove l'indicatore `MQSO_DURABLE` . Se si esegue l'esempio fornendo i valori predefiniti per gli altri parametri, viene creata una sottoscrizione temporanea aggiuntiva destinata a `STOCKTICKER` e riceve pubblicazioni duplicate. La prossima volta che si esegue l'esempio, senza alcun parametro, si riceve di nuovo una sola pubblicazione.

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                  &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
                gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
&CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes           */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
&CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

Figura 76. Sottoscrittore MQ non gestito - parte 3: corpo del codice.

Ulteriori commenti sul codice in questo esempio sono i seguenti:

if (strlen(subscriptionQueue))
Se non è presente alcun nome coda di sottoscrizione, l'esempio utilizza MQHO_NONE come valore di Hobj.

MQOPEN(...);
La coda di sottoscrizione viene aperta e l'handle di coda salvato in Hobj.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
La sottoscrizione viene aperta utilizzando il Hobj passato da MQOPEN (o MQHO_NONE se non è presente alcun nome coda di sottoscrizione). Una coda non gestita può essere ripresa senza aprirla esplicitamente con un MQOPEN.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
La sottoscrizione viene chiusa utilizzando l'handle di sottoscrizione. A seconda che la sottoscrizione sia durevole o meno, la sottoscrizione viene chiusa con un MQCO_KEEP_SUB o MQCO_REMOVE_SUBimplicito. È possibile chiudere una sottoscrizione durevole con MQCO_REMOVE_SUB, ma non è possibile chiudere una sottoscrizione non durevole con MQCO_KEEP_SUB. L'azione di MQCO_REMOVE_SUB consiste nel rimuovere la sottoscrizione che arresta l'invio di ulteriori pubblicazioni alla coda di sottoscrizione.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
Non viene eseguita alcuna azione speciale se la sottoscrizione non è gestita. Se la coda è gestita e la sottoscrizione è chiusa con un MQCO_REMOVE_SUBesplicito o implicito, tutte le pubblicazioni vengono eliminate dalla coda e la coda viene eliminata a questo punto.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
Assicurarsi che i messaggi ricevuti siano quelli per la nostra sottoscrizione.

I risultati dell'esempio illustrano alcuni aspetti della pubblicazione / sottoscrizione:

In Figura 77 a pagina 832 l'esempio inizia pubblicando 130 nell'argomento NYSE/IBM/PRICE .

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Figura 77. Pubblica 130 su NYSE/IBM/PRICE

In Figura 78 a pagina 832 l'esecuzione dell'esempio utilizzando i parametri predefiniti riceve la pubblicazione conservata 130. L'oggetto argomento fornito e la stringa argomento vengono ignorati, come mostrato in Figura 82 a pagina 834. L'oggetto argomento e la stringa argomento vengono sempre presi dall'oggetto sottoscrizione, quando ne viene fornito uno, e la stringa argomento è immutabile. Il funzionamento effettivo dell'esempio dipende dalla scelta o dalla combinazione di MQSO_CREATE, MQSO_RESUMEe MQSO_ALTER. In questo esempio MQSO_RESUME è l'opzione selezionata.

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

Figura 78. Ricevere la pubblicazione conservata

In (Figura 79 a pagina 833) non viene ricevuta alcuna pubblicazione, poiché la sottoscrizione durevole ha già ricevuto la pubblicazione conservata. In questo esempio, la sottoscrizione viene ripresa fornendo solo il nome della sottoscrizione senza il nome della coda. Se il nome della coda è stato fornito, la coda viene aperta per prima e l'handle viene passato a MQSUB.

Nota: L'errore 2038 proveniente da MQINQ è dovuto al MQOPEN implicito di STOCKTICKER da MQSUB che non include l'opzione MQOO_INQUIRE . Evitare il codice di ritorno 2038 da MQINQ aprendo esplicitamente la coda.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

Figura 79. Riprendi sottoscrizione

In Figura 80 a pagina 833, l'esempio crea una sottoscrizione non gestita durevole utilizzando STOCKTICKER come destinazione. Poiché si tratta di una nuova sottoscrizione, riceve la pubblicazione conservata.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

Figura 80. Ricevi pubblicazione conservata con nuova sottoscrizione non durevole non gestita

In Figura 81 a pagina 833, per dimostrare le sottoscrizioni sovrapposte, viene inviata un'altra pubblicazione, modificando la pubblicazione conservata. Successivamente, viene creata una nuova sottoscrizione non durevole e non gestita non fornendo un nome sottoscrizione. La pubblicazione conservata viene ricevuta due volte, una per la nuova sottoscrizione e una per la sottoscrizione IBMSTOCKPRICESUB durevole ancora attiva sulla coda STOCKTICKER . L'esempio è un'illustrazione che indica che la coda ha sottoscrizioni e non l'applicazione. Nonostante non faccia riferimento alla sottoscrizione IBMSTOCKPRICESUB in questa chiamata dell'applicazione, l'applicazione riceve la pubblicazione due volte: una dalla sottoscrizione durevole creata amministrativamente e una dalla sottoscrizione non durevole creata dall'applicazione stessa.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

Figura 81. Sovrapposizione delle sottoscrizioni

In Figura 82 a pagina 834 l'esempio dimostra che la fornitura di una nuova stringa di argomenti e di una sottoscrizione esistente non comporta una sottoscrizione modificata.

1. Nel primo caso, Resume riprende la sottoscrizione esistente, come potrebbe essere previsto, e ignora la stringa di argomenti modificata.
2. Nel secondo caso, Alter causa un errore, RC = 2510, Topic not alterable.
3. Nel terzo esempio, Create causa un errore RC = 2432, Sub already exists.

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

Figura 82. Impossibile modificare gli argomenti della sottoscrizione

Concetti correlati

“Esempio 1: consumer di pubblicazione MQ” a pagina 818
Il consumer di pubblicazione di MQ è un consumer di messaggi IBM MQ che non sottoscrive gli argomenti.

“Esempio 2: utente MQ gestito” a pagina 820
Il sottoscrittore MQ gestito è il modello preferito per la maggior parte delle applicazioni sottoscrittore. Una sottoscrizione gestita è quella in cui IBM MQ gestisce la sottoscrizione ed esegue la registrazione e l'annullamento della registrazione per conto dell'utente. L'esempio richiede *nessuna* definizione amministrativa di code, argomenti o sottoscrizioni.

“Scrittura delle applicazioni del publisher” a pagina 810
Inizia a scrivere applicazioni di pubblicazione studiando due esempi. Il primo è modellato il più strettamente possibile su un'applicazione point to point inserendo i messaggi in una coda e il secondo dimostra la creazione dinamica degli argomenti - un modello più comune per le applicazioni publisher.

Cicli di vita di pubblicazione / sottoscrizione

Considerare i cicli di vita di argomenti, sottoscrizioni, sottoscrittori, pubblicazioni, publisher e code nella progettazione di applicazioni di pubblicazione / sottoscrizione.

Il ciclo di vita di un oggetto, ad esempio una sottoscrizione, inizia con la sua creazione e termina con la sua eliminazione. Può anche includere altri stati e modifiche che attraversa, come la sospensione temporanea, la presenza di argomenti principali e secondari, la scadenza e l'eliminazione.

Tradizionalmente gli oggetti IBM MQ come le code vengono creati amministrativamente o dai programmi di gestione utilizzando PCF (Programmable Command Format). La pubblicazione / sottoscrizione è diversa nel fornire i verbi API MQSUB e MQCLOSE per creare ed eliminare le sottoscrizioni, avendo il concetto di sottoscrizioni gestite che non solo creano ed eliminano le code, ma anche ripuliscono i messaggi non utilizzati e hanno associazioni tra gli oggetti argomento creati amministrativamente e le stringhe argomento create in modo programmatico o amministrativo.

Questa ricchezza funzionale soddisfa un'ampia gamma di requisiti di pubblicazione / sottoscrizione e semplifica la progettazione di alcuni modelli comuni di applicazione di pubblicazione / sottoscrizione. Le sottoscrizioni gestite, ad esempio, semplificano sia la programmazione che la gestione di una sottoscrizione destinata a durare solo fino a quando il programma che l'ha creata. Le sottoscrizioni non gestite semplificano la programmazione quando vi è una connessione più flessibile tra la sottoscrizione e l'utilizzo delle pubblicazioni. Le sottoscrizioni create centralmente sono utili quando il modello è quello di instradare il traffico di pubblicazione verso i consumatori in base a un modello centralizzato di controllo, ad esempio l'invio di informazioni di volo a gate automatizzati, mentre le sottoscrizioni create programmaticamente possono essere utilizzate se il personale del gate è responsabile della sottoscrizione ai record dei passeggeri per quel volo, inserendo un numero di volo a un gate.

In questo ultimo esempio, una sottoscrizione durevole gestita potrebbe essere appropriata: gestita, perché le sottoscrizioni vengono create molto spesso e hanno un endpoint chiaro quando il gate si chiude e la sottoscrizione può essere rimossa in modo programmatico; durevole, per evitare di perdere un record di passeggeri a causa del programma del gate sottoscrittore che si disattiva per un motivo o per l'altro[8]. Per avviare la pubblicazione dei registri dei passeggeri al gate, una possibile progettazione potrebbe essere che l'applicazione del gate sottoscriva i registri dei passeggeri utilizzando il numero del

[8] L'editore deve inviare le registrazioni dei passeggeri come messaggi persistenti per evitare altri possibili guasti, naturalmente.

gate e pubblichi l'evento di apertura del gate utilizzando il numero del gate. L'editore risponde all'evento di apertura del gate pubblicando le registrazioni dei passeggeri - che potrebbero poi andare anche ad altre parti interessate, come la fatturazione, per registrare il volo in corso, e ai servizi clienti, per inviare notifiche di testo ai telefoni cellulari dei passeggeri del numero del gate.

La sottoscrizione gestita centralmente potrebbe utilizzare un modello non gestito durevole, instradando gli elenchi di passeggeri al gate utilizzando una coda predefinita per ogni gate.

I seguenti tre esempi di cicli di vita di pubblicazione / sottoscrizione illustrano il modo in cui i sottoscrittori non durevoli, non durevoli e non durevoli gestiti interagiscono con le sottoscrizioni, gli argomenti, le code, i publisher e il gestore code e il modo in cui le responsabilità possono essere suddivise tra i programmi di gestione e sottoscrittore.

Sottoscrittore non durevole gestito

Figura 83 a pagina 836 mostra un'applicazione che crea una sottoscrizione non durevole gestita, riceve due messaggi che vengono pubblicati nell'argomento identificato nella sottoscrizione e termina. Le interazioni etichettate in un carattere grigio corsivo con frecce tratteggiate sono implicite.

Ci sono alcuni punti da notare.

1. L'applicazione crea una sottoscrizione su un argomento che è già stato pubblicato due volte. Quando il sottoscrittore riceve la prima pubblicazione, riceve la *seconda* pubblicazione che è la pubblicazione attualmente conservata.
2. Il gestore code crea una coda di sottoscrizione temporanea e una sottoscrizione per l'argomento.
3. La sottoscrizione ha una scadenza. Quando la sottoscrizione scade, non vengono inviate ulteriori pubblicazioni sull'argomento a questa sottoscrizione, ma il sottoscrittore continua a ricevere i messaggi pubblicati prima della scadenza della sottoscrizione. La scadenza della pubblicazione non è influenzata dalla scadenza della sottoscrizione.
4. La quarta pubblicazione non viene collocata nella coda di sottoscrizione e di conseguenza l'ultima `MQGET` non restituisce una pubblicazione.
5. Sebbene il sottoscrittore chiuda la sottoscrizione, non chiude la connessione alla coda o al gestore code.
6. Il gestore code si ripulisce poco dopo la chiusura dell'applicazione. Poiché la sottoscrizione è gestita e non durevole, la coda di sottoscrizione viene eliminata.

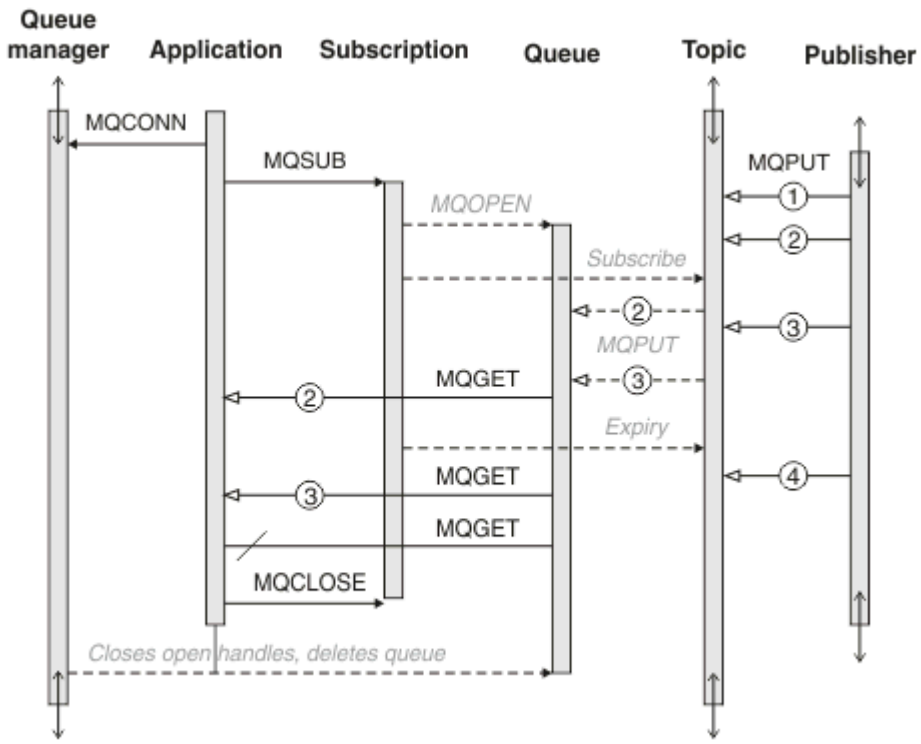


Figura 83. Cicli di vita del sottoscrittore non durevoli gestiti

Sottoscrittore durevole gestito

Il sottoscrittore (subscriber) duraturo gestito fa un passo avanti nell'esempio precedente e mostra una sottoscrizione gestita che sopravvive alla terminazione e al riavvio dell'applicazione di sottoscrizione.

Ci sono alcuni nuovi punti da notare.

1. In questo esempio, a differenza dell'ultimo, l'argomento della pubblicazione non esisteva prima che fosse definito nella sottoscrizione.
2. La prima volta che il sottoscrittore termina, chiude la sottoscrizione con l'opzione `MQCO_KEEP_SUB`. Questo è il comportamento predefinito per chiudere implicitamente una sottoscrizione durevole gestita.
3. Quando il sottoscrittore riprende la sottoscrizione, la coda di sottoscrizione viene riaperta.
4. La nuova pubblicazione 2, inserita nella coda prima della riapertura, è disponibile per `MQGET`, anche dopo la rimozione della sottoscrizione.

   Anche se la sottoscrizione è durevole, il sottoscrittore riceve in modo affidabile tutti i messaggi inviati dal publisher solo se *sia* la sottoscrizione è durevole e i messaggi persistenti. La persistenza del messaggio dipende dall'impostazione del campo `Persistent` nel `MQMD` del messaggio inviato dal publisher. Un sottoscrittore non ha alcun controllo su questo.
5. La chiusura della sottoscrizione con l'indicatore `MQCO_REMOVE_SUB` rimuove la sottoscrizione, arrestando eventuali ulteriori pubblicazioni inserite nella coda di sottoscrizione. Quando la coda di sottoscrizione viene chiusa, il gestore code rimuove la pubblicazione non letta 3ed elimina la coda. L'azione equivale all'eliminazione amministrativa della sottoscrizione.

   Nota: Non eliminare la coda manualmente o immettere `MQCLOSE` con l'opzione `MQCO_DELETE`o `MQCO_PURGE_DELETE`. I dettagli di implementazione visibili di una sottoscrizione gestita non fanno parte dell'interfaccia IBM MQ supportata. Il gestore code non è in grado di gestire una sottoscrizione in modo affidabile a meno che non disponga di un controllo completo.

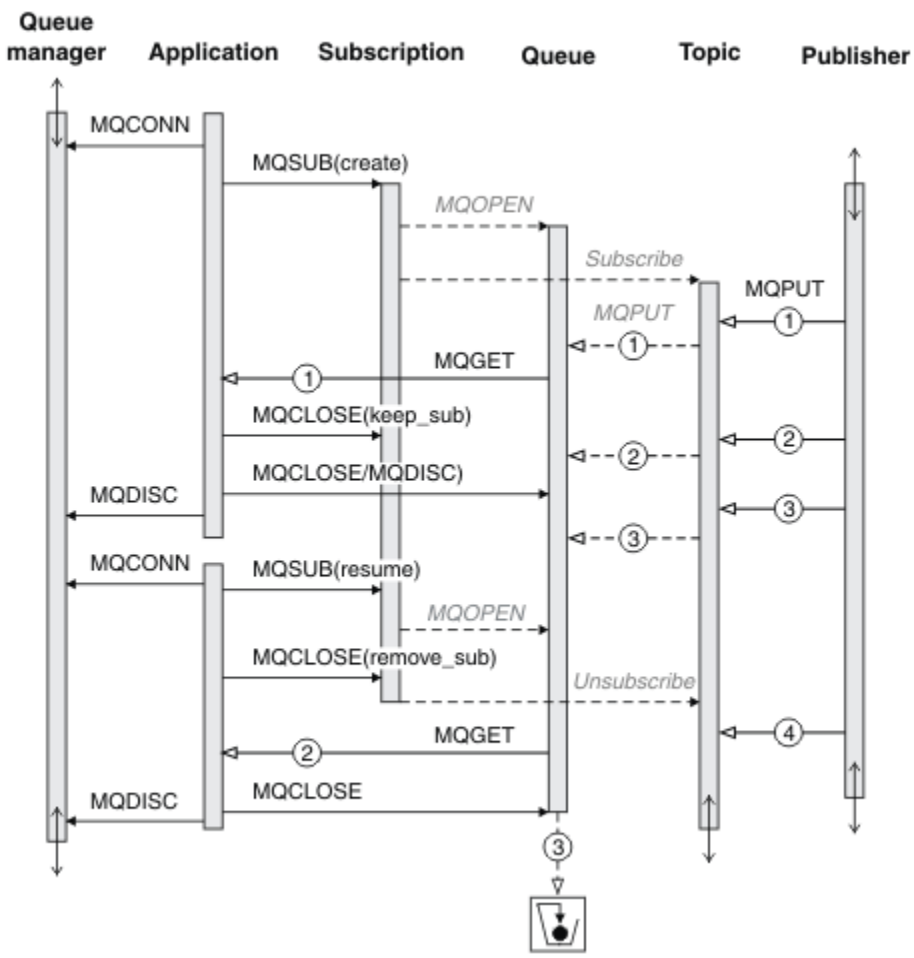


Figura 84. Cicli di vita del sottoscrittore durevoli gestiti

Sottoscrittore durevole non gestito

Un amministratore viene aggiunto nel terzo esempio: il sottoscrittore durevole non gestito. Questo è un buon esempio per mostrare come l'amministratore potrebbe interagire con un'applicazione di pubblicazione / sottoscrizione.

I punti da notare sono elencati.

1. Il publisher inserisce un messaggio, 1, in un argomento che in seguito viene associato all'oggetto argomento utilizzato per la sottoscrizione. L'oggetto argomento definisce una stringa di argomento che corrisponde all'argomento pubblicato utilizzando i caratteri jolly.
2. L'argomento ha una pubblicazione conservata.
3. L'amministratore crea un argomento, una coda e una sottoscrizione. L'oggetto argomento e la coda devono essere definiti prima della sottoscrizione.
4. L'applicazione apre la coda associata alla sottoscrizione e passa `MQSUB` l'handle della coda. In alternativa, potrebbe semplicemente aprire la sottoscrizione, passando l'handle della coda `MQHO_NONE`. L'inverso non è true, non può riprendere una sottoscrizione passando solo l'handle della coda senza un nome sottoscrizione - una coda potrebbe avere più sottoscrizioni.
5. L'applicazione apre la sottoscrizione utilizzando l'opzione `MQSO_RESUME` anche se è la prima volta che apre la sottoscrizione. Sta riprendendo una sottoscrizione creata in modo amministrativo.

6. Il sottoscrittore riceve la pubblicazione conservata, 1. La pubblicazione 2, sebbene sia stata pubblicata prima della ricezione di qualsiasi pubblicazione da parte del sottoscrittore, è stata pubblicata dopo l'inizio della sottoscrizione ed è la seconda pubblicazione sulla coda di sottoscrizione.

   Nota: Se la pubblicazione conservata non viene pubblicata come messaggio persistente, viene persa dopo il riavvio del gestore code.
7. In questo esempio la sottoscrizione è durevole. È possibile per un programma creare una sottoscrizione non durevole non gestita; dovrebbe essere ovvio che questo non è qualcosa che un amministratore può fare.
8. L'opzione MQCO_REMOVE_SUB alla chiusura della sottoscrizione ha l'effetto di rimuovere la sottoscrizione come se fosse stata eliminata dall'amministratore. Questa operazione arresta le ulteriori pubblicazioni inviate alla coda, ma non influisce sulle pubblicazioni già presenti nella coda, anche quando la coda è chiusa, a differenza di una sottoscrizione durevole *gestita* .
9. L'amministratore in seguito elimina il messaggio rimanente, 3, ed elimina la coda.

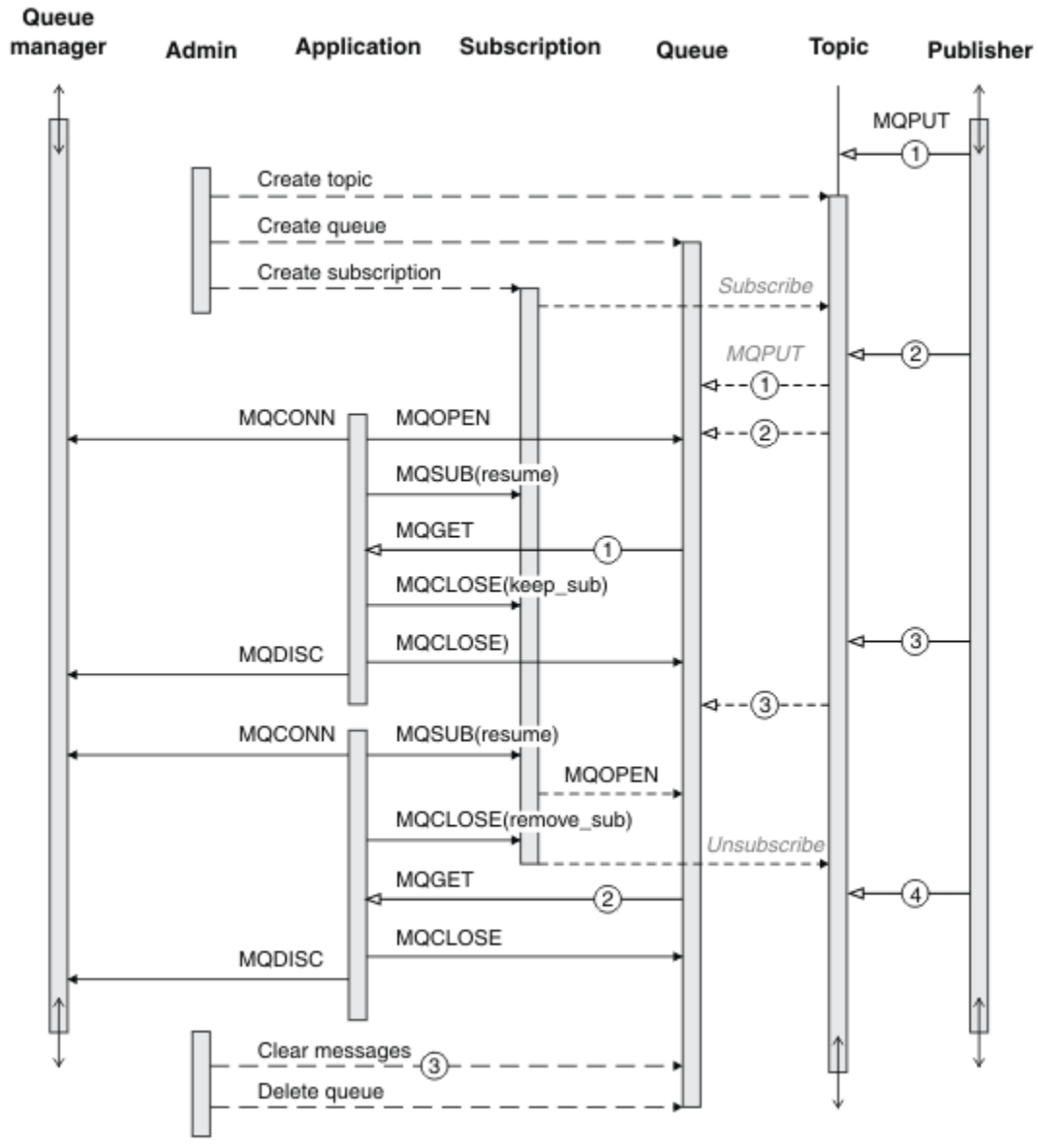


Figura 85. Cicli di vita del sottoscrittore durevoli non gestiti

Un modello normale per una sottoscrizione non gestita è per la gestione della coda e della sottoscrizione che deve essere eseguita dall'amministratore. In genere non si tenta di emulare il funzionamento di un sottoscrittore gestito e di riordinare le code e le sottoscrizioni in modo programmatico nel codice dell'applicazione. Se si ha bisogno di scrivere la logica di gestione, è necessario chiedersi se è possibile ottenere gli stessi risultati utilizzando un modello gestito. Non è facile scrivere codice di gestione strettamente sincronizzato e completamente affidabile. È più facile riordinare in un secondo momento, manualmente o utilizzando un programma di gestione automatizzato, quando è possibile essere sicuri che i messaggi, le sottoscrizioni e le code possano essere semplicemente eliminati, indipendentemente dal loro stato.

Proprietà dei messaggi di pubblicazione / sottoscrizione

Diverse proprietà del messaggio sono correlate alla messaggistica di pubblicazione / sottoscrizione IBM MQ .

Token PubAccounting

Questo è il valore che sarà nel campo AccountingToken di MQMD (Message Descriptor) di tutti i messaggi di pubblicazione corrispondenti a questa sottoscrizione. AccountingToken fa parte del contesto di identità del messaggio. Per ulteriori informazioni sul contesto del messaggio, consultare “Contesto messaggio” a pagina 48. Per ulteriori informazioni sul campo AccountingToken in MQMD, consultare AccountingToken.

PubApplIdentityData

Questo è il valore che si trova nel campo ApplIdentityData di MQMD (Message Descriptor) di tutti i messaggi di pubblicazione corrispondenti a questa sottoscrizione. ApplIdentityI dati fanno parte del contesto di identità del messaggio. Per ulteriori informazioni sul contesto del messaggio, consultare “Contesto messaggio” a pagina 48. Per ulteriori informazioni sul campo ApplIdentityData in MQMD, consultare ApplIdentityData.

Se l'opzione MQSO_SET_IDENTITY_CONTEXT non è specificata, i dati ApplIdentityche verranno impostati in ogni messaggio pubblicato per questa sottoscrizione sono vuoti, come informazioni di contesto predefinite.

Se viene specificata l'opzione MQSO_SET_IDENTITY_CONTEXT, l'IdentityData PubApplviene generato dall'utente e questo campo è un campo di input che contiene i dati ApplIdentityda impostare in ogni pubblicazione per questa sottoscrizione.

PubPriority

Questo è il valore che sarà nel campo Priorità di MQMD (Message Descriptor) di tutti i messaggi di pubblicazione che corrispondono a questa sottoscrizione. Per ulteriori informazioni sul campo Priorità in MQMD, consultare Priorità.

Il valore deve essere maggiore o uguale a zero; zero è la priorità più bassa. È inoltre possibile utilizzare i seguenti valori speciali:

- MQPRI_PRIORITY_AS_Q_DEF - Quando una coda di sottoscrizione viene fornita nel campo Hobj nella chiamata MQSUB e non è un handle gestito, la priorità per il messaggio viene presa dall'attributo DefProirity di questa coda. Se la coda così identificata è una coda cluster o esiste più di una definizione nel percorso di risoluzione del nome della coda, la priorità viene determinata quando il messaggio di pubblicazione viene inserito nella coda come descritto per Priorità in MQMD. Se la chiamata MQSUB utilizza un handle gestito, la priorità per il messaggio viene presa dall'attributo DefPriority della coda modello associata all'argomento sottoscritto.
- MQPRI_PRIORITY_AS_PUBLISHED - La priorità del messaggio è la priorità della pubblicazione originale. Questo è il valore iniziale di questo campo.

SubCorrelId

Attenzione: un identificatore di correlazione può essere passato solo tra i gestori code in un cluster di pubblicazione / sottoscrizione, non una gerarchia.

Tutte le pubblicazioni inviate per corrispondere a questa sottoscrizione conterranno questo identificativo di correlazione nel descrittore del messaggio. Se più sottoscrizioni utilizzano la stessa coda da cui ottenere le proprie pubblicazioni, l'uso di MQGET per ID correlazione consente di ottenere solo le pubblicazioni per una sottoscrizione specifica. Questo identificativo di correlazione può essere generato dal gestore code o dall'utente.

Se l'opzione MQSO_SET_CORREL_ID non viene specificata, l'identificatore di correlazione viene generato dal gestore code e questo campo è un campo di output che contiene l'identificatore di correlazione che verrà impostato in ogni messaggio pubblicato per questa sottoscrizione.

Se viene specificata l'opzione MQSO_SET_CORREL_ID, l'identificativo di correlazione viene generato dall'utente e questo campo è un campo di immissione che contiene l'identificativo di correlazione da impostare in ogni pubblicazione per questa sottoscrizione. In questo caso, se il campo contiene MQCI_NONE, l'identificativo di correlazione che verrà impostato in ogni messaggio pubblicato per questa sottoscrizione sarà l'identificativo di correlazione creato dall'inserimento originale del messaggio.

Se viene specificata l'opzione MQSO_GROUP_SUB e l'identificatore di correlazione specificato è lo stesso di una sottoscrizione raggruppata esistente che utilizza la stessa coda e una stringa di argomento sovrapposta, solo la sottoscrizione più significativa nel gruppo viene fornita con una copia della pubblicazione.

SubUserData

Questi sono i dati utente della sottoscrizione. I dati forniti nella sottoscrizione in questo campo verranno inclusi come proprietà del messaggio di dati MQSubUserdi ogni pubblicazione inviata a questa sottoscrizione.

Proprietà della pubblicazione

Tabella 124 a pagina 840 elenca le proprietà della pubblicazione fornite con un messaggio di pubblicazione.

È possibile accedere a queste proprietà direttamente dalla cartella **MQRFH2** o richiamarle utilizzando MQINQMP. MQINQMP accetta il nome della proprietà o il nome **MQRFH2** come nome della proprietà da analizzare.

Tabella 124. Proprietà della pubblicazione

Nome proprietà	Nome MQRFH2	Tipo	Descrizione
MQTopicString	mqps.Top	MQTYPE_STRING	Stringa argomento
MQSubUserData	mqps.Sud	MQTYPE_STRING	Dati utente sottoscrittore
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	Pubblicazione conservata
MQPubOptions	mqps.Pub	MQTYPE_INT32	Opzioni di pubblicazione
MQPubLevel	mqps.Pbl	MQTYPE_INT32	Livello di pubblicazione
MQPubTime	mqpse.Pts	MQTYPE_STRING	Ora di pubblicazione
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	Numero di sequenza pubblicazione
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	Dati stringa / intero aggiunti dal publisher

Tabella 124. Proprietà della pubblicazione (Continua)			
Nome proprietà	**Nome MQRFH2**	**Tipo**	**Descrizione**
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	Formato messaggio: MQRFH1 MQRFH2 PCF

Ordinamento dei messaggi

Per un particolare argomento, i messaggi vengono pubblicati dal gestore code nello stesso ordine in cui vengono ricevuti dalle applicazioni di pubblicazione (soggetto a riordinamento in base alla priorità del messaggio).

L'ordine dei messaggi normalmente indica che ogni sottoscrittore riceve messaggi da un determinato gestore code, su un determinato argomento, da uno specifico publisher nell'ordine in cui vengono pubblicati da tale publisher.

Tuttavia, come per tutti i messaggi IBM MQ , è possibile che i messaggi, occasionalmente, vengano consegnati in ordine non corretto. Ciò può verificarsi nelle situazioni seguenti:

- Se un collegamento nella rete si disattiva e i messaggi successivi vengono reinstradati lungo un altro collegamento
- Se una coda diventa temporaneamente piena o inibita, in modo che un messaggio venga inserito in una coda di messaggi non recapitabili e quindi ritardato, mentre i messaggi successivi passano direttamente attraverso di essa.
- Se l'amministratore elimina un gestore code quando i publisher e i sottoscrittori sono ancora operativi, i messaggi accodati vengono inseriti nella coda di messaggi non recapitabili e le sottoscrizioni vengono interrotte.

Se queste circostanze non possono verificarsi, le pubblicazioni vengono sempre consegnate in ordine.

Nota: Non è possibile utilizzare messaggi raggruppati o segmentati con Pubblicazione / Sottoscrizione.

Intercettazione delle pubblicazioni

È possibile intercettare una pubblicazione, modificarla e ripubblicarla prima che raggiunga qualsiasi altro sottoscrittore.

È possibile intercettare una pubblicazione prima che raggiunga un sottoscrittore per effettuare una delle seguenti operazioni:

- Allegare ulteriori informazioni al messaggio
- Blocca il messaggio
- Trasforma il messaggio

È possibile eseguire la stessa operazione su ciascun messaggio o modificare l'operazione in base alla sottoscrizione, al messaggio o all'intestazione del messaggio.

Riferimenti correlati

MQ_PUBLISH_EXIT - Uscita pubblicazione

Livelli di sottoscrizione

Impostare il livello di sottoscrizione di una sottoscrizione per intercettare una pubblicazione prima che raggiunga i relativi sottoscrittori finali. Un sottoscrittore intercettore effettua la sottoscrizione ad un livello di sottoscrizione superiore e ripubblica ad un livello di pubblicazione inferiore. Creare una catena di sottoscrittori di intercettazione per eseguire l'elaborazione dei messaggi su una pubblicazione prima che venga consegnata ai sottoscrittori finali.

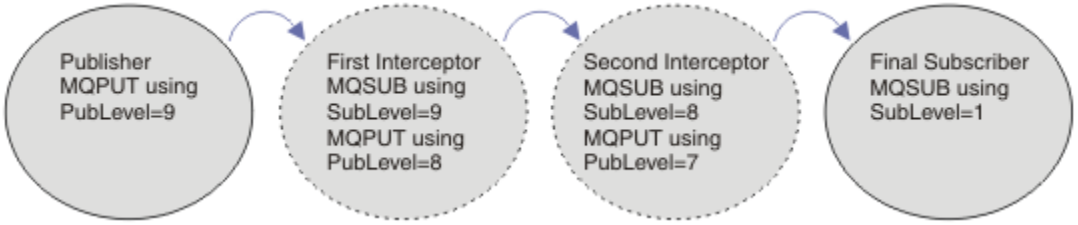


Figura 86. Sequenza di intercettazione dei sottoscrittori

Per intercettare una pubblicazione, utilizzare l'attributo **MQSD** `SubLevel` . Dopo che un messaggio è stato intercettato, è possibile trasformarlo e ripubblicarlo ad un livello di pubblicazione inferiore modificando l'attributo **MQPMO** `PubLevel` . Il messaggio viene quindi inviato ai sottoscrittori finali oppure viene intercettato nuovamente da un sottoscrittore intermedio a un livello di sottoscrizione inferiore.

Il sottoscrittore intercettore generalmente trasforma un messaggio prima di ripubblicarlo. Una sequenza di sottoscrittori di intercettazione forma un flusso di messaggi. In alternativa, è possibile non ripubblicare la pubblicazione intercettata: i sottoscrittori a livelli di sottoscrizione inferiori non riceveranno il messaggio.

Assicurarsi che l'intercettatore riceva le pubblicazioni prima di qualsiasi altro sottoscrittore. Impostare il livello di sottoscrizione dell'intercettatore su un valore superiore rispetto agli altri sottoscrittori. Per impostazione predefinita, i sottoscrittori hanno un `SubLevel` di 1. Il valore più alto è 9. Una pubblicazione deve cominciare con un `PubLevel` alto almeno quanto il `SubLevel`più alto. Pubblicare inizialmente con il `PubLevel` predefinito di 9.

- Se si dispone di un sottoscrittore intercettatore su un argomento, impostare `SubLevel` su 9.
- Per più applicazioni di intercettazione su un topic, impostare un `SubLevel` inferiore per ogni successivo sottoscrittore di intercettazione.
- È possibile implementare un massimo di 8 applicazioni di intercettazione, con livelli di sottoscrizione da 9 a 2 inclusi. Il destinatario finale del messaggio ha un `SubLevel` di 1.

L'intercettatore con il livello di sottoscrizione più alto uguale o inferiore al `PubLevel` della pubblicazione riceve prima la pubblicazione. Configurare solo un sottoscrittore intercettore per un argomento a un determinato livello di sottoscrizione. Se si dispone di più sottoscrittori a un determinato livello di sottoscrizione, vengono inviate più copie della pubblicazione alla serie finale di applicazioni di sottoscrizione.

Un sottoscrittore con un `SubLevel` `0` viene utilizzato come catchall. Riceve la pubblicazione se nessun sottoscrittore finale riceve il messaggio. Un sottoscrittore con `SubLevel` di `0` potrebbe essere utilizzato per monitorare le pubblicazioni che nessun altro sottoscrittore ha ricevuto.

Programmazione di un sottoscrittore intercettore

Utilizzare le opzioni di sottoscrizione descritte in Tabella 125 a pagina 842.

Tabella 125. Opzioni di sottoscrizione per intercettare i sottoscrittori

Opzione di sottoscrizione	**Note**
`MQSO_SET_CORREL_ID` e `SubCorrelId` impostato su `MQCI_NONE`	Mantenere il `CorrelId` della pubblicazione intercettata uguale alla pubblicazione originale. **Nota:** Non è possibile passare l'identificatore di correlazione di una pubblicazione in una gerarchia. Il campo viene utilizzato dal gestore code.
`PubPriority` impostato su `MQPRI_PRIORITY_AS_PUBLISHED`	Mantenere la priorità della pubblicazione intercettata uguale alla pubblicazione originale.

Le opzioni in Tabella 125 a pagina 842 devono essere utilizzate da tutti i sottoscrittori intercettatori. Il risultato è che l'identificatore di correlazione e la priorità del messaggio non vengono modificati dall'impostazione del publisher originale.

Quando il sottoscrittore intercettore ha elaborato la pubblicazione, ripubblica il messaggio nello stesso argomento in un PubLevel inferiore al SubLevel della propria sottoscrizione. Se il sottoscrittore di intercettazione imposta un SubLevel di 9, ripubblica il messaggio con un PubLevel di 8.

Per ripubblicare correttamente il messaggio, sono richieste diverse informazioni dalla pubblicazione originale. Riutilizzare lo stesso **MQMD** del messaggio originale e impostare MQPMO_PASS_ALL_CONTEXT per garantire che tutte le informazioni in **MQMD** vengano trasmesse al sottoscrittore successivo. Copiare i valori dalle proprietà del messaggio mostrate in Tabella 126 a pagina 843 nei corrispondenti campi del messaggio ripubblicato. Il sottoscrittore intercettore può modificare questi valori. Utilizzare l'operatore OR per aggiungere ulteriori valori a **MQPMO**. Campo Opzioni , per combinare le opzioni di inserimento del messaggio.

È necessario aprire esplicitamente la coda di pubblicazione piuttosto che utilizzare una coda di pubblicazione gestita. Non è possibile impostare MQSO_SET_CORREL_ID per una coda gestita. Non è inoltre possibile impostare MQOO_SAVE_ALL_CONTEXT su una coda gestita. Consultare i frammenti di codice elencati in “Esempi” a pagina 843.

Tabella 126. Valori MQPUT per i messaggi ripubblicati

Ripubblica messaggio utilizzando MQPUT	**Informazioni nel messaggio di pubblicazione**
MQOD. ObjectString	proprietà dei messaggi MQTopicString
MQPMO. Options	proprietà dei messaggi MQPubOptions

Il sottoscrittore finale ha la possibilità di impostare le proprie opzioni di sottoscrizione in modo diverso. Ad esempio, potrebbe impostare esplicitamente la priorità di pubblicazione piuttosto che MQPRI_PRIORITY_AS_PUBLISHED. Le impostazioni di un sottoscrittore finale influiscono solo sulla pubblicazione del sottoscrittore intercettatore finale nella catena.

Pubblicazioni conservate

Una pubblicazione conservata deve essere conservata dopo essere stata intercettata, copiando le opzioni put - message originali nel messaggio ripubblicato.

L'opzione MQPMO_RETAIN è impostata dall'autore (publisher). Ogni sottoscrittore intercettore deve trasferire il MQPubOptions alle opzioni put - message del messaggio ripubblicato come mostrato in Tabella 126 a pagina 843. La copia delle opzioni put - message conserva le opzioni impostate dal publisher originale, incluso se conservare la pubblicazione.

Quando una pubblicazione termina il suo passaggio nella catena di intercettazione dei sottoscrittori e viene consegnata ai sottoscrittori finali, viene infine conservata. I nuovi sottoscrittori, al livello secondario SubLevel 1, che richiedono la pubblicazione conservata, la ricevono senza ulteriori intercettazioni. I sottoscrittori a un SubLevel maggiore di 1 non ricevono la pubblicazione conservata. Di conseguenza, la pubblicazione conservata non viene modificata dalla catena di intercettazione dei sottoscrittori una seconda volta.

Esempi

Gli esempi sono frammenti di codice che possono essere combinati per creare un sottoscrittore intercettore. Il codice è scritto per essere breve, piuttosto che di qualità di produzione.

Le direttive del preprocessore in Figura 87 a pagina 844 definiscono le due proprietà da estrarre dai messaggi di pubblicazione richiesti dalla chiamata MQI MQINQMP .

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS        (MQPTR)(char*) "MQPubOptions",\
                                 0,\
                                 12,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
#define      MQTOPICSTRING       (MQPTR)(char*) "MQTopicString",\
                                 0,\
                                 13,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
```

Figura 87. Direttive del preprocessore

Figura 88 a pagina 844 elenca le dichiarazioni utilizzate nei frammenti di codice. Ad eccezione dei termini evidenziati, le dichiarazioni sono standard per un'applicazione IBM MQ .

Le opzioni Put e Get evidenziate vengono inizializzate per passare tutto il contesto. MQTOPICSTRING e MQPUBOPTIONS evidenziati sono inizializzatori MQCHARV per i nomi delle proprietà definiti nelle direttive del preprocessore. I nomi vengono passati a MQINQMP.

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

Figura 88. Dichiarazioni

Le inizializzazioni che non vengono eseguite facilmente nelle dichiarazioni vengono mostrate in Figura 89 a pagina 845. I valori evidenziati richiedono una spiegazione.

SYSTEM.NDURABLE.MODEL.QUEUE
: In questo esempio, invece di utilizzare MQSUB per aprire una sottoscrizione non duratura gestita, la coda modello, SYSTEM.NDURABLE.MODEL.QUEUE, è utilizzata per creare una coda dinamica temporanea. Il relativo handle viene passato a MQSUB. Aprendo direttamente la coda

è possibile salvare tutto il contesto del messaggio e impostare l'opzione di sottoscrizione, MQSO_SET_CORREL_ID.

MQGMO_CURRENT_VERSION
È importante utilizzare la versione corrente della maggior parte delle strutture IBM MQ . I campi come gmo.MsgHandle sono disponibili solo nella versione più recente delle strutture di controllo.

MQGMO_PROPERTIES_IN_HANDLE
La stringa di argomenti e le opzioni di inserimento dei messaggi impostati nella pubblicazione originale devono essere richiamati dal sottoscrittore intercettatore utilizzando le proprietà del messaggio. Un'alternativa potrebbe essere quella di leggere direttamente la struttura **MQRFH2** nel messaggio.

MQSO_SET_CORREL_ID
Utilizzare MQSO_SET_CORREL_ID in combinazione con,

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

L'effetto di queste opzioni è quello di trasmettere l'identificativo di correlazione. L'identificativo di correlazione impostato dal publisher originale viene inserito nel campo dell'identificativo di correlazione della pubblicazione ricevuto dal sottoscrittore intercettatore. Ogni sottoscrittore intercettore trasmette lo stesso identificativo di correlazione. Il sottoscrittore finale ha quindi la possibilità di ricevere lo stesso identificativo di correlazione.

Nota: Se la pubblicazione viene passata attraverso una gerarchia di pubblicazione / sottoscrizione, l'identificativo di correlazione non viene mai conservato.

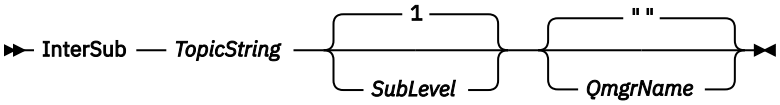
MQPRI_PRIORITY_AS_PUBLISHED
La pubblicazione viene inserita nella coda di pubblicazione con la stessa priorità del messaggio con cui è stata pubblicata.

```
strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version                = MQGMO_VERSION_4;
gmo.Options                = MQGMO_WAIT
                           | MQGMO_PROPERTIES_IN_HANDLE
                           | MQGMO_CONVERT;
gmo.WaitInterval           = 30000;
sd.Options                 = MQSO_CREATE
                           | MQSO_FAIL_IF_QUIESCING
                           | MQSO_SET_CORREL_ID;
sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version                 = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType             = MQOT_TOPIC;
putOD.ObjectString.VSPtr     = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
putOD.Version                = MQOD_VERSION_4;
pmo.Version                = MQPMO_VERSION_3;
```

Figura 89. Inizializzazioni

Figura 90 a pagina 846 mostra il frammento di codice per leggere i parametri della riga comandi, completare l'inizializzazione e creare la sottoscrizione intercettante.

Eseguire il programma con il comando,



Per rendere la gestione degli errori il più discreta possibile, il codice motivo di ogni chiamata MQI viene memorizzato in un elemento array differente. Dopo ogni chiamata, il codice di completamento viene verificato e, se il valore è MQCC_FAIL, il controllo esce dal blocco di codice do { } while(0) .

Le due linee di codice degne di nota sono:

pmo.PubLevel = sd.SubLevel - 1;
Imposta il livello di pubblicazione per il messaggio ripubblicato su un valore inferiore al livello di sottoscrizione del sottoscrittore intercettore.

gmo.MsgHandle = Hmsg;
Fornisce un handle del messaggio per MQGET per restituire le proprietà del messaggio.

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

Figura 90. Preparazione all'intercettazione delle pubblicazioni

Il frammento di codice principale, Figura 91 a pagina 847, richiama i messaggi dalla coda di pubblicazione. Interroga le proprietà del messaggio e ripubblica i messaggi utilizzando la stringa di argomenti e il **MQPMO**originale. Proprietà `option` della pubblicazione.

In questo esempio, non viene eseguita alcuna trasformazione sulla pubblicazione. La stringa di argomenti della pubblicazione ripubblicata corrisponde sempre alla stringa di argomenti sottoscritta dal sottoscrittore intercettatore. Se il sottoscrittore intercettatore è responsabile dell'intercettazione di più sottoscrizioni inviate alla stessa coda di pubblicazione, potrebbe essere necessario interrogare la stringa di argomenti per distinguere le pubblicazioni che corrispondono a sottoscrizioni differenti.

Vengono evidenziate le chiamate a `MQINQMP` . Le proprietà delle opzioni del messaggio di inserimento della pubblicazione e della stringa dell'argomento vengono scritte direttamente nelle strutture di controllo di output. L'unico motivo per modificare il campo di lunghezza MQCHARV di `putOD.ObjectString` da una lunghezza esplicita a una stringa con terminazione null è utilizzare `printf` per emettere la stringa.

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

Figura 91. Intercetta pubblicazione e ripubblica

Il frammento di codice finale viene mostrato in Figura 92 a pagina 847.

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

Figura 92. Completamento

Intercettazione di pubblicazioni e pubblicazione / sottoscrizione distribuita
Seguire un pattern semplice quando si distribuisce l'intercettazione di sottoscrittori o le uscite di pubblicazione in una topologia di pubblicazione / sottoscrizione distribuita. Distribuire i sottoscrittori di intercettazione sugli stessi gestori code dei publisher e le uscite di pubblicazione sugli stessi gestori code dei sottoscrittori finali.

La Figura 93 a pagina 848 mostra due gestori code connessi in un cluster di pubblicazione - sottoscrizione. Un publisher crea una pubblicazione per un argomento cluster a livello di pubblicazione 9. Le frecce numerate mostrano la sequenza di passi intrapresi dalla pubblicazione durante il flusso ai sottoscrittori dell'argomento cluster. La pubblicazione viene intercettata dal sottoscrittore con `Sublevel` 9 e ripubblicata con `Publevel` 8. Viene intercettata nuovamente da un sottoscrittore in `Livello secondario` 8. Il sottoscrittore ripubblica a `Publevel` 7. Il sottoscrittore proxy fornito dal gestore code inoltra la pubblicazione al gestore code B, dove è stata distribuita un'uscita di pubblicazione oltre a un sottoscrittore finale. La pubblicazione viene elaborata dall'uscita di pubblicazione prima che venga finalmente ricevuta dal sottoscrittore finale a `Sottolivello` 1. I sottoscrittori intercettatori e l'uscita di pubblicazione vengono visualizzati con contorni interrotti.

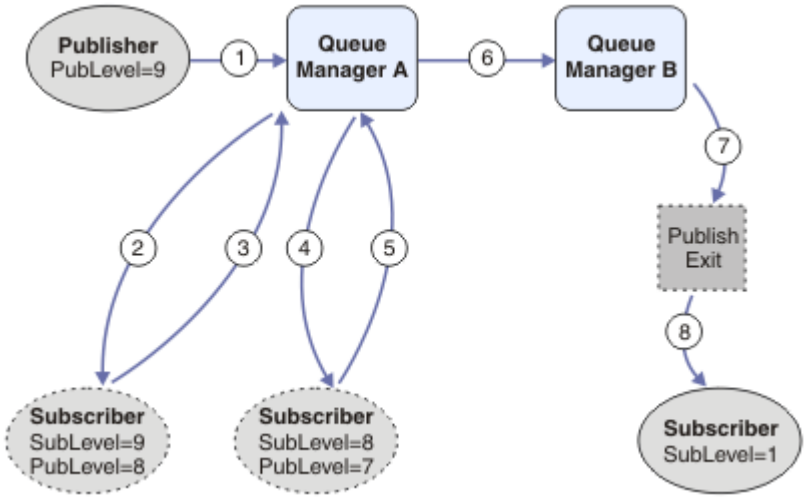


Figura 93. Uscita di pubblicazione e intercettazione in un cluster

L'obiettivo del modello semplice è che ogni sottoscrittore che riceve una pubblicazione riceva la pubblicazione identica. La pubblicazione passa attraverso la stessa sequenza di trasformazioni indipendentemente da dove è connesso il sottoscrittore. Probabilmente si desidera evitare che la sequenza di trasformazioni vari, a seconda di dove sono connessi i publisher o i sottoscrittori finali. Un'eccezione ragionevole sarebbe quella di adattare la pubblicazione finalmente consegnata a ogni singolo sottoscrittore. Utilizzare l'uscita di Pubblicazione per personalizzare la pubblicazione in base alla coda a cui la pubblicazione viene consegnata.

È necessario considerare attentamente dove distribuire le intercettazioni dei sottoscrittori e le uscite di pubblicazione in una topologia di pubblicazione / sottoscrizione distribuita. Il modello semplice distribuisce le intercettazioni dei sottoscrittori allo stesso gestore code dei publisher e le uscite di pubblicazione agli stessi gestori code dei sottoscrittori finali.

Anti - modello

Figura 94 a pagina 849 mostra come le cose possono andare storto, se non si segue un modello semplice. Per complicare la distribuzione, viene aggiunto un sottoscrittore finale al gestore code A e due ulteriori sottoscrittori di intercettazione vengono aggiunti al gestore code B.

La pubblicazione viene inoltrata al gestore code B in `PubLevel` 7, dove viene intercettata da un sottoscrittore in `SubLevel` 5 prima di essere utilizzata dal sottoscrittore finale in `SubLevel` 1. L'uscita di pubblicazione intercetta la pubblicazione prima che venga passata sia al consumer intercettore che a quello finale sul gestore code B. La pubblicazione raggiunge il sottoscrittore finale sul gestore code A senza essere elaborata dall'uscita di pubblicazione.

In una topologia di pubblicazione / sottoscrizione, i sottoscrittori proxy si sottoscrivono a `SubLevel` 1e passano il `PubLevel` impostato dall'ultimo sottoscrittore intercettore. In Figura 94 a pagina 849, il risultato è che la pubblicazione non viene intercettata dal sottoscrittore utilizzando `SubLevel` 9 sul gestore code B.

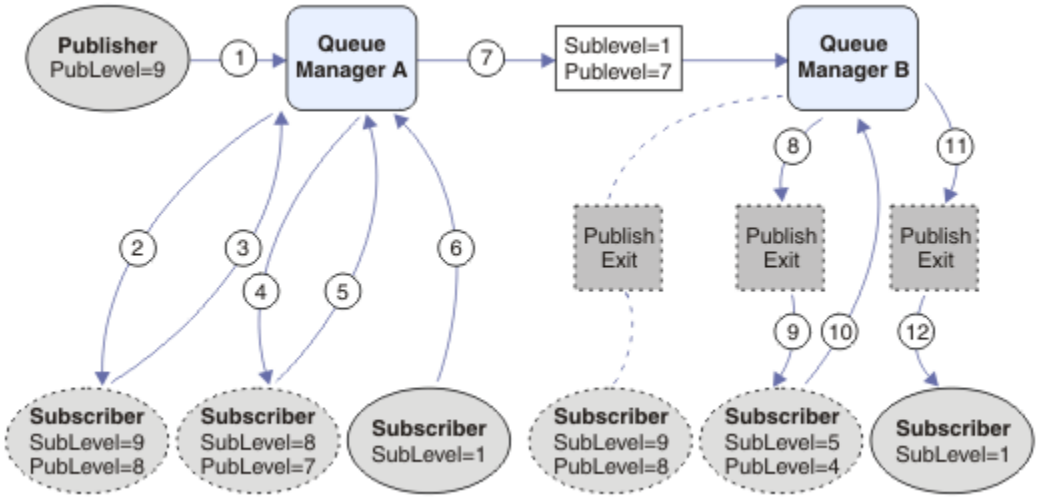


Figura 94. Distribuzione complessa dei sottoscrittori di intercettazioni

Opzioni di pubblicazione

Sono disponibili diverse opzioni che controllano la modalità di pubblicazione dei messaggi.

Rifiuto delle informazioni di risposta ai sottoscrittori

Se non si desidera che i sottoscrittori siano in grado di rispondere alle pubblicazioni che ricevono, è possibile nascondere le informazioni nei campi Qmgr ReplyToe ReplyTodi MQMD utilizzando l'opzione put - message MQPMO_SUPPRESS_REPLYTO. Se viene utilizzata questa opzione, il gestore code rimuove tali informazioni da MQMD quando riceve la pubblicazione prima di inoltrarla a qualsiasi sottoscrittore (subscriber).

Questa opzione non può essere utilizzata in combinazione con un'opzione di report che richiede una coda ReplyTo, se questa viene tentata con esito negativo con MQRC_MISSING_REPLY_TO_Q.

Livello di pubblicazione

L'utilizzo dei livelli di pubblicazione consente di controllare quali sottoscrittori ricevono la pubblicazione. Il livello di pubblicazione indica il livello di sottoscrizione di destinazione della pubblicazione. Solo le sottoscrizioni con il livello di sottoscrizione più alto inferiore o uguale al livello di pubblicazione della pubblicazione riceveranno la pubblicazione. Questo valore deve essere compreso tra zero e nove; zero è il livello di pubblicazione più basso. Il valore iniziale di questo campo è 9. Uno degli utilizzi dei livelli di pubblicazione e sottoscrizione è per intercettare le pubblicazioni.

Verifica se una pubblicazione non viene consegnata ad alcun sottoscrittore

Per controllare se una pubblicazione non è stata consegnata ad alcun sottoscrittore (subscriber), utilizzare l'opzione put - message MQPMO_WARN_IF_NO_SUBS_MATCHED con la chiamata MQPUT. Se l'operazione di inserimento restituisce un codice di completamento MQCC_WARNING e un codice motivo MQRC_NO_SUBS_MATCHED, la pubblicazione non è stata consegnata ad alcuna sottoscrizione. Se l'opzione MQPMO_RETAIN viene specificata sull'operazione di inserimento, il messaggio viene conservato e consegnato a qualsiasi sottoscrizione corrispondente definita successivamente. In un sistema di pubblicazione / sottoscrizione distribuito, il codice motivo MQRC_NO_SUBS_MATCHED viene restituito solo se non vi sono sottoscrizioni proxy registrate per l'argomento sul gestore code.

Opzioni di sottoscrizione

Sono disponibili diverse opzioni che controllano il modo in cui vengono gestite le sottoscrizioni dei messaggi.

Persistenza messaggio

I gestori code mantengono la persistenza delle pubblicazioni che inoltrano ai sottoscrittori come impostato dal publisher. Il publisher imposta la persistenza in modo che sia una delle seguenti opzioni:

0
Non permanente

1
permanente

2
Persistenza come definizione di coda / argomento

Per la pubblicazione / sottoscrizione, il publisher risolve l'oggetto argomento e **topicString** in un oggetto argomento risolto. Se il publisher specifica Persistenza come definizione coda / argomento, la persistenza predefinita dall'oggetto argomento risolto viene impostata per la pubblicazione.

Pubblicazioni conservate

Per controllare quando vengono ricevute le pubblicazioni conservate, i sottoscrittori possono utilizzare due opzioni di sottoscrizione:

Pubblica solo su richiesta, MQSO_PUBLICATIONS_ON_REQUEST

Se si desidera che un sottoscrittore abbia il controllo di quando riceve le pubblicazioni, è possibile utilizzare l'opzione di sottoscrizione MQSO_PUBLICATIONS_ON_REQUEST. Un sottoscrittore può quindi controllare quando riceve le pubblicazioni utilizzando la chiamata MQSUBRQ (specificando l'handle Hsub restituito dalla chiamata MQSUB originale) per richiedere che venga inviata la pubblicazione conservata di un argomento. I sottoscrittori che utilizzano l'opzione di sottoscrizione MQSO_PUBLICATIONS_ON_REQUEST non ricevono alcuna pubblicazione non conservata.

Se si specifica MQSO_PUBLICATIONS_ON_REQUEST è necessario utilizzare MQSUBRQ per richiamare qualsiasi pubblicazione. Se non si utilizza MQSO_PUBLICATIONS_ON_REQUEST, vengono visualizzati i messaggi man mano che vengono pubblicati.

Se un sottoscrittore (subscriber) utilizza la chiamata MQSUBRQ e utilizza i caratteri jolly nell'argomento della sottoscrizione, la sottoscrizione potrebbe corrispondere a più argomenti o nodi su una struttura ad albero degli argomenti, tutti i quali con messaggi conservati (se presenti) verranno inviati al sottoscrittore.

Questa opzione può essere particolarmente utile quando viene utilizzata con sottoscrizioni durevoli perché un gestore code continuerà a inviare pubblicazioni a un sottoscrittore se ha sottoscritto in modo duraturo anche se l'applicazione del sottoscrittore non è in esecuzione. Ciò potrebbe causare un accumulo di messaggi sulla coda del sottoscrittore. Questa creazione può essere evitata se il sottoscrittore (subscriber) si registra utilizzando l'opzione MQSO_PUBLICATIONS_ON_REQUEST. In alternativa, è possibile utilizzare sottoscrizioni non durevoli, se appropriato per la propria applicazione, per evitare un accumulo di messaggi indesiderati.

Se una sottoscrizione è durevole e un publisher utilizza pubblicazioni conservate, l'applicazione del sottoscrittore può utilizzare la chiamata MQSUBRQ per aggiornare le informazioni sullo stato dopo un riavvio. Il sottoscrittore deve quindi aggiornare periodicamente il proprio stato utilizzando la chiamata MQSUBRQ.

Nessuna pubblicazione verrà inviata come risultato della chiamata MQSUB che utilizza questa opzione. Una sottoscrizione durevole che è stata ripresa dopo la disconnessione utilizzerà l'opzione MQSO_PUBLICATIONS_ON_REQUEST se la sottoscrizione originale è stata configurata per utilizzare questa opzione.

Solo nuove pubblicazioni, MQSO_NEW_PUBLICATIONS_ONLY

Se una pubblicazione conservata esiste su un argomento, tutti i sottoscrittori che effettuano una sottoscrizione dopo che la pubblicazione è stata effettuata riceveranno una copia di tale pubblicazione. Se un sottoscrittore (subscriber) non desidera ricevere le

pubblicazioni effettuate prima della sottoscrizione, può utilizzare l'opzione di sottoscrizione MQSO_NEW_PUBLICATIONS_ONLY.

Raggruppamento di sottoscrizioni

Considerare il raggruppamento delle sottoscrizioni se è stata impostata una coda per ricevere le pubblicazioni e si dispone di un numero di sottoscrizioni sovrapposte che alimentano le pubblicazioni nella stessa coda. Questa situazione è simile all'esempio in Sottoscrizioni sovrapposte.

È possibile evitare di ricevere pubblicazioni duplicate impostando l'opzione MQSO_GROUP_SUB quando si sottoscrive un argomento. Il risultato è che quando più di una sottoscrizione nel gruppo corrisponde all'argomento di una pubblicazione, solo una sottoscrizione è responsabile dell'inserimento della pubblicazione nella coda. Le altre sottoscrizioni corrispondenti all'argomento della pubblicazione vengono ignorate.

La sottoscrizione responsabile dell'inserimento della pubblicazione nella coda viene scelta in base al fatto che ha la stringa di argomento corrispondente più lunga, prima di incontrare qualsiasi carattere jolly. Può essere pensato come l'abbonamento corrispondente più vicino. Le sue proprietà vengono propagate alla pubblicazione, incluso se dispone della proprietà MQSO_NOT_OWN_PUBS . In caso contrario, non viene consegnata alcuna pubblicazione alla coda, anche se altre sottoscrizioni corrispondenti potrebbero non avere la proprietà MQSO_NOT_OWN_PUBS .

Non è possibile inserire tutte le sottoscrizioni in un unico gruppo per eliminare le pubblicazioni duplicate. Le sottoscrizioni raggruppate devono soddisfare queste condizioni:

1. Nessuna delle sottoscrizioni è gestita.
2. Un gruppo di sottoscrizioni consegna pubblicazioni alla stessa coda.
3. Ogni sottoscrizione deve essere allo stesso livello di sottoscrizione.
4. Il messaggio di pubblicazione per ogni sottoscrizione nel gruppo ha lo stesso identificativo di correlazione.

   Per assicurarsi che ogni sottoscrizione risulti in un messaggio di pubblicazione con lo stesso identificativo di correlazione, impostare MQSO_SET_CORREL_ID per creare il proprio identificativo di correlazione nella pubblicazione e impostare lo stesso valore nel campo **SubCorrelId** in ciascuna sottoscrizione. Non impostare **SubCorrelId** sul valore MQCI_NONE.

Per ulteriori informazioni, consultare ../refdev/q100080_.dita#q100080_/mqso_group_sub .

Richiesta di informazioni e impostazione degli attributi dell'oggetto

Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

Influiscono sul modo in cui un gestore code elabora un oggetto. Gli attributi di ciascun tipo di oggetto IBM MQ sono descritti in dettaglio in Attributi di oggetti.

Alcuni attributi sono impostati quando l'oggetto è definito e possono essere modificati solo utilizzando i comandi IBM MQ ; un esempio di tale attributo è la priorità predefinita per i messaggi inseriti in una coda. Altri attributi sono influenzati dal funzionamento del gestore code e possono cambiare nel tempo; un esempio è la profondità corrente di una coda.

È possibile richiedere informazioni sui valori correnti della maggior parte degli attributi utilizzando la chiamata MQINQ. La MQI fornisce anche una chiamata MQSET con cui è possibile modificare alcuni attributi della coda. Non è possibile utilizzare le chiamate MQI per modificare gli attributi di qualsiasi altro tipo di oggetto. È invece necessario utilizzare una delle seguenti risorse:

- ALW La funzionalità di MQSC, descritta in Comandi MQSC.
- IBM i I comandi CL CHGMQMx, descritti in Riferimento ai comandi CL per IBM io la funzione MQSC.
- z/OS I comandi dell'operatore ALTER o i comandi DEFINE con l'opzione REPLACE, descritti in Comandi MQSC.

Nota: I nomi degli attributi degli oggetti vengono mostrati in questa documentazione nel formato in cui vengono utilizzati con le chiamate MQINQ e MQSET. Quando si utilizzano comandi IBM MQ per definire, modificare o visualizzare gli attributi, è necessario identificare gli attributi utilizzando le parole chiave mostrate nelle descrizioni dei comandi nei collegamenti degli argomenti.

Sia le chiamate MQINQ che MQSET utilizzano array di selettori per identificare gli attributi che si desidera interrogare o impostare. Esiste un selettore per ogni attributo che è possibile utilizzare. Il nome del selettore ha un prefisso, determinato dalla natura dell'attributo:

Tabella 127. Prefissi per i nomi dei selettori

Prefisso	**Descrizione**
MQCA_	Questi selettori fanno riferimento ad attributi che contengono dati di caratteri (ad esempio, il nome di una coda).
MQIA_	Questi selettori fanno riferimento ad attributi che contengono valori numerici (ad esempio, *CurrentQueueDepth*, il numero di messaggi su una coda) o un valore costante (ad esempio, *SyncPoint*, se il gestore code supporta i punti di sincronizzazione).

Prima di utilizzare le chiamate MQINQ o MQSET, l'applicazione deve essere connessa al gestore code ed è necessario utilizzare la chiamata MQOPEN per aprire l'oggetto per l'impostazione o la richiesta di attributi. Queste operazioni sono descritte in “Connessione e disconnessione da un gestore code” a pagina 735 e “Apertura e chiusura di oggetti” a pagina 742.

Utilizzare i seguenti collegamenti per ulteriori informazioni sull'acquisizione delle informazioni e sull'impostazione degli attributi degli oggetti:

- “Richiesta di informazioni sugli attributi di un oggetto” a pagina 853
- “Alcuni casi in cui la chiamata MQINQ ha esito negativo” a pagina 853
- “Impostazione degli attributi della coda” a pagina 854

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890

Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

"Applicazioni bridge IMS e IMS su IBM MQ for z/OS" a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Richiesta di informazioni sugli attributi di un oggetto

Utilizzare la chiamata MQINQ per richiedere informazioni sugli attributi di qualsiasi tipo di IBM MQ.

Come input per questa chiamata, è necessario fornire:

- Un handle di connessione.
- Un handle di oggetto.
- Il numero di selettori.
- Un array di selettori di attributi, ogni selettore con formato MQCA_ * o MQIA_ *. Ogni selettore rappresenta un attributo con un valore che si desidera analizzare e ciascun selettore deve essere valido per il tipo di oggetto rappresentato dall'handle dell'oggetto. È possibile specificare i selettori in qualsiasi ordine.
- Il numero di attributi interi che si stanno interrogando. Specificare zero se non si sta analizzando gli attributi integer.
- La lunghezza del buffer degli attributi carattere in *CharAttrLength*. Deve essere almeno la somma delle lunghezze richieste per contenere ciascuna stringa attributo carattere. Specificare zero se non si stanno analizzando gli attributi dei caratteri.

L'output di MQINQ è:

- Una serie di valori di attributo interi copiati nell'array. Il numero di valori è determinato da *IntAttrCount*. Se *IntAttrCount* o *SelectorCount* è zero, questo parametro non viene utilizzato.
- Il buffer in cui vengono restituiti gli attributi carattere. La lunghezza del buffer viene fornita dal parametro **CharAttrLength** . Se *CharAttrLength* o *SelectorCount* è zero, questo parametro non viene utilizzato.
- Un codice di completamento. Se il codice di completamento fornisce un'avvertenza, ciò significa che la chiamata è stata completata solo parzialmente. In questo caso, esaminare il codice di errore.
- Un codice di errore. Esistono tre situazioni di completamento parziale:
  - Il selettore non si applica al tipo di coda
  - Non è consentito spazio sufficiente per gli attributi integer
  - Non è consentito spazio sufficiente per gli attributi carattere

  Se si verifica più di una di queste situazioni, viene restituita la prima che si applica.

Se si apre una coda per l'emissione o l'interrogazione e questa si risolve in una coda cluster non locale, è possibile interrogare solo il nome della coda, il tipo di coda e gli attributi comuni. I valori degli attributi comuni sono quelli della coda scelta se è stato utilizzato MQOO_BIND_ON_OPEN. I valori sono quelli di una delle possibili code cluster arbitrarie se è stato utilizzato MQOO_BIND_NOT_FIXED o MQOO_BIND_ON_GROUP o MQOO_BIND_AS_Q_DEF e l'attributo della coda **DefBind** era MQBND_BIND_NOT_FIXED. Per ulteriori informazioni, consultare "MQOPEN e cluster" a pagina 887 e MQOPEN .

Nota: I valori restituiti dalla chiamata sono un'istantanea degli attributi selezionati. Gli attributi possono essere modificati prima che il programma agisca sui valori restituiti.

Esiste una descrizione della chiamata MQINQ in MQINQ.

Alcuni casi in cui la chiamata MQINQ ha esito negativo

Se si apre un alias per informarsi sui relativi attributi, vengono restituiti gli attributi della coda alias (l'oggetto IBM MQ utilizzato per accedere ad un'altra coda), non quelli della coda di base.

Tuttavia, la definizione della coda di base in cui si risolve l'alias viene aperta anche dal gestore code e se un altro programma modifica l'utilizzo della coda di base nell'intervallo tra le chiamate MQOPEN e

MQINQ, la chiamata MQINQ ha esito negativo e restituisce il codice motivo MQRC_OBJECT_CHANGED. La chiamata ha esito negativo anche se vengono modificati gli attributi dell'oggetto coda alias.

Allo stesso modo, quando si apre una coda remota per interrogarne gli attributi, vengono restituiti solo gli attributi della definizione locale della coda remota.

Se si specificano uno o più selettori non validi per il tipo di attributi della coda su cui si sta analizzando, la chiamata MQINQ viene completata con un'avvertenza e imposta l'output come segue:

- Per gli attributi integer, gli elementi corrispondenti di *IntAttrs* sono impostati su MQIAV_NOT_APPLICABLE.
- Per gli attributi carattere, le parti corrispondenti della stringa *CharAttrs* sono impostate su asterischi.

Se si specificano uno o più selettori che non sono validi per il tipo di attributi oggetto su cui si sta analizzando, la chiamata MQINQ non riesce e restituisce il codice motivo MQRC_SELECTOR_ERROR.

Non è possibile richiamare MQINQ per esaminare una coda modello; utilizzare la funzionalità MQSC o i comandi disponibili sulla propria piattaforma.

Impostazione degli attributi della coda

Utilizzare queste informazioni per informazioni su come impostare gli attributi della coda utilizzando la chiamata MQSET.

È possibile impostare solo i seguenti attributi della coda utilizzando la chiamata MQSET:

- *InhibitGet* (ma non per le code remote)
- *DistList* (non su z/OS)
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

La chiamata MQSET ha gli stessi parametri della chiamata MQINQ. Tuttavia, per MQSET, tutti i parametri tranne il codice di completamento e il codice motivo sono parametri di immissione. Non esistono situazioni di completamento parziale.

Nota: Non è possibile utilizzare MQI per impostare gli attributi degli oggetti IBM MQ diversi dalle code definite localmente.

Per ulteriori dettagli sulla chiamata MQSET, consultare MQSET.

Commit e backout delle unità di lavoro

Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

I seguenti termini vengono utilizzati in questo argomento:

- Commit
- Backout
- Coordinamento del punto di sincronizzazione
- Punto di sincronizzazione
- Unità di lavoro
- commit a singola fase
- commit a due fasi

Se si ha familiarità con questi termini di elaborazione della transazione, è possibile passare a “Considerazioni sul punto di sincronizzazione nelle applicazioni IBM MQ” a pagina 856.

Commit e backout

Quando un programma inserisce un messaggio in una coda all'interno di un'unità di lavoro, quel messaggio viene reso visibile ad altri programmi solo quando il programma esegue il commit dell'unità di lavoro. Per eseguire il commit di un'unità di lavoro, tutti gli aggiornamenti devono essere eseguiti correttamente per preservare l'integrità dei dati. Se il programma rileva un errore e decide che l'operazione di inserimento non è permanente, può eseguire il backout dell'unità di lavoro. Quando un programma esegue un backout, IBM MQ ripristina la coda rimuovendo i messaggi inseriti nella coda da tale unità di lavoro. Il modo in cui il programma esegue le operazioni di commit e di backout dipende dall'ambiente in cui il programma è in esecuzione.

Allo stesso modo, quando un programma richiama un messaggio da una coda all'interno di un'unità di lavoro, quel messaggio rimane nella coda fino a quando il programma non esegue il commit dell'unità di lavoro, ma il messaggio non è disponibile per essere richiamato da altri programmi. Il messaggio viene cancellato definitivamente dalla coda quando il programma esegue il commit dell'unità di lavoro. Se il programma esegue il backout dell'unità di lavoro, IBM MQ ripristina la coda rendendo i messaggi disponibili per essere richiamati da altri programmi.

Coordinamento del punto di sincronizzazione, punto di sincronizzazione, unità di lavoro

Il *coordinamento del punto di sincronizzazione* è il processo mediante il quale le unità di lavoro vengono sottoposte a commit o a backout con l'integrità dei dati.

La decisione di eseguire il commit o il backout delle modifiche viene presa, nel caso più semplice, al termine di una transazione. Tuttavia, può essere più utile per un'applicazione sincronizzare le modifiche dei dati in altri punti logici all'interno di una transazione. Questi punti logici sono chiamati *punti di sincronizzazione* (o *punti di sincronizzazione*) e il periodo di elaborazione di una serie di aggiornamenti tra due syncpoint è denominato *unità di lavoro*. Diverse chiamate MQGET e MQPUT possono essere parte di una singola unità di lavoro.

Il numero massimo di messaggi all'interno di un'unità di lavoro può essere controllato dall'attributo MAXUMSGS del comando ALTER QMGR .

commit a singola fase

Un processo *commit a fase singola* è un processo in cui un programma può eseguire il commit degli aggiornamenti su una coda senza coordinare le relative modifiche con altri gestori risorse.

commit a due fasi

Un processo di *commit a due fasi* è un processo in cui gli aggiornamenti effettuati da un programma alle code IBM MQ possono essere coordinati con aggiornamenti ad altre risorse (ad esempio, database sotto il controllo di Db2). In tale processo, viene eseguito il commit o il backout degli aggiornamenti a tutte le risorse.

Per facilitare la gestione delle unità di lavoro, IBM MQ fornisce l'attributo **BackoutCount** . Viene incrementato ogni volta che viene eseguito il backout di un messaggio all'interno di un'unità di lavoro. Se il messaggio causa ripetutamente la fine anomala dell'unità di lavoro, il valore di *BackoutCount* alla fine supera quello di *BackoutThreshold*. Questo valore viene impostato quando la coda è definita. In questa situazione, l'applicazione può rimuovere il messaggio dall'unità di lavoro e inserirlo in un'altra coda, come definito in *BackoutRequeueQName*. Quando il messaggio viene spostato, l'unità di lavoro può eseguire il commit.

Utilizzare i seguenti collegamenti per ulteriori informazioni sul commit e il backout delle unità di lavoro:

- “Considerazioni sul punto di sincronizzazione nelle applicazioni IBM MQ” a pagina 856
- z/OS “Punti di sincronizzazione nelle applicazioni IBM MQ for z/OS” a pagina 857
- IBM i “Syncpoint in applicazioni CICS per IBM i” a pagina 860
- “Punti di sincronizzazione in IBM MQ for Multiplatforms” a pagina 860
- IBM i “Interfacce con il gestore del punto di sincronizzazione esterno IBM i” a pagina 865

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Considerazioni sul punto di sincronizzazione nelle applicazioni IBM MQ

Utilizzare queste informazioni per informazioni sull'utilizzo dei punti di sincronizzazione nelle applicazioni IBM MQ .

Il commit a due fasi è supportato dai seguenti ambienti:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Transaction Server per z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS z/OS batch con RRS
- Altri coordinatori esterni che utilizzano l'interfaccia X/Open XA

Il commit a fase singola è supportato dai seguenti ambienti:

- Multi IBM MQ for Multiplatforms
- z/OS z/OS batch

Per ulteriori informazioni sulle interfacce esterne, consultare “Interfacce con gestori syncpoint esterni su Multiplatforms” a pagina 863e la documentazione XA *CAE Specification Distributed Transaction Processing: The XA Specification*, pubblicata da The Open Group. I gestori transazioni (come CICS, IMS, Encina e Tuxedo) possono partecipare al commit a due fasi, coordinato con altre risorse recuperabili. Ciò significa che le funzioni di accodamento fornite da IBM MQ possono essere portate nell'ambito di un'unità di lavoro, gestita dal gestore transazioni.

Gli esempi forniti con IBM MQ mostrano IBM MQ il coordinamento di database compatibili con XA. Per ulteriori informazioni su questi esempi, consultare “Utilizzo dei programmi procedurali di esempio IBM MQ” a pagina 1060.

Nell'applicazione IBM MQ , è possibile specificare su ogni chiamata put e get se si desidera che la chiamata sia sotto il controllo del punto di sincronizzazione. Per far funzionare un'operazione di

inserimento sotto il controllo del punto di sincronizzazione, utilizzare il valore MQPMO_SYNCPOINT nel campo *Options* della struttura MQPMO quando si richiama MQPUT. Per un'operazione get, utilizzare il valore MQGMO_SYNCPOINT nel campo *Options* della struttura MQGMO. Se non si sceglie esplicitamente un'opzione, l'azione predefinita dipende dalla piattaforma:

- Multi Il valore predefinito del controllo del punto di sincronizzazione è NO.
- z/OS Il valore predefinito del controllo del punto di sincronizzazione è YES.

Quando una chiamata MQPUT1 viene emessa con MQPMO_SYNCPOINT, il funzionamento predefinito cambia, in modo che l'operazione di inserimento venga completata in modo asincrono. Ciò potrebbe causare una modifica nel comportamento di alcune applicazioni che si basano su determinati campi nelle strutture MQOD e MQMD che vengono restituiti, ma che ora contengono valori non definiti. Un'applicazione può specificare MQPMO_SYNC_RESPONSE per garantire che l'operazione di inserimento venga eseguita in maniera sincrona e che tutti i valori dei campi appropriati siano completati.

Quando l'applicazione riceve un codice motivo MQRC_BACKED_OUT in risposta a un MQPUT o MQGET nel punto di sincronizzazione, l'applicazione normalmente deve eseguire il backout della transazione corrente utilizzando MQBACK e quindi, se appropriato, riprovare l'intera transazione. Se l'applicazione riceve MQRC_BACKED_OUT in risposta a una chiamata MQCMIT o MQDISC, non è necessario richiamare MQBACK.

Ogni volta che viene eseguito il backout di una chiamata MQGET, il campo *BackoutCount* della struttura MQMD del messaggio interessato viene incrementato. Un valore elevato di *BackoutCount* indica un messaggio di cui è stato ripetutamente eseguito il backout. Ciò potrebbe indicare un problema con questo messaggio, che è necessario esaminare. Consultare BackoutCount per dettagli su *BackoutCount*.

Ad eccezione del batch z/OS con RRS, se un programma emette la chiamata MQDISC mentre sono presenti richieste di cui non è stato eseguito il commit, si verifica un punto di sincronizzazione implicito. Se il programma termina in modo anomalo, si verifica un backout implicito.

z/OS Su z/OS, un punto di sincronizzazione implicito si verifica anche se il programma termina normalmente senza prima richiamare MQDISC. Il programma si considera terminato normalmente se il TCB connesso a MQ termina normalmente. Quando si utilizza z/OS UNIX System Services e LE (Language Environment), viene richiamata la gestione della condizione predefinita per le interruzioni o i segnali. I gestori condizioni LE elaborano la condizione di errore e il TCB termina normalmente. In queste condizioni, MQ esegue il commit dell'unità di lavoro. Per ulteriori informazioni, consultare Introduzione alla gestione delle condizioni Language Environment.

z/OS Per i programmi IBM MQ for z/OS , è possibile utilizzare l'opzione MQGMO_MARK_SKIP_BACKOUT per specificare che un messaggio non deve essere ripristinato se si verifica il backout (per evitare un loop *MQGET - error - backout*). Per informazioni sull'utilizzo di questa opzione, consultare “Backout ignorato” a pagina 800.

Le modifiche agli attributi della coda (dalla chiamata MQSET o dai comandi) non sono influenzate dal commit o dal backout delle unità di lavoro.

z/OS *Punti di sincronizzazione nelle applicazioni IBM MQ for z/OS*

Questo argomento spiega come utilizzare i punti di sincronizzazione nel gestore transazioni (CICS e IMS) e le applicazioni batch.

z/OS *Punti di sincronizzazione in applicazioni CICS Transaction Server per z/OS*

In un'applicazione CICS si stabilisce un punto di sincronizzazione utilizzando il comando EXEC CICS SYNCPOINT.

Per annullare tutte le modifiche al punto di sincronizzazione precedente, è possibile utilizzare il comando EXEC CICS SYNCPOINT ROLLBACK. Per ulteriori informazioni, consultare *CICS Application Programming Reference*.

Se altre risorse recuperabili sono coinvolte nell'unità di lavoro, il gestore code (insieme al gestore syncpoint CICS) partecipa a un protocollo di commit a due fasi; altrimenti, il gestore code esegue un processo di commit a fase singola.

Se un'applicazione CICS emette la chiamata MQDISC, non viene utilizzato alcun punto di sincronizzazione implicito. Se l'applicazione viene chiusa normalmente, tutte le code aperte vengono chiuse e si verifica un commit implicito. Se l'applicazione si chiude in modo anomalo, tutte le code aperte vengono chiuse e si verifica un backout implicito.

z/OS *Punti di sincronizzazione nelle applicazioni IMS*

In un'applicazione IMS , stabilire un punto di sincronizzazione utilizzando chiamate IMS come GU (get unique) a IOPCB e CHKP (checkpoint).

Per annullare tutte le modifiche dal punto di controllo precedente, è possibile utilizzare la chiamata IMS ROLB (rollback). Per ulteriori informazioni, consultare la documentazione di IMS .

Il gestore code (insieme al gestore del punto di sincronizzazione IMS) partecipa a un protocollo di commit a due fasi se anche altre risorse recuperabili sono coinvolte nell'unità di lavoro.

Tutte le maniglie aperte vengono chiuse dall'adattatore IMS in un punto di sincronizzazione (tranne in un ambiente BMP batch o non basato su messaggi). Ciò è dovuto al fatto che un utente differente potrebbe avviare la successiva unità di lavoro e il controllo di sicurezza di IBM MQ viene eseguito quando vengono effettuate le chiamate MQCONN, MQCONNX e MQOPEN, non quando vengono effettuate le chiamate MQPUT o MQGET.

Tuttavia, in un ambiente WFI (Wait - for - Input) o PWFI (pseudo Wait - for - Input) IMS non notifica IBM MQ di chiudere gli handle fino a quando non arriva il messaggio successivo o non viene restituito un codice di stato QC all'applicazione. Se l'applicazione è in attesa nella regione IMS e uno di questi handle appartiene a code attivate, l'attivazione non si verificherà perché le code sono aperte. Per questo motivo, le applicazioni in esecuzione in un ambiente WFI o PWFI devono eseguire esplicitamente MQCLOSE la gestione della coda prima di eseguire la GU all'IOPCB per il messaggio successivo.

Se un'applicazione IMS (BMP o MPP) emette la chiamata MQDISC, le code aperte vengono chiuse ma non viene utilizzato alcun punto di sincronizzazione implicito. Se l'applicazione viene chiusa normalmente, tutte le code aperte vengono chiuse e si verifica un commit implicito. Se l'applicazione si chiude in modo anomalo, tutte le code aperte vengono chiuse e si verifica un backout implicito.

z/OS *Punti di sincronizzazione nelle applicazioni batch z/OS*

Per le applicazioni batch, è possibile utilizzare le chiamate di gestione syncpoint IBM MQ : MQCMIT e MQBACK. Per la compatibilità con le versioni precedenti, CSQBCMT e CSQBBAK sono disponibili come sinonimi.

Nota: Se è necessario eseguire il commit o il backout degli aggiornamenti alle risorse gestite da diversi gestori risorse, come IBM MQ e Db2, all'interno di una singola unità di lavoro è possibile utilizzare RRS. Per ulteriori informazioni, fare riferimento a “Gestione delle transazioni e servizi di gestione delle risorse recuperabili” a pagina 859.

Commit delle modifiche utilizzando la chiamata MQCMIT

Come input, è necessario fornire l'handle di connessione (*Hconn*) restituito dalla chiamata MQCONN o MQCONNX.

L'output di MQCMIT è un codice di completamento e un codice motivo. La chiamata viene completata con un'avvertenza se il punto di sincronizzazione è stato completato, ma il gestore code ha eseguito il backout delle operazioni put e get dal punto di sincronizzazione precedente.

Il corretto completamento della chiamata MQCMIT indica al gestore code che l'applicazione ha raggiunto un punto di sincronizzazione e che tutte le operazioni di inserimento e ricezione effettuate dal precedente punto di sincronizzazione sono state rese permanenti.

Non tutte le risposte di errore indicano che MQCMIT non è stato completato. Ad esempio, l'applicazione può ricevere MQRC_CONNECTION_BROKEN.

Esiste una descrizione della chiamata MQCMIT in MQCMIT.

Ripristino delle modifiche mediante la chiamata MQBACK

Come input, è necessario fornire un handle di connessione (*Hconn*). Utilizzare l'handle restituito dalla chiamata MQCONN o MQCONNX.

L'output di MQBACK è un codice di completamento e un codice motivo.

L'output indica al gestore code che l'applicazione ha raggiunto un punto di sincronizzazione e che tutti i richiami e gli inserimenti effettuati dall'ultimo punto di sincronizzazione sono stati ripristinati.

Esiste una descrizione della chiamata MQBACK in MQBACK.

Gestione delle transazioni e servizi di gestione delle risorse recuperabili

RRS (transaction management and recoverable resource manager services) è una funzione z/OS per fornire il supporto del punto di sincronizzazione a due fasi tra i gestori risorse partecipanti.

Un'applicazione può aggiornare le risorse recuperabili gestite da vari gestori risorse z/OS come IBM MQ e Db2, quindi eseguire il commit o il backout di tali aggiornamenti come una singola unità di lavoro. RRS fornisce la registrazione dello stato dell'unità di lavoro necessaria durante l'esecuzione normale, coordina l'elaborazione del punto di sincronizzazione e fornisce le informazioni sullo stato dell'unità di lavoro appropriate durante il riavvio del sottosistema.

IBM MQ for z/OS Il supporto partecipante RRS consente alle applicazioni IBM MQ negli ambienti batch, TSO e delle procedure memorizzate Db2 di aggiornare le risorse IBM MQ e nonIBM MQ (ad esempio, Db2) all'interno di una singola unità logica di lavoro. Per informazioni sul supporto dei partecipanti RRS, consultare *z/OS MVS Programming: Resource Recovery*.

L'applicazione IBM MQ può utilizzare MQCMIT e MQBACK o le chiamate RRS equivalenti, SRRCMIT e SRRBACK. Per ulteriori informazioni, consultare “L'adattatore batch RRS” a pagina 893.

Disponibilità RRS

Se RRS non è attivo sul sistema z/OS , qualsiasi chiamata IBM MQ emessa da un programma collegato allo stub RRS (CSQBRSTB o CSQBRRSI) restituisce MQRC_ENVIRONMENT_ERROR.

Db2Procedure memorizzate

Se si utilizzano le procedure memorizzate Db2 con RRS, tenere presente quanto segue:

- Le procedure memorizzate Db2 che utilizzano RRS devono essere gestite da WLM (workload manager).
- Se una procedura memorizzata gestita da Db2contiene chiamate IBM MQ ed è collegata allo stub RRS (CSQBRSTB o CSQBRRSI), la chiamata MQCONN o MQCONNX restituisce MQRC_ENVIRONMENT_ERROR.
- Se una procedura memorizzata gestita da WLM contiene chiamate IBM MQ ed è collegata ad uno stub non RRS, la chiamata MQCONN o MQCONNX restituisce MQRC_ENVIRONMENT_ERROR, a meno che non sia la prima chiamata IBM MQ eseguita dall'avvio dello spazio di indirizzo della procedura memorizzata.
- Se la procedura memorizzata Db2 contiene chiamate IBM MQ ed è collegata ad uno stub non RRS, le risorse IBM MQ aggiornate in tale procedura memorizzata non vengono sottoposte a commit fino a quando lo spazio di indirizzo della procedura memorizzata non termina o fino a quando una procedura memorizzata successiva non esegue un MQCMIT (utilizzando uno stub IBM MQ Batch/TSO).
- Più copie della stessa procedura memorizzata possono essere eseguite contemporaneamente nello stesso spazio di indirizzo. Accertarsi che il programma sia codificato in modo rientrante se si desidera che Db2 utilizzi una singola copia della procedura memorizzata. Altrimenti, è possibile ricevere MQRC_HCONN_ERROR su qualsiasi chiamata IBM MQ nel programma.
- Non codificare MQCMIT o MQBACK in una procedura memorizzata Db2 gestita da WLM.

- Progetta tutti i programmi da eseguire in Language Environment (LE).

IBM i *Syncpoint in applicazioni CICS per IBM i*

IBM MQ for IBM i partecipa a CICS per IBM i unità di lavoro. È possibile utilizzare MQI all'interno di un'applicazione CICS for IBM i per inserire e richiamare messaggi all'interno dell'unità di lavoro corrente.

È possibile utilizzare il comando EXEC CICS SYNCPOINT per stabilire un punto di sincronizzazione che includa operazioni IBM MQ for IBM i . Per ripristinare tutte le modifiche fino al punto di sincronizzazione precedente, è possibile utilizzare il comando EXEC CICS SYNCPOINT ROLLBACK.

Se si utilizza MQPUT, MQPUT1o MQGET con l'opzione MQPMO_SYNCPOINT o MQGMO_SYNCPOINT, impostata in un'applicazione CICS per IBM i , non è possibile scollegarsi CICS per IBM i fino a quando IBM MQ for IBM i non ha rimosso la registrazione come risorsa di commit API. Eseguire il commit o il backout delle operazioni put o get in sospeso prima di disconnettersi dal gestore code. Ciò consente di scollegare CICS per IBM i.

Multi *Punti di sincronizzazione in IBM MQ for Multiplatforms*

Il supporto del punto di sincronizzazione opera su due tipi di unità di lavoro: locale e globale.

Un'unità di lavoro *locale* è un'unità in cui le uniche risorse aggiornate sono quelle del gestore code IBM MQ . Qui il coordinamento del punto di sincronizzazione viene fornito dal gestore code stesso utilizzando una procedura di commit a fase singola.

Un'unità di lavoro *globale* è un'unità in cui vengono aggiornate anche le risorse appartenenti ad altri gestori risorse, come i database. IBM MQ può coordinare tali unità di lavoro. Possono anche essere coordinati da un controller di commit esterno. Ad esempio:

- Un altro gestore transazioni
- IBM i Il controller di commit IBM i

Per la piena integrità, utilizzare una procedura di commit a due fasi. Il commit a due fasi può essere fornito da gestori transazioni e database conformi a XA. Ad esempio:

- TXSeries
- UDB
- IBM i il controller di commit IBM i

ALW I prodotti IBM MQ possono coordinare unità di lavoro globali utilizzando un processo di commit a due fasi.

IBM i IBM MQ for IBM i può agire come gestore risorse per le unità di lavoro globali in un ambiente WebSphere Application Server , ma non può agire come gestore transazioni.

Punto di sincronizzazione implicito

Durante l'inserimento di messaggi persistenti, IBM MQ è ottimizzato per l'inserimento di messaggi persistenti nel punto di sincronizzazione. Più applicazioni che inseriscono messaggi persistenti nella stessa coda funzionano meglio se tali applicazioni utilizzano il punto di sincronizzazione. Ciò è dovuto alla minore contesa per la coda, se il punto di sincronizzazione viene utilizzato per inserire i messaggi persistenti.

ImplSyncOpenOutput aggiunge un punto di sincronizzazione implicito quando le applicazioni inserono messaggi persistenti al di fuori del punto di sincronizzazione. Ciò fornisce un miglioramento delle prestazioni, senza che le applicazioni siano consapevoli del punto di sincronizzazione implicito.

Il punto di sincronizzazione implicito fornisce solo un miglioramento delle prestazioni quando ci sono più applicazioni che vengono collocate nella coda, poiché riduce il conflitto per la coda. Quindi, **ImplSyncOpenOutput** specifica il numero minimo di applicazioni che hanno una coda aperta per l'output prima che venga aggiunto un punto di sincronizzazione implicito. Il valore predefinito è 2. Ciò

significa che se non si specifica **ImplSyncOpenOutput**, il punto di sincronizzazione implicito viene aggiunto solo se più applicazioni vengono inserite nella coda.

Per ulteriori informazioni, consultare Parametri di ottimizzazione .

Unità di lavoro locali su Multiplatforms
Le unità di lavoro che coinvolgono solo il gestore code sono denominate unità di lavoro *locali* . Il coordinamento del punto di sincronizzazione viene fornito dal gestore code stesso (coordinamento interno) utilizzando un processo di commit a fase singola.

Per avviare un'unità di lavoro locale, l'applicazione emette richieste MQGET, MQPUT o MQPUT1 che specificano l'opzione del punto di sincronizzazione appropriata. L'unità di lavoro viene sottoposta a commit utilizzando MQCMIT o a rollback utilizzando MQBACK. Tuttavia, l'unità di lavoro termina anche quando la connessione tra l'applicazione e il gestore code viene interrotta, intenzionalmente o involontariamente.

Se un'applicazione si disconnette (MQDISC) da un gestore code mentre un'unità di lavoro globale coordinata da IBM MQ è ancora attiva, viene effettuato un tentativo di eseguire il commit dell'unità di lavoro. Se, tuttavia, l'applicazione termina senza disconnettersi, viene eseguito il rollback dell'unità di lavoro poiché si ritiene che l'applicazione sia terminata in modo anomalo.

Unità di lavoro globali su Multiplatforms
Utilizzare le unità di lavoro globali quando è necessario anche includere aggiornamenti alle risorse appartenenti ad altri gestori risorse.

Qui il coordinamento può essere interno o esterno al gestore code:

Coordinamento del punto di sincronizzazione interno

Il coordinamento del gestore code delle unità di lavoro globali non è supportato da IBM MQ for IBM i o IBM MQ for z/OS. Non è supportata in un IBM MQ MQI client ambiente.

In questo caso, IBM MQ esegue il coordinamento. Per avviare un'unità di lavoro globale, l'applicazione emette la chiamata MQBEGIN.

Come input per la chiamata MQBEGIN, è necessario fornire l'handle di connessione (*Hconn*) restituito dalla chiamata MQCONN o MQCONNX. Questo handle rappresenta la connessione al gestore code IBM MQ .

L'applicazione emette richieste MQGET, MQPUT o MQPUT1 che specificano l'opzione del punto di sincronizzazione appropriata. Ciò significa che è possibile utilizzare MQBEGIN per avviare un'unità di lavoro globale che aggiorna le risorse locali, le risorse appartenenti ad altri gestori risorse o entrambi. Gli aggiornamenti effettuati alle risorse appartenenti ad altri gestori risorse vengono effettuati utilizzando l'API di tale gestore risorse. Tuttavia, non è possibile utilizzare la MQI per aggiornare le code che appartengono ad altri gestori code. Immettere MQCMIT o MQBACK prima di avviare ulteriori unità di lavoro (locali o globali).

L'unità di lavoro globale viene sottoposta a commit utilizzando MQCMIT; questo avvia un commit a due fasi di tutti i gestori risorse coinvolti nell'unità di lavoro. Viene utilizzato un processo di commit a due fasi in base al quale i gestori risorse (ad esempio, i gestori database compatibili con XA come Db2, Oraclee Sybase) vengono prima richiesti per preparare il commit. Solo se tutti sono preparati viene chiesto loro di eseguire il commit. Se un gestore risorse segnala che non è possibile eseguire il commit, a ciascuno viene richiesto di eseguire il backout. In alternativa, è possibile utilizzare MQBACK per eseguire il rollback degli aggiornamenti di tutti i gestori risorse.

Se un'applicazione si disconnette (MQDISC) mentre un'unità di lavoro globale è ancora attiva, viene eseguito il commit dell'unità di lavoro. Se, tuttavia, l'applicazione termina senza disconnettersi, viene eseguito il rollback dell'unità di lavoro poiché si ritiene che l'applicazione sia terminata in modo anomalo.

L'output di MQBEGIN è un codice di completamento e un codice motivo.

Quando si utilizza MQBEGIN per avviare un'unità di lavoro globale, vengono inclusi tutti i gestori risorse esterni configurati con il gestore code. Tuttavia, la chiamata avvia un'unità di lavoro ma viene completata con un'avvertenza se:

- Non sono presenti gestori risorse partecipanti (ossia, non è stato configurato alcun gestore risorse con il gestore code)

o

- Uno o più gestori risorse non sono disponibili.

In questi casi, l'unità di lavoro deve includere aggiornamenti solo ai gestori risorse che erano disponibili quando è stata avviata l'unità di lavoro.

Se uno dei gestori risorse non è in grado di eseguire il commit dei relativi aggiornamenti, a tutti i gestori risorse viene richiesto di eseguire il rollback dei relativi aggiornamenti e MQCMIT viene completato con un'avvertenza. In circostanze inusuali (di solito, l'intervento dell'operatore), una chiamata MQCMIT potrebbe non riuscire se alcuni gestori risorse eseguono il commit dei propri aggiornamenti ma altri ne eseguono il rollback; si ritiene che il lavoro sia stato completato con un risultato *misto* . Tali ricorrenze vengono diagnosticate nel log degli errori del gestore code in modo che sia possibile intraprendere un'azione correttiva.

Un MQCMIT di un'unità di lavoro globale ha esito positivo se tutti i gestori risorse coinvolti eseguono il commit dei relativi aggiornamenti.

Per una descrizione della chiamata MQBEGIN, consultare MQBEGIN.

Coordinamento del punto di sincronizzazione esterno

Ciò si verifica quando è stato selezionato un coordinatore del punto di sincronizzazione diverso da IBM MQ ; ad esempio, CICS, Encina o Tuxedo.

In questa situazione, i sistemi IBM MQ for AIX, Linux, and Windows registrano il loro interesse nel risultato dell'unità di lavoro con il coordinatore del punto di sincronizzazione in modo che possano eseguire il commit o il rollback di tutte le operazioni get o put non sottoposte a commit come richiesto. Il coordinatore del punto di sincronizzazione esterno determina se vengono forniti protocolli di commit a una o due fasi.

Quando si utilizza un coordinatore esterno, MQCMIT, MQBACK e MQBEGIN non possono essere emessi. Le chiamate a queste funzioni non riescono con il codice motivo MQRC_ENVIRONMENT_ERROR.

Il modo in cui viene avviata un'unità di lavoro coordinata esternamente dipende dall'interfaccia di programmazione fornita dal coordinatore del punto di sincronizzazione. Potrebbe essere richiesta una chiamata esplicita. Se è richiesta una chiamata esplicita e si emette una chiamata MQPUT specificando l'opzione MQPMO_SYNCPOINT quando un'unità di lavoro non è avviata, viene restituito il codice di completamento MQRC_SYNCPOINT_NOT_AVAILABLE.

L'ambito dell'unità di lavoro è determinato dal coordinatore del punto di sincronizzazione. Lo stato della connessione tra l'applicazione e il gestore code influisce sull'esito positivo o negativo delle chiamate MQI che un'applicazione emette e non sullo stato dell'unità di lavoro. Un'applicazione può, ad esempio, disconnettersi e riconnettersi a un gestore code durante un'unità di lavoro attiva ed eseguire ulteriori operazioni MQGET e MQPUT all'interno della stessa unità di lavoro. Ciò è noto come disconnessione in sospeso.

È possibile utilizzare le chiamate API IBM MQ nei programmi CICS , se si sceglie di utilizzare le capacità XA di CICS. Se non si utilizza XA, le operazioni di inserimento e ricezione dei messaggi da e verso le code non verranno gestite all'interno delle unità di lavoro atomiche CICS . Un motivo per scegliere questo metodo è che la coerenza generale dell'unità di lavoro non è importante per voi.

Se l'integrità delle unità di lavoro è importante per l'utente, è necessario utilizzare XA. Quando si utilizza XA, CICS utilizza un protocollo di commit a due fasi per garantire che tutte le risorse all'interno dell'unità di lavoro vengano aggiornate insieme.

Per ulteriori informazioni sull'impostazione del supporto transazionale, consultare Scenari di supporto transazionalee la documentazione TXSeries CICS , ad esempio, *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

Multi *Punto di sincronizzazione implicito su Multiplatforms*

Il supporto del punto di sincronizzazione implicito abilita gli inserimenti di messaggi permanenti al di fuori del punto di sincronizzazione.

Durante l'inserimento di messaggi persistenti, IBM MQ è ottimizzato per l'inserimento di messaggi persistenti nel punto di sincronizzazione. Più applicazioni che inseriscono simultaneamente messaggi persistenti nella stessa coda generalmente funzionano meglio se tali applicazioni utilizzano il punto di sincronizzazione. Ciò è dovuto al fatto che la strategia di blocco di IBM MQ è più efficiente se si utilizza il punto di sincronizzazione durante l'inserimento di messaggi persistenti.

Il parametro **ImplSyncOpenOutput** nel file `qm.ini` , controlla se è possibile aggiungere un punto di sincronizzazione implicito quando le applicazioni inseriscono messaggi persistenti al di fuori del punto di sincronizzazione. Ciò può fornire un miglioramento delle prestazioni, senza che le applicazioni siano a conoscenza del punto di sincronizzazione implicito.

Il punto di sincronizzazione implicito fornisce solo un miglioramento delle prestazioni quando ci sono più applicazioni che si inseriscono simultaneamente nella coda, perché riduce il conflitto di blocchi. **ImplSyncOpenOutput** specifica il numero minimo di applicazioni che hanno una coda aperta per l'output prima di poter aggiungere un punto di sincronizzazione implicito. Il valore predefinito è *2*. Ciò significa che se non si specifica esplicitamente **ImplSyncOpenOutput**, il punto di sincronizzazione implicito viene aggiunto solo se più applicazioni vengono inserite nella coda.

Se si aggiunge un punto di sincronizzazione implicito, le statistiche riflettono tale situazione e potrebbe essere visualizzato un output della transazione da **runmqsc display conn**.

Impostare **ImplSyncOpenOutput**=*OFF* se non si desidera mai aggiungere un punto di sincronizzazione implicito.

Per ulteriori informazioni, consultare Parametri di ottimizzazione .

Interfacce con gestori syncpoint esterni su Multiplatforms

IBM MQ for Multiplatforms supporta il coordinamento delle transazioni da parte di gestori syncpoint esterni che utilizzano l'interfaccia X/Open XA.

Alcuni gestori transazioni XA (TXSeries) richiedono che ogni gestore risorse XA fornisca il proprio nome. Questa è la stringa denominata `name` nella struttura dello switch XA.

- ALW Il gestore risorse per IBM MQ su AIX, Linux, and Windows è denominato MQSeries_XA_RMI.
- IBM i Per IBM i, il nome del gestore risorse è MQSeries XA RMI.

Per ulteriori dettagli sulle interfacce XA, fare riferimento alla documentazione XA *CAE Specification Distributed Transaction Processing: The XA Specification*, pubblicata da The Open Group.

In una configurazione XA, IBM MQ for Multiplatforms svolge il ruolo di gestore risorse XA. Un coordinatore del punto di sincronizzazione XA può gestire una serie di gestori risorse XA e sincronizzare il commit o il backout delle transazioni in entrambi i gestori risorse. Questo è il modo in cui funziona per un gestore risorse registrato staticamente:

1. Un'applicazione notifica al coordinatore del punto di sincronizzazione che desidera avviare una transazione.
2. Il coordinatore del punto di sincronizzazione invia una chiamata a tutti i gestori risorse di cui è a conoscenza, per notificare loro la transazione corrente.
3. L'applicazione emette chiamate per aggiornare le risorse gestite dai gestori risorse associati alla transazione corrente.

4. L'applicazione richiede che il coordinatore del punto di sincronizzazione esegua il commit o il rollback della transazione.
5. Il coordinatore del punto di sincronizzazione emette chiamate a ciascun gestore risorse utilizzando protocolli di commit a due fasi per completare la transazione come richiesto.

La specifica XA richiede che ciascun gestore risorse fornisca una struttura denominata XA Switch. Questa struttura dichiara le funzioni del gestore risorse e le funzioni che devono essere richiamate dal coordinatore del punto di sincronizzazione.

Esistono due versioni di questa struttura:

Tabella 128. Versioni di XA Switch

Versione	**Descrizione**
MQRMIXASwitch	Gestione delle risorse XA statici
MQRMIXASwitchDynamic	Gestione dinamica delle risorse XA

Per un elenco delle librerie che contengono questa struttura, consultare La IBM MQ struttura di switch XA.

Il metodo che deve essere utilizzato per collegarli ad un coordinatore del punto di sincronizzazione XA viene definito dal coordinatore; consultare la documentazione fornita da tale coordinatore per determinare come abilitare IBM MQ a cooperare con il coordinatore del punto di sincronizzazione XA.

La struttura *xa_info* che viene passata su qualsiasi chiamata *xa_open* dal coordinatore del punto di sincronizzazione può essere il nome del gestore code che deve essere gestito. Questo assume lo stesso formato del nome del gestore code passato a MQCONN o MQCONNX e può essere vuoto se deve essere utilizzato il gestore code predefinito. Tuttavia, è possibile utilizzare i due parametri aggiuntivi TPM e AXLIB

TPM consente di specificare IBM MQ il nome del gestore transazioni, ad esempio CICS. AXLIB consente di specificare il nome effettivo della libreria nel gestore transazioni in cui si trovano i punti di ingresso XA AX.

Se si utilizza uno di questi parametri o un gestore code non predefinito, è necessario specificare il nome del gestore code utilizzando il parametro QMNAME. Per ulteriori informazioni, consultare I parametri CHANNEL, TRPTYPE, CONNAME e QMNAME della stringa xa_open.

Limitazioni

1. Le unità di lavoro globali non sono consentite con un Hconn condiviso (come descritto in “Connessioni condivise (indipendenti dal thread) con MQCONNX” a pagina 739.
2. IBM i IBM MQ for IBM i non supporta la registrazione dinamica dei gestori risorse XA.

   L'unico gestore transazioni supportato è WebSphere Application Server.
3. Windows Su sistemi Windows , tutte le funzioni dichiarate nello switch XA sono dichiarate come funzioni _cdecl.
4. Un coordinatore del punto di sincronizzazione esterno può gestire solo un gestore code alla volta. Ciò è dovuto al fatto che il coordinatore ha una connessione effettiva a ciascun gestore code ed è pertanto soggetto alla regola che è consentita solo una connessione alla volta.

   Nota: Nota: un'applicazione client JMS (applicazione CLIENT JEE) in esecuzione in un server JEE non ha questa restrizione, quindi una singola transazione gestita da server JEE può coordinare più gestori code nella stessa transazione. Tuttavia, un'applicazione server JMS , in esecuzione in modalità bind, è ancora soggetta alla regola per cui è consentita una sola connessione alla volta.
5. Tutte le applicazioni che vengono eseguite utilizzando il coordinatore del punto di sincronizzazione possono connettersi solo al gestore code gestito dal coordinatore perché sono già effettivamente connesse a tale gestore code. Devono emettere MQCONN o MQCONNX per ottenere un handle di connessione e devono emettere MQDISC prima di uscire. In alternativa, possono utilizzare l'uscita UE014015 per TXSeries CICS.

IBM i *Interfacce con il gestore del punto di sincronizzazione esterno IBM i*

IBM MQ for IBM i può utilizzare il controllo del commit IBM i nativo come coordinatore del punto di sincronizzazione esterno.

Le connessioni (condivise) indipendenti dal sottoprocesso non sono consentite con il controllo di commit. Consultare il manuale *IBM i Programming: Backup and Recovery Guide, SC21-8079* per ulteriori informazioni sulle funzioni di controllo del commit di IBM i.

Per avviare le funzioni di controllo del commit IBM i , utilizzare il comando del sistema STRCMTCTL. Per terminare il controllo di sincronia, utilizzare il comando di sistema ENDCMTCTL.

Nota: Il valore predefinito di *Ambito definizione di commit* è *ACTGRP. Deve essere definito come *JOB per IBM MQ per IBM i. Ad esempio:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i può anche eseguire unità di lavoro locali contenenti solo aggiornamenti alle risorse IBM MQ . La scelta tra le unità di lavoro locali e la partecipazione nelle unità di lavoro globali coordinate da IBM i viene effettuata in ogni applicazione quando l'applicazione richiama MQPUT, MQPUT1o MQGET, specificando MQPMO_SYNCPOINT o MQGMO_SYNCPOINT o MQBEGIN. Se il controllo di commit non è attivo quando viene emessa la prima chiamata di questo tipo, IBM MQ avvia un'unità di lavoro locale e tutte le ulteriori unità di lavoro per questa connessione a IBM MQ utilizzano anche le unità di lavoro locali, indipendentemente dal fatto che il controllo di commit venga avviato o meno. Per eseguire il commit di un'unità di lavoro locale, utilizzare MQCMIT. Per eseguire il backout di un'unità di lavoro locale, utilizzare MQBACK. Le chiamate di commit e rollback IBM i come il comando CL COMMIT non hanno alcun effetto sulle unità di lavoro locali IBM MQ .

Se si desidera utilizzare il controllo del commit IBM MQ for IBM i con IBM i nativo come coordinatore del punto di sincronizzazione esterno, assicurarsi che qualsiasi lavoro con il controllo del commit sia attivo e che si stia utilizzando IBM MQ in un lavoro a thread singolo. Se si richiama MQPUT, MQPUT1o MQGET, specificando MQPMO_SYNCPOINT o MQGMO_SYNCPOINT, in un lavoro a più sottoprocessi in cui è stato avviato il controllo del commit, la chiamata ha esito negativo con un codice motivo MQRC_SYNCPOINT_NOT_AVAILABLE.

È possibile utilizzare le unità di lavoro locali e le chiamate MQCMIT e MQBACK in un lavoro a più sottoprocessi.

Se si richiama MQPUT, MQPUT1o MQGET, specificando MQPMO_SYNCPOINT o MQGMO_SYNCPOINT, dopo aver avviato il controllo di commit, IBM MQ for IBM i si aggiunge come risorsa di commit API alla definizione di commit. Questa è in genere la prima chiamata di questo tipo in un lavoro. Mentre ci sono delle risorse di commit API registrate in una particolare definizione di commit, non è possibile terminare il controllo di commit per tale definizione.

IBM MQ for IBM i rimuove la relativa registrazione come risorsa di commit API quando ci si disconnette dal gestore code, se non ci sono operazioni MQI in sospeso nell'unità di lavoro corrente.

Se ci si disconnette dal gestore code mentre ci sono operazioni MQPUT, MQPUT1o MQGET in sospeso nell'unità di lavoro corrente, IBM MQ for IBM i rimane registrato come una risorsa di commit API in modo che venga notificato il successivo commit o rollback. Quando il punto di sincronizzazione successivo viene raggiunto, IBM MQ for IBM i esegue il commit o il rollback delle modifiche come richiesto. Un'applicazione può disconnettere e riconnettersi a un gestore code durante un'unità di lavoro attiva ed eseguire ulteriori operazioni MQGET e MQPUT all'interno della stessa unità di lavoro (si tratta di una disconnessione in sospeso).

Se si tenta di immettere un comando di sistema ENDCMTCTL per tale definizione di commit, viene emesso il messaggio CPF8355 , che indica che le modifiche in sospeso erano attive. Questo messaggio viene visualizzato anche nella registrazione lavoro al termine del lavoro. Per evitare ciò, eseguire il commit o il rollback di tutte le operazioni IBM MQ for IBM i in sospeso e disconnettersi dal gestore code. Pertanto, l'utilizzo dei comandi COMMIT o ROLLBACK prima di ENDCMTCTL abilita il completamento con esito positivo del controllo di commit finale.

Quando si utilizza il controllo del commit IBM i come coordinatore del punto di sincronizzazione esterno, non è possibile emettere chiamate MQCMIT, MQBACK e MQBEGIN. Le chiamate a queste funzioni non riescono con il codice motivo MQRC_ENVIRONMENT_ERROR.

Per eseguire il commit o il rollback (ovvero, per eseguire il backout) della propria unità di lavoro, utilizzare uno dei linguaggi di programmazione che supporta il controllo del commit. Ad esempio:

- Comandi CL: COMMIT e ROLLBACK
- Funzioni di programmazione ILE C: _Rcommit e _Rrollback
- ILE RPG: COMMIT e ROLBK
- COBOL/400: COMMIT e ROLLBACK

Quando si utilizza il controllo di commit IBM i come coordinatore del punto di sincronizzazione esterno con IBM MQ for IBM i, IBM i esegue un protocollo di commit a due fasi a cui partecipa IBM MQ . Poiché ogni unità di lavoro viene sottoposta a commit in due fasi, il gestore code potrebbe diventare non disponibile per la seconda fase dopo aver votato per il commit nella prima fase. Ciò può verificarsi, ad esempio, se i lavori interni del gestore code vengono terminati. In questa situazione, la registrazione lavoro che esegue il commit contiene il messaggio CPF835F che indica che un'operazione di commit o rollback non è riuscita. I messaggi precedenti indicano la causa del problema, se si è verificato durante un'operazione di commit o di rollback e anche l'ID LUWID (logical unit of work ID) per l'unità di lavoro non riuscita.

Se il problema è stato causato dall'errore della risorsa di commit API IBM MQ durante il commit o il rollback di un'unità di lavoro preparata, è possibile utilizzare il comando WRKMQMTRN per completare l'operazione e ripristinare l'integrità della transazione. Il comando richiede che l'utente conosca il LUWID dell'unità di lavoro di cui eseguire il commit e il backout.

Avvio delle applicazioni IBM MQ utilizzando i trigger

Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

Alcune applicazioni IBM MQ che servono le code vengono eseguite continuamente, quindi sono sempre disponibili per richiamare i messaggi che arrivano sulle code. Tuttavia, è possibile che ciò non sia necessario quando il numero di messaggi in arrivo nelle code è imprevedibile. In questo caso, le applicazioni potrebbero utilizzare le risorse di sistema anche quando non ci sono messaggi da recuperare.

IBM MQ fornisce una funzione che consente di avviare automaticamente un'applicazione quando sono disponibili messaggi da richiamare. Questa funzione è nota come *attivazione*.

Per informazioni sull'attivazione dei canali, vedere Trigger dei canali.

Cos' è l'attivazione?

Il gestore code definisce alcune condizioni che costituiscono *eventi trigger*.

Se il trigger è abilitato per una coda e si verifica un evento trigger, il gestore code invia un messaggio di *trigger* a una coda denominata *coda di iniziazione*. La presenza del messaggio trigger sulla coda di iniziazione indica che si è verificato un evento trigger.

I messaggi trigger generati dal gestore code non sono persistenti. Ciò riduce la registrazione (migliorando le prestazioni) e riducendo i duplicati durante il riavvio, migliorando il tempo di riavvio.

Il programma che elabora la coda di iniziazione viene chiamato *applicazione trigger - monitor*e la sua funzione è quella di leggere il messaggio del trigger e intraprendere l'azione appropriata, in base alle informazioni contenute nel messaggio del trigger. Di solito, questa azione consente di avviare un'altra applicazione per elaborare la coda che ha generato il messaggio trigger. Dal punto di vista del gestore code, non c'è nulla di speciale nell'applicazione di controllo dei trigger; si tratta semplicemente di un'altra applicazione che legge i messaggi da una coda (la coda di avvio).

Se il trigger è abilitato per una coda, è possibile creare un *oggetto di definizione del processo* ad esso associato. Questo oggetto contiene informazioni sull'applicazione che elabora il messaggio che ha causato l'evento trigger. Se l'oggetto di definizione del processo viene creato, il gestore code estrae

queste informazioni e le inserisce nel messaggio del trigger, per l'utilizzo da parte dell'applicazione di controllo dei trigger. Il nome della definizione del processo associato a una coda viene fornito dall'attributo della coda locale *ProcessName* . Ogni coda può specificare una definizione di processo differente oppure diverse code possono condividere la stessa definizione.

Se si desidera attivare l'avvio di un canale, non è necessario definire un oggetto definizione processo. Viene utilizzata invece la definizione della coda di trasmissione.

Il trigger è supportato dai client IBM MQ in esecuzione su AIX, Linux, and Windows. Un'applicazione in esecuzione in un ambiente client è uguale a quella in esecuzione in un ambiente IBM MQ completo, tranne per il fatto che viene collegata alle librerie client. Tuttavia, il controllo trigger e l'applicazione da avviare devono essere entrambi nello stesso ambiente.

L'attivazione implica:

Coda applicazione
Una *coda dell'applicazione* è una coda locale che, quando ha il trigger impostato e quando le condizioni vengono soddisfatte, richiede la scrittura dei messaggi del trigger.

Definizione di processo
Una coda dell'applicazione può avere un *oggetto definizione processo* associato ad essa che contiene i dettagli dell'applicazione che acquisirà i messaggi dalla coda dell'applicazione. (Consultare Attributi per definizioni di processi per un elenco di attributi.)

Si ricordi che se si desidera che un trigger avvii un canale, non è necessario definire un oggetto definizione processo.

Coda di trasmissione
È necessaria una coda di trasmissione se si desidera che un trigger avvii un canale.

Per una coda di trasmissione su qualsiasi piattaforma diversa da Linux, l'attributo *TriggerData* della coda di trasmissione può specificare il nome del canale da avviare. Può sostituire la definizione del processo per i canali di attivazione, ma viene utilizzata solo quando non viene creata una definizione del processo.

Evento trigger
Un *evento trigger* è un evento che causa la generazione di un messaggio trigger da parte del gestore code. Si tratta di solito di un messaggio che arriva su una coda dell'applicazione, ma può verificarsi anche in altri momenti. Ad esempio, consultare “Condizioni per un evento trigger” a pagina 873.

IBM MQ ha una gamma di opzioni che consentono di controllare le condizioni che causano un evento trigger (consultare “Controllo degli eventi trigger” a pagina 877).

messaggio di trigger
Il gestore code crea un *messaggio trigger* quando riconosce un evento trigger. Copia nel messaggio trigger le informazioni sull'applicazione da avviare. Queste informazioni provengono dalla coda dell'applicazione e dall'oggetto di definizione processo associato alla coda dell'applicazione.

I messaggi di trigger hanno un formato fisso (consultare “Formato dei messaggi di trigger” a pagina 885).

Inizializzazione coda
Una *coda di inizializzazione* è una coda locale in cui il gestore code inserisce i messaggi del trigger. Tenere presente che una coda di iniziazione non può essere una coda alias o una coda modello.

Un gestore code può essere proprietario di più di una coda di iniziazione e ciascuna di esse è associata a una o più code dell'applicazione.

z/OS Una coda condivisa, una coda locale accessibile dai gestori code in un gruppo di condivisione code, può essere una coda di avvio su IBM MQ for z/OS.

Controllo trigger
Un *controllo trigger* è un programma in esecuzione continua che serve una o più code di iniziazione. Quando un messaggio di trigger arriva su una coda di iniziazione, il controllo di trigger richiama il messaggio. Il controllo trigger utilizza le informazioni nel messaggio trigger. Immette un comando per avviare l'applicazione che deve richiamare i messaggi in arrivo sulla coda dell'applicazione, passando

le informazioni contenute nell'intestazione del messaggio trigger, che include il nome della coda dell'applicazione.

Su tutte le piattaforme, uno speciale controllo trigger noto come iniziatore di canali è responsabile dell'avvio dei canali.

z/OS Su z/OS, l'iniziatore di canali viene in genere avviato manualmente oppure può essere eseguito automaticamente all'avvio di un gestore code modificando CSQINP2 nel JCL di avvio del gestore code.

Multi Su Multiplatforms, l'iniziatore di canali viene avviato automaticamente all'avvio del gestore code oppure manualmente con il comando **runmqchi** .

Per ulteriori informazioni, consultare “Elaborazione della coda di iniziazione da parte dei controlli trigger” a pagina 881.

Per comprendere come funziona il trigger, considerare Figura 95 a pagina 868, che è un esempio di tipo di trigger FIRST (MQTT_FIRST).

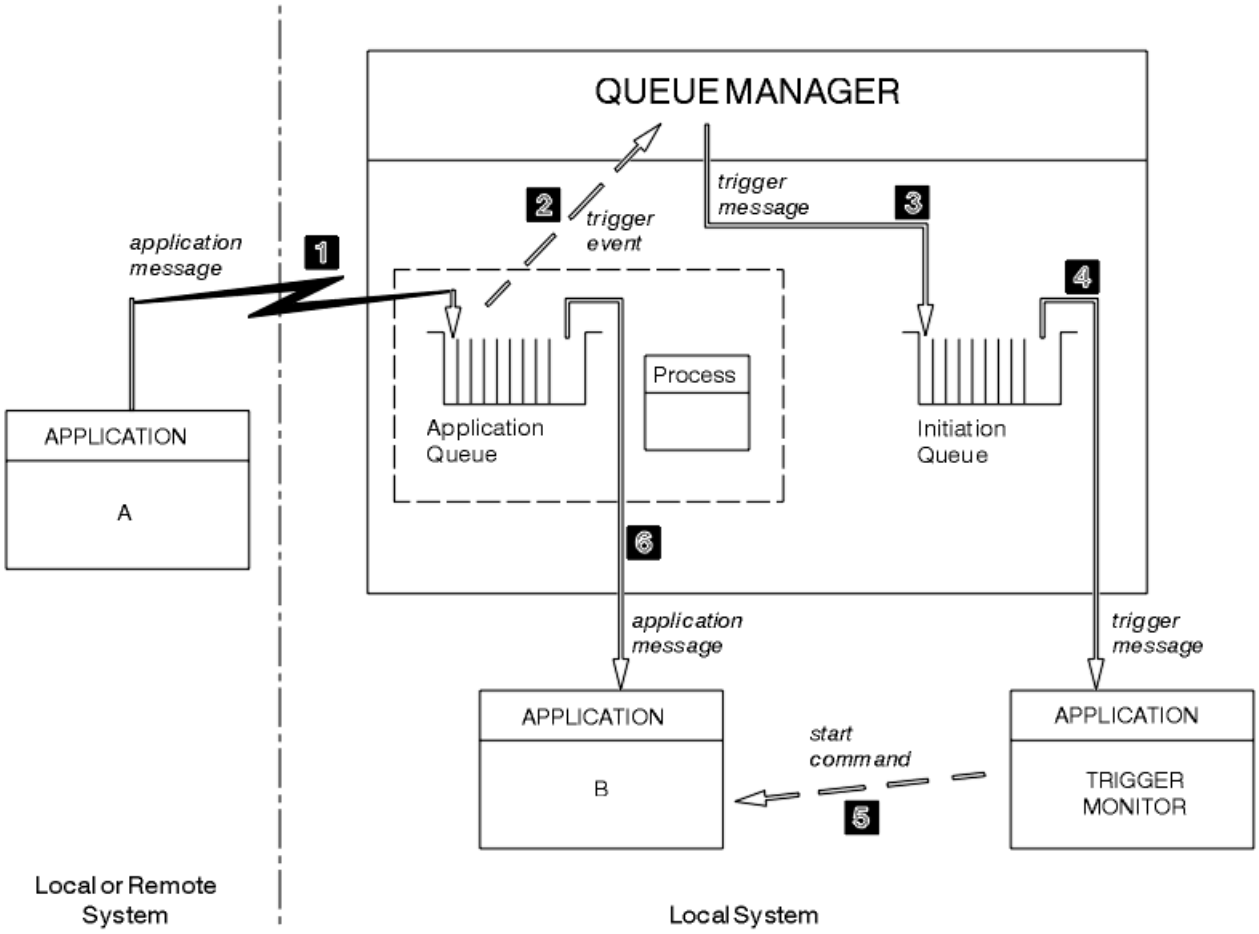


Figura 95. Flusso di messaggi di applicazione e trigger

In Figura 95 a pagina 868, la sequenza di eventi è:

1. L'applicazione A, che può essere locale o remota per il gestore code, inserisce un messaggio nella coda dell'applicazione. Nessuna applicazione ha questa coda aperta per l'immissione. Tuttavia, questo fatto è rilevante solo per attivare il tipo FIRST e DEPTH.
2. Il gestore code controlla se sono soddisfatte le condizioni in base alle quali deve generare un evento trigger. Lo sono e viene generato un evento trigger. Le informazioni contenute nell'oggetto di definizione processo associato vengono utilizzate durante la creazione del messaggio trigger.

3. Il gestore code crea un messaggio trigger e lo inserisce nella coda di iniziazione associata a questa coda dell'applicazione, ma solo se un'applicazione (controllo trigger) ha la coda di iniziazione aperta per l'input.
4. Il controllo trigger richiama il messaggio trigger dalla coda di iniziazione.
5. Il controllo trigger emette un comando per avviare l'applicazione B (l'applicazione server).
6. L'applicazione B apre la coda dell'applicazione e richiama i messaggi.

Nota:

1. Se la coda dell'applicazione è aperta per l'input, da qualsiasi programma, e ha il trigger impostato per FIRST o DEPTH, non si verifica alcun evento trigger perché la coda è già servita.
2. Se la coda di iniziazione non è aperta per l'input, il gestore code non genera alcun messaggio trigger; attende fino a quando un'applicazione non apre la coda di iniziazione per l'input.
3. Quando si utilizza il trigger per i canali, utilizzare il tipo di trigger FIRST o DEPTH.
4. Le applicazioni attivate vengono eseguite con l'ID utente e il gruppo dell'utente che ha avviato il controllo dei trigger, l'utente CICS o l'utente che ha avviato il gestore code.

Finora, la relazione tra le code all'interno del trigger è stata solo su una base uno - a - uno. Considerare Figura 96 a pagina 870.

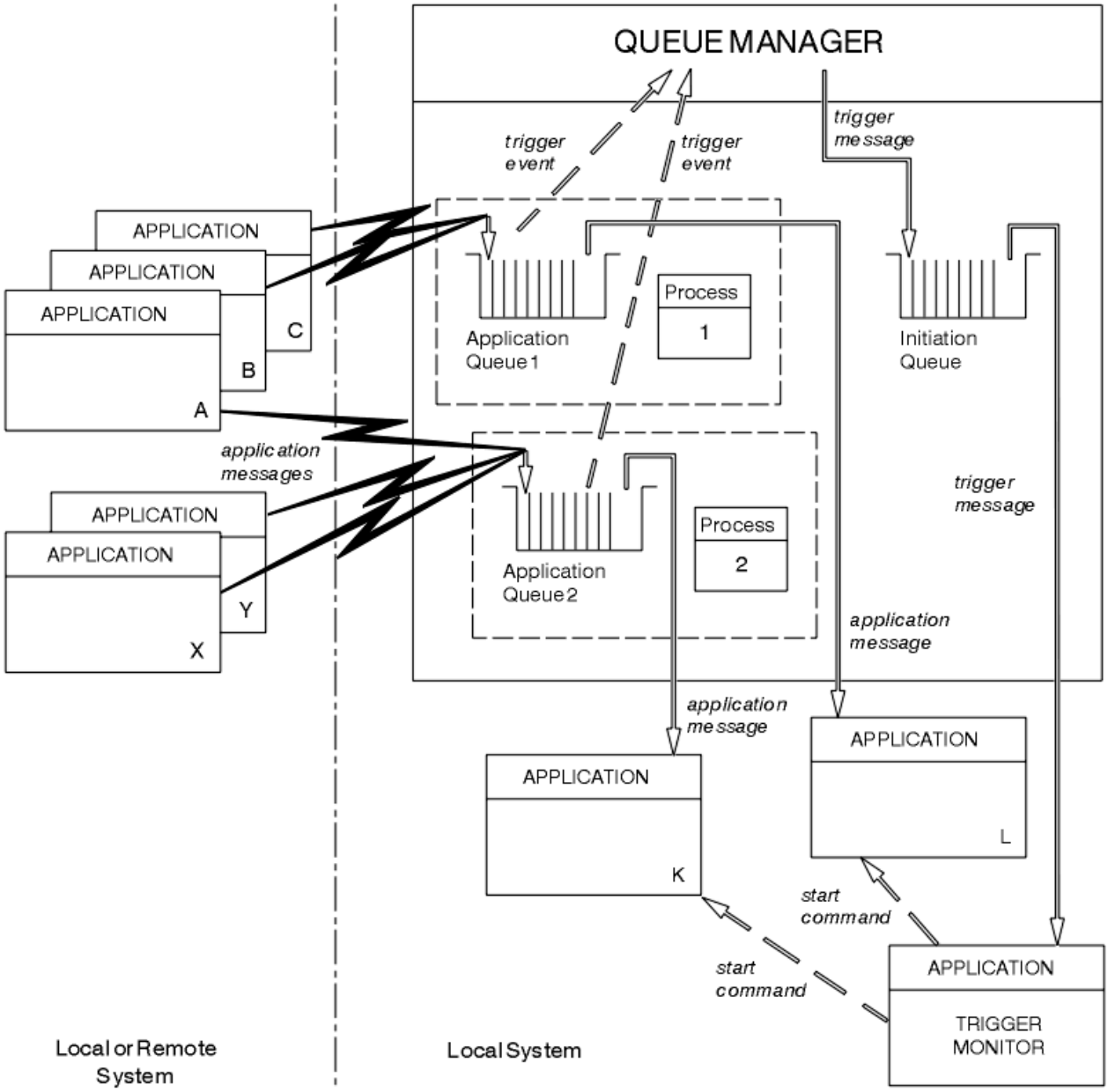


Figura 96. Relazione delle code all'interno del trigger

Una coda dell'applicazione ha un oggetto di definizione processo associato che contiene i dettagli dell'applicazione che elaborerà il messaggio. Il gestore code inserisce le informazioni nel messaggio del trigger, pertanto è necessaria solo una coda di avvio. Il controllo dei trigger estrae queste informazioni dal messaggio del trigger e avvia l'applicazione pertinente per gestire il messaggio su ciascuna coda dell'applicazione.

Tenere presente che, se si desidera attivare l'avvio di un canale, non è necessario definire un oggetto definizione processo. La definizione della coda di trasmissione può determinare il canale da attivare.

Utilizzare i seguenti link per ulteriori informazioni sull'avvio delle applicazioni IBM MQ mediante i trigger:

- “Prerequisiti per l'attivazione” a pagina 871
- “Condizioni per un evento trigger” a pagina 873
- “Controllo degli eventi trigger” a pagina 877
- “Progettazione di un'applicazione che utilizza code attivate” a pagina 879
- “Elaborazione della coda di iniziazione da parte dei controlli trigger” a pagina 881

- “Proprietà dei messaggi trigger” a pagina 884
- “Quando l'attivazione non funziona” a pagina 886

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

Prerequisiti per l'attivazione

Utilizzare queste informazioni per informazioni sui passi da eseguire prima di utilizzare il trigger.

Prima che la tua applicazione possa sfruttare l'attivazione, completa la seguente procedura:

1. Le alternative sono:
   a. Creare una coda di iniziazione per la coda dell'applicazione. Ad esempio:

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   o
   b. Determinare il nome di una coda locale esistente e che può essere utilizzata dall'applicazione (di solito, questo nome è SYSTEM.DEFAULT.INITIATION.QUEUE o, se si stanno avviando canali con trigger, SYSTEM.CHANNEL.INITQ) e specificarne il nome nel campo *InitiationQName* della coda dell'applicazione.

2. Associare la coda di iniziazione alla coda dell'applicazione. Un gestore code può possedere più di una coda di iniziazione. È possibile che si desideri che alcune delle code dell'applicazione vengano servite da programmi differenti, in tal caso, è possibile utilizzare una coda di avvio per ciascun programma

di utilizzo, anche se non è necessario. Di seguito è riportato un esempio di come creare una coda dell'applicazione:

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i Di seguito è riportato un estratto da un programma CL per IBM MQ for IBM i che crea una coda di iniziazione:

```
/*   Queue used by AMQSINQA                                        */
           CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                       QTYPE(*LCL)  REPLACE(*YES)                  +
                       MQMNAME                                     +
                       TEXT('queue for AMQSINQA')                  +
                       SHARE(*YES)                 /* Shareable   */+
                       DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                   +
                       TRGENBL(*YES)  /* Trigger control on      */+
                       TRGTYPE(*FIRST)/* Trigger on first message*/+
                       PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                       INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. Se si sta attivando un'applicazione, creare un oggetto di definizione del processo per contenere le informazioni relative all'applicazione che deve servire la coda dell'applicazione. Ad esempio, per attivare - avviare una transazione di retribuzione CICS denominata PAYR:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i Di seguito è riportato un estratto da un programma CL per IBM MQ for IBM i che crea un oggetto definizione processo:

```
/*   Process definition                                            */
           CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                       REPLACE(*YES)                               +
                       MQMNAME                                     +
                       TEXT('trigger process for AMQSINQA')        +
                       ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                       APPID('AMQSINQA')    /* Program name      */
```

Quando il gestore code crea un messaggio trigger, copia le informazioni dagli attributi dell'oggetto definizione del processo nel messaggio trigger.

Piattaforma	Per creare un oggetto definizione processo
Sistemi AIX, Linux, and Windows	Utilizzare DEFINE PROCESS o SYSTEM.DEFAULT.PROCESS e modifica utilizzando ALTER PROCESS
z/OS z/OS	Utilizzare DEFINE PROCESS (vedere il codice di esempio nel passo “3” a pagina 872) o utilizzare le operazioni e i pannelli di controllo.
IBM i IBM i	Utilizzare un programma CL che contiene il codice come nel passo “3” a pagina 872.

4. Facoltativo: creare una definizione di coda di trasmissione e utilizzare spazi vuoti per l'attributo **ProcessName** .

   L'attributo **TrigData** può contenere il nome del canale da attivare o può essere lasciato vuoto. Tranne su IBM MQ for z/OS, se viene lasciato vuoto, l'iniziatore del canale ricerca i file di definizione del canale fino a quando non trova un canale associato alla coda di trasmissione denominata. Quando il gestore code crea un messaggio trigger, copia le informazioni dall'attributo **TrigData** della definizione della coda di trasmissione nel messaggio trigger.

5. Se è stato creato un oggetto di definizione del processo per specificare le proprietà dell'applicazione che deve servire la coda dell'applicazione, associare l'oggetto del processo alla coda dell'applicazione denominandolo nell'attributo **ProcessName** della coda.

Piattaforma	Utilizza comandi
Sistemi AIX, Linux, and Windows	MODIFICA QLOCAL
z/OS z/OS z/OS	MODIFICA QLOCAL
IBM i IBM i IBM i	CHGMQMQ

6. Avviare le istanze dei controlli dei trigger IBM i (o dei server dei trigger in IBM MQ for IBM i) che devono servire le code di iniziazione definite. Per ulteriori informazioni, fare riferimento a “Elaborazione della coda di iniziazione da parte dei controlli trigger” a pagina 881.

Se si desidera essere a conoscenza di eventuali messaggi di trigger non recapitati, assicurarsi che il gestore code disponga di una coda di messaggi non recapitabili (non recapitabile) definita. Specificare il nome della coda nel campo Gestore code *DeadLetterQName* .

È quindi possibile impostare le condizioni trigger richieste, utilizzando gli attributi dell'oggetto coda che definisce la coda dell'applicazione. Per ulteriori informazioni, consultare “Controllo degli eventi trigger” a pagina 877.

Condizioni per un evento trigger

Il gestore code crea un messaggio trigger quando vengono soddisfatte le condizioni descritte in questo argomento.

I riferimenti alle code condivise in questo argomento indicano code condivise in un gruppo di condivisione code, disponibili solo su IBM MQ for z/OS.

Le seguenti condizioni causano la creazione di un messaggio trigger da parte del gestore code:

1. Un messaggio è *inserito* su una coda.
2. Il messaggio ha una priorità maggiore o uguale alla priorità del trigger di soglia della coda. Questa priorità è impostata nell'attributo della coda locale **TriggerMsgPriority** ; se è impostata su zero, qualsiasi messaggio è qualificato.
3. Il numero di messaggi sulla coda con priorità maggiore o uguale a *TriggerMsgPriority* era in precedenza, a seconda di *TriggerType*:
   - Zero (per il tipo di trigger MQTT_FIRST)
   - Qualsiasi numero (per il tipo di trigger MQTT_EVERY)
   - *TriggerDepth* meno 1 (per il tipo di trigger MQTT_DEPTH)

   Nota:

   a. Per le code locali non condivise, il gestore code conta i messaggi di cui è stato eseguito il commit e di cui non è stato eseguito il commit quando valuta se esistono le condizioni per

un evento trigger. Di conseguenza, un'applicazione potrebbe essere avviata quando non ci sono messaggi da richiamare poiché non è stato eseguito il commit dei messaggi sulla coda. In questa situazione, considerare l'utilizzo dell'opzione di attesa con un *WaitInterval*adatto, in modo che l'applicazione attenda l'arrivo dei messaggi.

b. Per le code condivise locali, il gestore code conta solo i messaggi di cui è stato eseguito il commit.

4. Per l'attivazione di tipo FIRST o DEPTH, nessun programma ha la coda dell'applicazione aperta per la rimozione dei messaggi (ossia, l'attributo della coda locale **OpenInputCount** è zero).

   Nota:

   a. Per le code condivise, si applicano condizioni speciali quando più gestori code dispongono di controlli trigger in esecuzione su una coda. In questa situazione, se uno o più gestori code hanno la coda aperta per l'input condiviso, i criteri di trigger sugli altri gestori code vengono considerati come *TriggerType* MQTT_FIRST e *TriggerMsgPriority* zero. Quando tutti i gestori code chiudono la coda per l'input, le condizioni di trigger ritornano a quelle specificate nella definizione della coda.

      Uno scenario di esempio interessato da questa condizione è rappresentato da più gestori code QM1, QM2e QM3 con un controllo trigger in esecuzione per una coda dell'applicazione A. Un messaggio arriva su A che soddisfa le condizioni per l'attivazione e viene generato un messaggio di trigger sulla coda di iniziazione. Il controllo trigger su QM1 richiama il messaggio trigger e attiva un'applicazione. L'applicazione attivata apre la coda dell'applicazione per l'input condiviso. Da questo punto in poi le condizioni di trigger per la coda dell'applicazione A vengono valutate come *TriggerType* MQTT_FIRST e *TriggerMsgPriority* zero sui gestori code QM2 e QM3, fino a quando QM1 non chiude la coda dell'applicazione.

   b. Per le code condivise, questa condizione viene applicata per ciascun gestore code. In altre parole, il valore *OpenInputCount* di un gestore code per una coda deve essere zero affinché un messaggio trigger venga generato per la coda da tale gestore code. Tuttavia, se un gestore code nel gruppo di condivisione code ha la coda aperta utilizzando l'opzione MQOO_INPUT_EXCLUSIVE, nessun messaggio trigger viene generato per tale coda da uno dei gestori code nel gruppo di condivisione code.

      La modifica nella modalità di valutazione delle condizioni di trigger si verifica quando l'applicazione attivata apre la coda per l'input. Negli scenari in cui è in esecuzione un solo controllo trigger, altre applicazioni possono avere lo stesso effetto perché aprono in modo simile la coda dell'applicazione per l'input. Non importa se la coda dell'applicazione è stata aperta da un'applicazione avviata da un controllo dei trigger o da un'altra applicazione; è il fatto che la coda è aperta per l'input su un altro gestore code che causa la modifica nei criteri dei trigger.

5. Su IBM MQ for z/OS, se la coda dell'applicazione è una con un attributo **Usage** di MQUS_NORMAL, le richieste get per essa non sono inibite (ossia, l'attributo della coda **InhibitGet** è MQQA_GET_ALLOWED). Inoltre, se la coda dell'applicazione attivata è una coda con un attributo **Usage** di MQUS_XMITQ, le richieste get per essa non sono inibite.
6. Le alternative sono:

   - L'attributo della coda locale **ProcessName** per la coda non è vuoto e l'oggetto di definizione del processo identificato da tale attributo è stato creato oppure
   - L'attributo della coda locale **ProcessName** per la coda è vuoto, ma la coda è una coda di trasmissione. Poiché la definizione del processo è facoltativa, l'attributo **TriggerData** potrebbe contenere anche il nome del canale da avviare. In tal caso, il messaggio trigger contiene attributi con i seguenti valori:
     - **QName**: nome coda
     - **ProcessName**: spazi
     - **TriggerData**: dati trigger
     - **ApplType**: MQAT_SCONOSCIUTO
     - **ApplId**: spazi
     - **EnvData**: spazi

– **UserData**: spazi

7. È stata creata una coda di iniziazione ed è stata specificata nell'attributo della coda locale **InitiationQName** . Inoltre:
   - Le richieste di richiamo non sono inibite per la coda di iniziazione (ossia, il valore dell'attributo della coda di **InhibitGet** è MQQA_GET_ALLOWED).
   - Le richieste di inserimento non devono essere inibite per la coda di avvio (ovvero, il valore dell'attributo della coda **InhibitPut** deve essere MQQA_PUT_ALLOWED).
   - Il valore dell'attributo **Usage** della coda di iniziazione deve essere MQUS_NORMAL.
   - In ambienti in cui sono supportate code dinamiche, la coda di iniziazione non deve essere una coda dinamica contrassegnata come eliminata logicamente.
8. Un controllo trigger attualmente ha la coda di iniziazione aperta per la rimozione dei messaggi (ovvero, l'attributo della coda locale **OpenInputCount** è maggiore di zero).
9. Il controllo trigger (attributo della coda locale**TriggerControl**) per la coda dell'applicazione è impostato su MQTC_ON. Per eseguire questa operazione, impostare l'attributo **trigger** quando si definisce la coda oppure utilizzare il comando ALTER QLOCAL.
10. Il tipo di trigger (attributo coda locale**TriggerType**) non è MQTT_NONE.

    Se vengono soddisfatte tutte le condizioni richieste e il messaggio che ha causato la condizione del trigger viene inserito come parte di un'unità di lavoro, il messaggio del trigger non diventa disponibile per il richiamo da parte dell'applicazione di controllo del trigger fino a quando l'unità di lavoro non viene completata, se è stato eseguito il commit dell'unità di lavoro o, per il tipo di trigger MQTT_FIRST o MQTT_DEPTH, è stato eseguito il backout.
11. Un messaggio adatto viene inserito nella coda, per un **TriggerType** di MQTT_FIRST o MQTT_DEPTH e la coda:
    - Non era precedentemente vuoto (MQTT_FIRST) o
    - Aveva **TriggerDepth** o più messaggi (MQTT_DEPTH)

    e le condizioni da “2” a pagina 873 a “10” a pagina 875 (escluso “3” a pagina 873) sono soddisfatte, se nel caso di MQTT_FIRST è trascorso un intervallo sufficiente (attributo del gestore code**TriggerInterval**) da quando è stato scritto l'ultimo messaggio trigger per questa coda.

    Ciò consente a un server di coda di terminare prima di elaborare tutti i messaggi sulla coda. Lo scopo dell'intervallo di trigger è quello di ridurre il numero di messaggi di trigger duplicati generati.

    Nota: Se si arresta e si riavvia il gestore code, il timer **TriggerInterval** viene reimpostato. C'è una piccola finestra durante la quale è possibile produrre due messaggi trigger. La finestra esiste quando l'attributo trigger della coda è impostato su abilitato contemporaneamente all'arrivo di un messaggio e la coda non era precedentemente vuota (MQTT_FIRST) o aveva **TriggerDepth** o più messaggi (MQTT_DEPTH).
12. L'unica applicazione che serve una coda emette una chiamata MQCLOSE, per un **TriggerType** di MQTT_FIRST o MQTT_DEPTH, ed è presente almeno:
    - Uno (MQTT_FIRST) o
    - **TriggerDepth** (MQTT_DEPTH)

    i messaggi sulla coda con priorità sufficiente (condizione “2” a pagina 873) e le condizioni da “6” a pagina 874 a “10” a pagina 875 vengono soddisfatte.

    Ciò consente un server di coda che emette una chiamata MQGET, trova la coda vuota e quindi termina; tuttavia, nell'intervallo tra le chiamate MQGET e MQCLOSE, arrivano uno o più messaggi.

    Nota:

    a. Se il programma che serve la coda dell'applicazione non richiama tutti i messaggi, ciò può causare un loop chiuso. Ogni volta che il programma chiude la coda, il gestore code crea un altro messaggio trigger che fa sì che il controllo dei trigger avvii di nuovo il programma server.

b. Se il programma che serve la coda dell'applicazione esegue il backout della richiesta di richiamo (o se il programma termina in modo anomalo) prima di chiudere la coda, lo stesso accade. Tuttavia, se il programma chiude la coda prima di eseguire il backout della richiesta di richiamo e la coda è altrimenti vuota, non viene creato alcun messaggio trigger.

c. Per evitare che si verifichi un loop di questo tipo, utilizzare il campo *BackoutCount* di MQMD per rilevare i messaggi di cui viene ripetutamente eseguito il backout. Per ulteriori informazioni, consultare "Messaggi di cui è stato eseguito il backout" a pagina 47.

13. Le seguenti condizioni vengono soddisfatte utilizzando MQSET o un comando:

a. • **TriggerControl** è stato modificato in MQTC_ON oppure
   • **TriggerControl** è già MQTC_ON e il valore di **TriggerType**, **TriggerMsgPriority**o **TriggerDepth** (se pertinente) viene modificato,

   e ci sono almeno:

   • Uno (MQTT_FIRST o MQTT_EVERY) o
   • **TriggerDepth** (MQTT_DEPTH)

   messaggi sulla coda di priorità sufficiente (condizione "2" a pagina 873) e condizioni "4" a pagina 874 tramite "10" a pagina 875 (escluso "8" a pagina 875) sono soddisfatti.

   Ciò consente a un'applicazione o a un operatore di modificare i criteri di attivazione, quando le condizioni per il verificarsi di un trigger sono già soddisfatte.

b. Il valore dell'attributo della coda **InhibitPut** di una coda di iniziazione cambia da MQQA_PUT_INIBITO a MQA_PUT_ALLOWED e c'è almeno:

   • Uno (MQTT_FIRST o MQTT_EVERY) o
   • **TriggerDepth** (MQTT_DEPTH)

   messaggi di priorità sufficiente (condizione "2" a pagina 873) su una qualsiasi delle code per cui questa è la coda di iniziazione e le condizioni da "4" a pagina 874 a "10" a pagina 875 sono soddisfatte. (Viene generato un messaggio trigger per ciascuna coda che soddisfa le condizioni.)

   Ciò consente ai messaggi trigger di non essere generati a causa della condizione MQQA_PUT_INIITED sulla coda di avvio, ma questa condizione ora è stata modificata.

c. Il valore dell'attributo della coda **InhibitGet** di una coda dell'applicazione viene modificato da MQQA_GET_INIBITO a MQQA_GET_ALLOWED e vi è almeno:

   • Uno (MQTT_FIRST o MQTT_EVERY) o
   • **TriggerDepth** (MQTT_DEPTH)

   messaggi di priorità sufficiente (condizione "2" a pagina 873) sulla coda e vengono soddisfatte anche le condizioni da "4" a pagina 874 a "10" a pagina 875, escluso "5" a pagina 874.

   Ciò consente alle applicazioni di essere attivati solo quando possono richiamare i messaggi dalla coda dell'applicazione.

d. Un'applicazione di controllo dei trigger emette una chiamata MQOPEN per l'immissione da una coda di avvio e vi è almeno:

   • Uno (MQTT_FIRST o MQTT_EVERY) o
   • **TriggerDepth** (MQTT_DEPTH)

   messaggi di priorità sufficiente (condizione "2" a pagina 873) su una qualsiasi delle code dell'applicazione per cui questa è la coda di iniziazione e le condizioni da "4" a pagina 874 a "10" a pagina 875 (escluso "8" a pagina 875) sono soddisfatte e nessuna altra applicazione ha la coda di iniziazione aperta per l'input (viene generato un messaggio trigger per ciascuna coda che soddisfa le condizioni).

   Ciò consente ai messaggi in arrivo sulle code mentre non è in esecuzione il controllo dei trigger e al gestore code di riavviare e attivare i messaggi (non persistenti) che vengono persi.

14. MSGDLVSQ è impostato correttamente. Se si imposta MSGDLVSQ=FIFO, i messaggi vengono consegnati alla coda in base al primo in uscita. La priorità del messaggio viene ignorata e la priorità predefinita della coda viene assegnata al messaggio. Se **TriggerMsgPriority** è impostato su un valore superiore rispetto alla priorità predefinita della coda, non viene attivato alcun messaggio. Se **TriggerMsgPriority** è impostato su un valore uguale o inferiore alla priorità predefinita della coda, l'attivazione si verifica per il tipo FIRST, EVERY e DEPTH. Per informazioni su questi tipi, consultare la descrizione del campo **TriggerType** in "Controllo degli eventi trigger" a pagina 877.

    Se si imposta MSGDLVSQ=PRIORITY e la priorità del messaggio è maggiore o uguale al campo *TriggerMsgPriority* , i messaggi vengono conteggiati solo per un evento trigger. In questo caso, il trigger si verifica per il tipo FIRST, EVERY e DEPTH. Ad esempio, se si immettono 100 messaggi di priorità inferiore rispetto a **TriggerMsgPriority**, la profondità della coda effettiva per l'attivazione è ancora zero. Se si inserisce un altro messaggio sulla coda, ma questa volta la priorità è maggiore o uguale a **TriggerMsgPriority**, la profondità effettiva della coda aumenta da zero a uno e la condizione per **TriggerType** FIRST viene soddisfatta.

Note:

1. Dal passo "12" a pagina 875 (dove i messaggi trigger vengono generati come risultato di un evento diverso da un messaggio in arrivo sulla coda dell'applicazione), il messaggio trigger non viene inserito come parte di un'unità di lavoro. Inoltre, se **TriggerType** è MQTT_EVERY e se vi sono uno o più messaggi nella coda dell'applicazione, viene generato solo un messaggio trigger.
2. Se IBM MQ segmenta un messaggio durante MQPUT, un evento trigger non verrà elaborato fino a quando tutti i segmenti non saranno stati posizionati correttamente nella coda. Tuttavia, una volta che i segmenti di messaggi si trovano nella coda, IBM MQ li considera come singoli messaggi per l'attivazione. Ad esempio, un singolo messaggio logico suddiviso in tre parti fa sì che venga elaborato solo un evento trigger quando viene prima MQPUT e segmentato. Tuttavia, ciascuno dei tre segmenti provoca l'elaborazione dei propri eventi trigger man mano che vengono spostati nella rete IBM MQ .
3. Per IBM MQ for z/OS, se una coda condivisa è impostata per l'attivazione e la connessione alla CF (Coupling Facility) che ospita la coda condivisa viene persa, è possibile che venga generato un evento trigger e che venga inserito un messaggio nella coda di avvio. Ciò può verificarsi anche quando nessun messaggio è stato inserito nella configurazione della coda condivisa originale per l'attivazione. Ciò è causato dalla sovraindicazione dei bit da parte della macro IXLVECTR come documentato in Il vettore di notifica elenco.

Controllo degli eventi trigger

Gli eventi trigger vengono controllati utilizzando alcuni attributi che definiscono la coda dell'applicazione. Queste informazioni forniscono anche esempi di utilizzo dei tipi di trigger: EVERY, FIRST e DEPTH.

È possibile abilitare e disabilitare il trigger ed è possibile selezionare il numero o la priorità dei messaggi che contano per un evento trigger. Esiste una descrizione completa di questi attributi in Attributi degli oggetti.

Gli attributi rilevanti sono:

TriggerControl
: Utilizzare questo attributo per abilitare e disabilitare il trigger per una coda di applicazioni.

TriggerMsgPriority
: La priorità minima che un messaggio deve avere per essere conteggiato per un evento trigger. Se un messaggio con priorità inferiore a *TriggerMsgPriority* arriva sulla coda dell'applicazione, il gestore code ignora il messaggio quando determina se creare un messaggio trigger. Se *TriggerMsgPriority* è impostato su zero, tutti i messaggi vengono conteggiati per un evento trigger.

TriggerType
: Oltre al tipo di trigger NONE (che disabilita il trigger come l'impostazione di *TriggerControl* su OFF), è possibile utilizzare i seguenti tipi di trigger per impostare la sensibilità di una coda per attivare gli eventi:

Ogni
Un evento trigger si verifica ogni volta che un messaggio arriva sulla coda dell'applicazione. Utilizzare questo tipo di trigger se si desidera avviare più istanze di un'applicazione.

PRIMO
Un evento trigger si verifica solo quando il numero di messaggi sulla coda dell'applicazione cambia da zero a uno. Utilizzare questo tipo di trigger se si desidera che un programma di utilizzo venga avviato quando il primo messaggio arriva su una coda, continuare fino a quando non ci sono più messaggi da elaborare, quindi terminare. È sempre necessario elaborare la coda finché non è vuota. Consultare anche “Caso speciale del tipo di trigger FIRST” a pagina 879.

PROFOND
Un evento trigger si verifica solo quando il numero di messaggi nella coda dell'applicazione raggiunge il valore dell'attributo **TriggerDepth** . Un uso tipico di questo tipo di attivazione è quello di avviare un programma quando vengono ricevute tutte le risposte a una serie di richieste.

Attivazione per profondità: Con l'attivazione per profondità, il gestore code disabilita l'attivazione (utilizzando l'attributo *TriggerControl*) dopo aver creato un messaggio trigger. L'applicazione deve riabilitare l'attivazione stessa (utilizzando la chiamata MQSET) dopo che ciò si è verificato.

L'azione di disabilitazione del trigger non è sotto il controllo del syncpoint, quindi il trigger non può essere riabilitato ripristinando un'unità di lavoro. Se un programma esegue il backout di una richiesta put che ha causato un evento trigger o se il programma termina in modo anomalo, è necessario riabilitare il trigger utilizzando la chiamata MQSET o il comando ALTER QLOCAL.

TriggerDepth
Il numero di messaggi su una coda che causa un evento trigger quando si utilizza l'attivazione per profondità.

Le condizioni che devono essere soddisfatte per un gestore code per creare un messaggio trigger sono descritte in “Condizioni per un evento trigger” a pagina 873.

Esempio di utilizzo del tipo di trigger EVERY

Si consideri un'applicazione che genera richieste di assicurazione autoveicoli. L'applicazione potrebbe inviare messaggi di richiesta a un certo numero di compagnie di assicurazione, specificando la stessa coda di risposta ogni volta. Potrebbe impostare un trigger di tipo EVERY su questa coda di risposta in modo che ogni volta che arriva una risposta, la risposta potrebbe attivare un'istanza del server per elaborare la risposta.

Esempio di utilizzo del tipo di trigger FIRST

Considerare un'organizzazione con un numero di filiali che trasmettono i dettagli dei giorni lavorativi alla sede principale. Lo fanno tutti allo stesso tempo, alla fine della giornata lavorativa, e presso la sede centrale c'è una applicazione che elabora i dettagli da tutte le filiali. Il primo messaggio che arriva alla sede principale potrebbe causare un evento trigger che avvia questa applicazione. Questa applicazione continuerà l'elaborazione fino a quando non ci saranno più messaggi sulla sua coda.

Esempio di utilizzo del tipo di trigger DEPTH

Prendere in considerazione un'applicazione dell'agenzia di viaggi che crea una singola richiesta per confermare una prenotazione di volo, per confermare una prenotazione per una camera d'albergo, per noleggiare un'auto e per ordinare alcuni controlli dei viaggiatori. L'applicazione potrebbe separare questi elementi in quattro messaggi di richiesta, inviando ciascuno a una destinazione separata. Potrebbe impostare un trigger di tipo DEPTH sulla relativa coda di risposta (con la profondità impostata sul valore 4), in modo che venga riavviato solo quando tutte e quattro le risposte sono arrivate.

Se un altro messaggio (probabilmente proveniente da una richiesta diversa) arriva sulla coda di risposta prima dell'ultima delle quattro risposte, l'applicazione richiedente viene attivata in anticipo. Per evitare ciò, quando si utilizza il trigger DEPTH per raccogliere più risposte a una richiesta, utilizzare sempre una nuova coda di risposta per ogni richiesta.

Caso speciale del tipo di trigger FIRST

Con il tipo di trigger FIRST, se è già presente un messaggio sulla coda dell'applicazione quando arriva un altro messaggio, il gestore code in genere non crea un altro messaggio trigger.

Tuttavia, l'applicazione che serve la coda potrebbe non aprire effettivamente la coda (ad esempio, l'applicazione potrebbe terminare, probabilmente a causa di un problema di sistema). Se è stato inserito un nome applicazione non corretto nell'oggetto di definizione del processo, l'applicazione che serve la coda non raccoglierà nessuno dei messaggi. In queste situazioni, se un altro messaggio arriva sulla coda dell'applicazione, non c'è alcun server in esecuzione per elaborare questo messaggio (e qualsiasi altro messaggio sulla coda).

Per risolvere questo problema, il gestore code crea ulteriori messaggi trigger nelle seguenti circostanze:

- Se un altro messaggio arriva sulla coda dell'applicazione, ma solo se è trascorso un intervallo di tempo predefinito da quando il gestore code ha creato l'ultimo messaggio trigger per quella coda. Questo intervallo di tempo è definito nell'attributo gestore code *TriggerInterval*. Il valore predefinito è 999 999 999 millisecondi.
- Su IBM MQ for z/OS, le code dell'applicazione che denominano una coda di avvio aperta vengono sottoposte periodicamente a scansione. Se *TRIGINT* millisecondi sono trascorsi da quando è stato inviato l'ultimo messaggio trigger e la coda soddisfa le condizioni per un evento trigger e CURDEPTH è maggiore di zero, viene generato un messaggio trigger. Questo processo viene chiamato trigger di backstop.

Considerare i seguenti punti quando si decide un valore per l'intervallo di trigger da utilizzare nella propria applicazione:

- Se si imposta *TriggerInterval* su un valore basso e non c'è alcuna applicazione che serve la coda dell'applicazione, il tipo di trigger FIRST potrebbe comportarsi come tipo di trigger EVERY. Ciò dipende dalla frequenza con cui i messaggi vengono inseriti nella coda dell'applicazione, che a sua volta potrebbe dipendere da un'altra attività del sistema. Questo perché, se l'intervallo di trigger è molto piccolo, viene generato un altro messaggio di trigger ogni volta che un messaggio viene inserito nella coda dell'applicazione, anche se il tipo di trigger è FIRST, non EVERY. (Il tipo di trigger FIRST con un intervallo di trigger di zero è equivalente al tipo di trigger EVERY.)
- Su IBM MQ for z/OS se si imposta *TRIGINT* su un valore basso e non c'è alcuna applicazione che serve il trigger di tipo FIRST della coda dell'applicazione, il trigger di backstop genererà un messaggio trigger ogni volta che si verifica la scansione periodica delle code dell'applicazione che denominano le code di iniziazione aperte.
- Se viene eseguito il backout di un'unità di lavoro (consultare Messaggi di trigger e unità di lavoro) e l'intervallo di trigger è stato impostato su un valore elevato (o sul valore predefinito), viene generato un messaggio di trigger quando l'unità di lavoro viene ripristinata. Tuttavia, se l'intervallo di trigger è stato impostato su un valore basso o su zero (causando il comportamento del tipo di trigger FIRST come tipo di trigger EVERY), possono essere generati molti messaggi di trigger. Se viene eseguito il backout dell'unità di lavoro, tutti i messaggi trigger vengono ancora resi disponibili. Il numero di messaggi di trigger generati dipende dall'intervallo di trigger. Se l'intervallo di trigger è impostato su zero, viene generato il numero massimo di messaggi.

Progettazione di un'applicazione che utilizza code attivate

Si è visto come impostare e controllare l'attivazione per le applicazioni. Di seguito sono riportati alcuni consigli da considerare quando si progetta l'applicazione.

Attiva messaggi e unità di lavoro

I messaggi trigger creati a causa di eventi trigger che non fanno parte di un'unità di lavoro vengono inseriti nella coda di avvio, al di fuori di qualsiasi unità di lavoro, senza alcuna dipendenza da altri messaggi e sono immediatamente disponibili per il richiamo da parte del controllo trigger.

I messaggi trigger creati a causa di eventi trigger che fanno parte di un'unità di lavoro vengono resi disponibili sulla coda di iniziazione quando la UOW viene risolta, se è stato eseguito il commit o il backout dell'unità di lavoro

Se il gestore code non riesce a inserire un messaggio trigger in una coda di iniziazione, verrà inserito nella coda di messaggi non recapitabili (messaggio non recapito).

Nota:

1. Il gestore code conta sia i messaggi di cui è stato eseguito il commit che quelli di cui non è stato eseguito il commit quando valuta se esistono le condizioni per un evento trigger.

   Con l'attivazione di tipo FIRST o DEPTH, i messaggi di trigger vengono resi disponibili anche se viene eseguito il backout dell'unità di lavoro, in modo che un messaggio di trigger sia sempre disponibile quando vengono soddisfatte le condizioni richieste. Ad esempio, considerare una richiesta di inserimento all'interno di un'unità di lavoro per una coda attivata con il tipo di trigger FIRST. Ciò fa sì che il gestore code crei un messaggio trigger. Se si verifica un'altra richiesta di inserimento, da un'altra unità di lavoro, ciò non causa un altro evento trigger poiché il numero di messaggi nella coda dell'applicazione è ora cambiato da uno a due, il che non soddisfa le condizioni per un evento trigger. Ora, se viene eseguito il backout della prima unità di lavoro, ma viene eseguito il commit della seconda, viene comunque creato un messaggio di trigger.

   Tuttavia, ciò significa che i messaggi trigger vengono a volte creati quando le condizioni per un evento trigger non sono soddisfatte. Le applicazioni che utilizzano il trigger devono sempre essere preparate a gestire questa situazione. Si consiglia di utilizzare l'opzione di attesa con la chiamata MQGET, impostando *WaitInterval* su un valore appropriato.

   I messaggi trigger creati vengono sempre resi disponibili, indipendentemente dal fatto che l'unità di lavoro sia ripristinata o sottoposta a commit.
2. Per le code condivise locali (ossia le code condivise in un gruppo di condivisione code) il gestore code conta solo i messaggi di cui è stato eseguito il commit.

Ricezione di messaggi da una coda attivata

Quando si progettano le applicazioni che utilizzano il trigger, tenere presente che potrebbe verificarsi un ritardo tra un controllo trigger che avvia un programma e altri messaggi che diventano disponibili sulla coda dell'applicazione. Ciò può verificarsi quando il messaggio che causa l'evento trigger viene sottoposto a commit prima degli altri.

Per consentire l'arrivo dei messaggi, utilizzare sempre l'opzione di attesa quando si utilizza la chiamata MQGET per rimuovere i messaggi da una coda per cui sono impostate condizioni di trigger. Il *WaitInterval* deve essere sufficiente per consentire il tempo ragionevole più lungo tra l'inserimento di un messaggio e il commit della chiamata di inserimento. Se il messaggio arriva da un gestore code remoto, questa ora è influenzata da:

- Il numero di messaggi immessi prima del commit
- La velocità e la disponibilità del collegamento di comunicazione
- Le dimensioni dei messaggi

Per un esempio di una situazione in cui è necessario utilizzare la chiamata MQGET con l'opzione di attesa, considerare lo stesso esempio utilizzato quando si descrivono le unità di lavoro. Si trattava di una richiesta di inserimento all'interno di un'unità di lavoro per una coda attivata con trigger di tipo FIRST. Questo evento fa sì che il gestore code crei un messaggio trigger. Se si verifica un'altra richiesta di inserimento, da un'altra unità di lavoro, ciò non causa un altro evento trigger poiché il numero di messaggi sulla coda dell'applicazione non è stato modificato da zero a uno. Ora, se viene eseguito il backout della prima unità di lavoro, ma viene eseguito il commit della seconda, viene comunque creato un messaggio di trigger. Quindi, il messaggio di trigger viene creato nel momento in cui viene eseguito il backout della prima unità di lavoro. Se si verifica un ritardo significativo prima che venga eseguito il commit del secondo messaggio, l'applicazione attivata potrebbe dover attendere.

Con l'attivazione di tipo DEPTH, è possibile che si verifichi un ritardo anche se viene eseguito il commit di tutti i messaggi pertinenti. Si supponga che l'attributo della coda **TriggerDepth** abbia il valore 2. Quando due messaggi arrivano sulla coda, il secondo causa la creazione di un messaggio trigger. Tuttavia, se il secondo messaggio è il primo di cui eseguire il commit, è in quel momento che il messaggio del trigger diventa disponibile. Il controllo trigger avvia il programma del server, ma il programma può

richiamare solo il secondo messaggio fino a quando non viene eseguito il commit del primo. Quindi il programma potrebbe dover attendere che il primo messaggio sia reso disponibile.

Progettare l'applicazione in modo che termini se nessun messaggio è disponibile per il richiamo quando l'intervallo di attesa scade. Se uno o più messaggi arrivano in un secondo momento, fai affidamento sul fatto che la tua applicazione venga riattivata per elaborarli. Questo metodo impedisce alle applicazioni di essere inattive e di utilizzare inutilmente le risorse.

Elaborazione della coda di iniziazione da parte dei controlli trigger

Per un gestore code, un controllo trigger è come qualsiasi altra applicazione che serve una coda. Tuttavia, un controllo trigger serve le code di iniziazione.

Un controllo trigger è di solito un programma in esecuzione continua. Quando un messaggio trigger arriva su una coda di iniziazione, il controllo trigger richiama tale messaggio. Utilizza le informazioni nel messaggio per emettere un comando per avviare l'applicazione che deve elaborare i messaggi sulla coda dell'applicazione.

Il controllo trigger deve passare informazioni sufficienti al programma che sta avviando in modo che il programma possa eseguire le azioni corrette sulla corretta coda dell'applicazione.

Un iniziatore di canali è un esempio di un tipo speciale di controllo trigger per gli agent del canale dei messaggi. In questa situazione, tuttavia, è necessario utilizzare il tipo di trigger FIRST o DEPTH.

ALW *Controlli dei trigger sui sistemi AIX, Linux, and Windows*

Questo argomento contiene informazioni sui controlli trigger forniti sui sistemi AIX, Linux, and Windows .

I seguenti controlli trigger vengono forniti per l'ambiente del server:

amqstrg0
Questo è un controllo trigger di esempio che fornisce un sottoinsieme della funzione fornita da **runmqtrm**. Per ulteriori informazioni su amqstrg0, consultare “Utilizzo dei programmi di esempio su Multiplatforms” a pagina 1061 .

runmqtrm
La sintassi di questo comando è **runmqtrm** [*-m QMgrName*] [*-q InitQ*], dove QMgrName è il gestore code e InitQ è la coda di iniziazione. La coda predefinita è SYSTEM.DEFAULT.INITIATION.QUEUE sul gestore code predefinito. Richiama i programmi per i messaggi trigger appropriati. Questo controllo trigger supporta il tipo di applicazione predefinito.

La stringa di comando trasmessa dal controllo trigger al sistema operativo viene creata come segue:

1. Il *ApplId* dalla definizione PROCESS pertinente (se creato)
2. La struttura MQTMC2 , racchiusa tra doppi apici
3. Il *EnvData* dalla definizione PROCESS pertinente (se creato)

dove *ApplId* è il nome del programma da eseguire come verrebbe immesso sulla riga comandi.

Il parametro passato è la struttura di caratteri MQTMC2 . Viene richiamata una stringa di comando che contiene questa stringa, esattamente come fornita, tra virgolette, in modo che il comando di sistema la accetti come un parametro.

Il controllo dei trigger non controlla se è presente un altro messaggio sulla coda di iniziazione fino al completamento dell'applicazione appena avviata. Se l'applicazione ha molta elaborazione da fare, il controllo trigger potrebbe non essere in grado di tenere il passo con il numero di messaggi trigger in arrivo. Hai due opzioni:

- Avere più controlli trigger in esecuzione
- Eseguire le applicazioni avviate in background

Se si dispone di più controlli dei trigger in esecuzione, è possibile controllare il numero massimo di applicazioni che possono essere eseguite contemporaneamente. Se si eseguono le applicazioni in

background, non vi è alcuna limitazione imposta da IBM MQ sul numero di applicazioni che possono essere eseguite.

Linux AIX Per eseguire l'applicazione avviata in background su AIX and Linux, inserire un & alla fine del *EnvData* della definizione PROCESS.

Per eseguire l'applicazione avviata in background su sistemi Windows , all'interno del campo *ApplId* , anteporre un comando START al nome dell'applicazione. Ad esempio:

```
START ?B AMQSECHA
```

Nota: Windows Quando un percorso Windows contiene spazi come parte del nome percorso, questi devono essere racchiusi tra virgolette (") per assicurarsi che sia gestito come un singolo argomento. Ad esempio, "`C:\Program Files\Application Directory\Application.exe`".

Di seguito viene riportato un esempio di stringa APPLICID in cui il nome file include spazi come parte del percorso:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

La sintassi del comando Windows START nell'esempio include una stringa vuota racchiusa tra virgolette doppie. START specifica che il primo argomento tra virgolette verrà considerato come titolo del nuovo comando. Per assicurarsi che Windows non sbagli il percorso dell'applicazione per un argomento 'title', aggiungere una stringa del titolo racchiusa tra virgolette al comando prima del nome dell'applicazione.

I seguenti controlli trigger vengono forniti per il client IBM MQ :

runmqtmc
È uguale a runmqtrm tranne che si collega alle librerie IBM MQ MQI client .

ALW *Controllo dei trigger per CICS*

Il controllo dei trigger amqltmc0 viene fornito per CICS. Funziona allo stesso modo del controllo dei trigger standard, runmqtrm, ma viene eseguito in modo diverso e vengono attivate transazioni CICS .

Questo argomento si applica solo a sistemi Windows, AIX and Linux x86-64 .

Il controllo dei trigger viene fornito come programma CICS ; definirlo con un nome transazione di 4 caratteri. Immettere il nome di 4 caratteri per avviare il controllo trigger. Utilizza il gestore code predefinito (come indicato nel file qm.ini o, su IBM MQ for Windows, nel registro) e il SISTEMA SYSTEM.CICS.INITIATION.QUEUE.

Se si desidera utilizzare un gestore code o una coda differenti, creare la struttura del controllo dei trigger MQTMC2 : ciò richiede la scrittura di un programma utilizzando la chiamata EXEC CICS START, poiché la struttura è troppo lunga per essere aggiunta come parametro. Quindi, passare la struttura MQTMC2 come dati alla richiesta START per il controllo trigger.

Quando si utilizza la struttura MQTMC2 , è necessario fornire solo i parametri *StrucId*, *Version*, *QName*e **QMgrName** al controllo trigger poiché non fa riferimento ad altri campi.

I messaggi vengono letti dalla coda di iniziazione e utilizzati per avviare le transazioni CICS , utilizzando EXEC CICS START, supponendo che APPL_TYPE nel messaggio del trigger sia MQAT_CICS. La lettura dei messaggi dalla coda di avvio viene eseguita sotto il controllo del punto di sincronizzazione CICS .

I messaggi vengono generati quando il monitoraggio viene avviato e arrestato e quando si verificano errori. Questi messaggi vengono inviati alla coda di dati temporanei CSMT.

Tabella 129. Versioni disponibili del controllo trigger.

Una tabella con due colonne. la prima colonna elenca le versioni disponibili del controllo trigger e la seconda colonna mostra per quali piattaforme viene utilizzata ciascuna versione.

Versione	**Utilizzo**
amqltmc0	TXSeries Per: • AIX • Linux Sistemi Linux x86-64
amqltmc4	Windows TXSeries per Windows 5.1
amqltmcc	Versione di collegamento client del controllo trigger CICS

Se è necessario un controllo trigger per altri ambienti, scrivere un programma che possa elaborare i messaggi trigger che il gestore code inserisce nelle code di iniziazione. Tale programma deve eseguire le azioni riportate di seguito:

1. Utilizzare la chiamata MQGET per attendere l'arrivo di un messaggio sulla coda di avvio.
2. Esaminare i campi della struttura di MQTM del messaggio trigger per individuare il nome dell'applicazione da avviare e l'ambiente in cui viene eseguito.
3. Immettere un comando di avvio specifico dell'ambiente.

   z/OS Ad esempio, su un batch z/OS , inoltrare un lavoro al reader interno.
4. Convertire la struttura MQTM nella struttura MQTMC2 , se necessario.
5. Passare la struttura MQTMC2 o MQTM all'applicazione avviata. Può contenere dati utente.
6. Associare alla coda dell'applicazione l'applicazione che deve servire tale coda. A tal fine, denominare l'oggetto di definizione del processo (se creato) nell'attributo **ProcessName** della coda. Per denominare l'oggetto definizione processo, è possibile utilizzare il comando **DEFINE QLOCAL** o **ALTER QLOCAL** .

   IBM i Su IBM i, è anche possibile utilizzare CRTMQ o CHGMQMQ.

Per ulteriori informazioni sull'interfaccia di controllo del trigger, consultare MQTMC2.

IBM i *Controlli trigger su IBM i*

Su IBM i, invece del comando di controllo **runmqtrm** , utilizzare il IBM MQ for IBM i comando CL **STRMQMTRM**.

Utilizzare il comando STRMQMTRM nel modo seguente:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

I dettagli sono come per runmqtrm.

Vengono inoltre forniti i seguenti programmi di esempio, che è possibile utilizzare come modelli per scrivere i propri controlli trigger:

AMQSTRG4
Si tratta di un controllo trigger che inoltra un lavoro IBM i per il processo che deve essere avviato, ma ciò significa che vi è un'ulteriore elaborazione associata a ciascun messaggio trigger.

AMQSERV4
Questo è un server trigger. Per ogni messaggio di trigger, questo server esegue il comando per il processo nel proprio lavoro e può richiamare le transazioni CICS .

Sia il controllo dei trigger che il server dei trigger passano una struttura MQTMC2 ai programmi che avviano. Per una descrizione di questa struttura, consultare MQTMC2. Entrambi questi esempi vengono forniti sia in formato sorgente che in formato eseguibile.

Poiché questi controlli trigger possono richiamare solo programmi IBM i nativi, non possono attivare direttamente i programmi Java , poiché le classi Java si trovano in IFS. Tuttavia, i programmi Java possono essere attivati indirettamente attivando un programma CL che richiama il programma Java e passa attraverso la struttura TMC2 . La dimensione minima della struttura TMC2 è 732 byte.

Di seguito è riportata l'origine di un CLP di esempio:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

Il seguente programma di controllo dei trigger viene fornito per IBM MQ MQI client: RUNMQTMC

Richiamare RUNMQTMC nel modo seguente:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

Proprietà dei messaggi trigger

I seguenti argomenti descrivono alcune altre proprietà dei messaggi trigger.

- “Persistenza e priorità dei messaggi trigger” a pagina 884
- “Messaggi di trigger e riavvio del gestore code” a pagina 884
- “Attiva messaggi e modifiche agli attributi dell'oggetto” a pagina 884
- “Formato dei messaggi di trigger” a pagina 885

Persistenza e priorità dei messaggi trigger

I messaggi di trigger non sono persistenti perché non è necessario che lo siano.

Tuttavia, le condizioni per la generazione di eventi di attivazione persistono, quindi i messaggi di trigger vengono generati ogni volta che queste condizioni vengono soddisfatte. Se un messaggio di trigger viene perso, l'esistenza continua del messaggio dell'applicazione sulla coda dell'applicazione garantisce che il gestore code genera un messaggio di trigger non appena vengono soddisfatte tutte le condizioni.

Se viene eseguito il rollback di un'unità di lavoro, tutti i messaggi di trigger generati vengono sempre consegnati.

I messaggi trigger assumono la priorità predefinita della coda di avvio.

Messaggi di trigger e riavvio del gestore code

In seguito al riavvio di un gestore code, quando una coda di iniziazione viene successivamente aperta per l'input, è possibile inserire un messaggio trigger in questa coda di iniziazione se una coda di applicazione ad essa associata contiene messaggi ed è definita per l'attivazione.

Attiva messaggi e modifiche agli attributi dell'oggetto

I messaggi trigger vengono creati in base ai valori degli attributi trigger in vigore al momento dell'evento trigger.

Se il messaggio trigger non viene reso disponibile per un controllo trigger fino a un momento successivo (poiché il messaggio che ne ha causato la creazione è stato inserito all'interno di un'unità di lavoro), qualsiasi modifica agli attributi trigger nel frattempo non ha alcun effetto sul messaggio trigger. In particolare, la disabilitazione del trigger non impedisce che un messaggio trigger venga reso disponibile

una volta creato. Inoltre, la coda dell'applicazione potrebbe non esistere più nel momento in cui il messaggio trigger viene reso disponibile.

Formato dei messaggi di trigger

Il formato di un messaggio trigger è definito dalla struttura MQTM.

Sono presenti i seguenti campi, che il gestore code compila quando crea il messaggio trigger, utilizzando le informazioni nelle definizioni di oggetto della coda dell'applicazione e del processo associato a tale coda:

StrucId
L'identificativo della struttura.

Version
La versione della struttura.

QName
Il nome della coda dell'applicazione in cui si è verificato l'evento trigger. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **QName** della coda dell'applicazione.

ProcessName
Il nome dell'oggetto di definizione processo associato alla coda dell'applicazione. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **ProcessName** della coda dell'applicazione.

TriggerData
Un campo a formato libero per l'utilizzo da parte del controllo trigger. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **TriggerData** della coda dell'applicazione. Su qualsiasi prodotto IBM MQ tranne IBM MQ for z/OS, questo campo può essere utilizzato per specificare il nome del canale da attivare.

ApplType
Il tipo di applicazione che il controllo trigger deve avviare. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **ApplType** dell'oggetto definizione processo identificato in *ProcessName*.

ApplId
Una stringa di caratteri che identifica l'applicazione che il controllo trigger deve avviare. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **ApplId** dell'oggetto definizione processo identificato in *ProcessName*.

Quando si utilizza il controllo trigger CKTI, fornito da CICS, l'attributo **ApplId** dell'oggetto di definizione del processo è un identificativo della transazione CICS .

Quando si utilizza CSQQTRMN fornito da IBM MQ for z/OS, l'attributo **ApplId** dell'oggetto definizione processo è un identificativo della transazione IMS .

EnvData
Un campo di caratteri che contiene i dati relativi all'ambiente per l'utilizzo da parte del controllo trigger. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **EnvData** dell'oggetto definizione processo identificato in *ProcessName*. Il controllo trigger fornito da CICS(CKTI) o il controllo trigger fornito da IBM MQ for z/OS(CSQQTRMN) non utilizzano questo campo, ma altri controlli trigger potrebbero scegliere di utilizzarlo.

UserData
Un campo di caratteri contenente i dati utente che possono essere utilizzati dal controllo trigger. Quando il gestore code crea un messaggio trigger, riempie questo campo utilizzando l'attributo **UserData** dell'oggetto definizione processo identificato in *ProcessName*. Questo campo può essere utilizzato per specificare il nome del canale da attivare.

È disponibile una descrizione completa della struttura del messaggio trigger in MQTM.

Quando l'attivazione non funziona

Un programma non viene attivato se il controllo trigger non può avviare il programma o il gestore code non può consegnare il messaggio trigger. Ad esempio, l'applid nell'oggetto processo deve specificare che il programma deve essere avviato in background; altrimenti, il controllo trigger non può avviare il programma.

Se un messaggio trigger viene creato ma non può essere inserito nella coda di iniziazione (ad esempio, perché la coda è piena o la lunghezza del messaggio trigger è maggiore della lunghezza massima del messaggio specificata per la coda di iniziazione), il messaggio trigger viene inserito nella coda di messaggi non recapitabili (messaggio non consegnato).

Se l'operazione di inserimento nella coda di messaggi non instradabili non può essere completata correttamente, il messaggio di trigger viene eliminato e un messaggio di avviso viene inviato z/OS alla console z/OS o all'operatore di sistema oppure viene inserito nella registrazione degli errori.

L'inserimento del messaggio trigger nella coda di messaggi non instradabili potrebbe generare un messaggio trigger per tale coda. Questo secondo messaggio di trigger viene eliminato se aggiunge un messaggio alla coda di messaggi non recapitabili.

Se il programma viene attivato correttamente ma si interrompe prima di ricevere il messaggio dalla coda, utilizzare un programma di utilità di traccia (ad esempio, CICS AUXTRACE se il programma è in esecuzione in CICS) per individuare la causa dell'errore.

Utilizzo di MQI e cluster

Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

Utilizzare i seguenti link per ulteriori informazioni sulle opzioni disponibili sulle chiamate e sui codici di ritorno da utilizzare con i cluster:

- “MQOPEN e cluster” a pagina 887
- “MQPUT, MQPUT1 e cluster” a pagina 888
- “MQINQ e cluster” a pagina 888
- “MQSET e cluster” a pagina 889
- “Codici di ritorno” a pagina 889

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

MQOPEN e cluster

La coda in cui viene inserito un messaggio o da cui viene letto, quando viene aperta una coda cluster, dipende dalla chiamata MQOPEN .

Selezione della coda di destinazione

Se non si fornisce un nome gestore code nel descrittore dell'oggetto, MQOD, il gestore code seleziona il gestore code a cui inviare il messaggio. Se si fornisce un nome gestore code nel descrittore dell'oggetto, i messaggi vengono sempre inviati al gestore code selezionato.

Se il gestore code sta selezionando il gestore code di destinazione, la selezione dipende dalle opzioni di bind MQOO_BIND_* e se esiste una coda locale. Se è presente un'istanza locale della coda, questa viene sempre aperta in preferenza a un'istanza remota, a meno che l'attributo CLWLUSEQ non sia impostato su ANY. Altrimenti, la selezione dipende dalle opzioni di collegamento. È necessario specificare MQOO_BIND_ON_OPEN o MQOO_BIND_ON_GROUP quando si utilizzano gruppi di messaggi con cluster per garantire che tutti i messaggi nel gruppo vengano elaborati nella stessa destinazione.

Se il gestore code sta selezionando il gestore code di destinazione, lo fa in modo round - robin, utilizzando l'algoritmo di gestione del carico di lavoro; consultare Bilanciamento del carico di lavoro nei cluster.

Quando viene utilizzato l'algoritmo di bilanciamento del carico di lavoro dipende dal modo in cui viene aperta la coda cluster:

- MQOO_BIND_ON_OPEN - l'algoritmo viene utilizzato una volta al momento dell'apertura della coda da parte dell'applicazione.
- MQOO_BIND_NOT_FIXED - l'algoritmo viene utilizzato per ogni messaggio inserito nella coda.
- MQOO_BIND_ON_GROUP - l'algoritmo viene utilizzato una volta all'inizio di ciascun gruppo di messaggi.

MQOO_BIND_ON_OPEN
L'opzione MQOO_BIND_ON_OPEN sulla chiamata MQOPEN specifica che il gestore code di destinazione deve essere corretto. Utilizzare l'opzione MQOO_BIND_ON_OPEN se vi sono più istanze della stessa coda all'interno di un cluster. Tutti i messaggi immessi nella coda specificando l'handle dell'oggetto restituito dalla chiamata MQOPEN vengono indirizzati allo stesso gestore code.

- Utilizzare l'opzione MQOO_BIND_ON_OPEN se i messaggi hanno affinità. Ad esempio, se un batch di messaggi deve essere tutti elaborato dallo stesso gestore code, specificare MQOO_BIND_ON_OPEN quando si apre la coda. IBM MQ corregge il gestore code e l'instradamento che deve essere utilizzato da tutti i messaggi inseriti in tale coda.
- Se viene specificata l'opzione MQOO_BIND_ON_OPEN , la coda deve essere riaperta per poter selezionare una nuova istanza della coda.

MQOO_BIND_NOT_FIXED
L'opzione MQOO_BIND_NOT_FIXED nella chiamata MQOPEN specifica che il gestore code di destinazione non è fisso. I messaggi scritti nella coda che specificano l'handle dell'oggetto restituito dalla chiamata MQOPEN vengono instradati a un gestore code MQPUT in base al messaggio. Utilizzare l'opzione MQOO_BIND_NOT_FIXED se non si desidera forzare la scrittura di tutti i messaggi nella stessa destinazione.

- Non specificare MQOO_BIND_NOT_FIXED e MQMF_SEGMENTATION_ALLOWED contemporaneamente. In tal caso, i segmenti del messaggio potrebbero essere consegnati a gestori code differenti, sparsi in tutto il cluster.

MQOO_BIND_ON_GROUP
Consente a una applicazione di richiedere che un gruppo di messaggi sia assegnato alla stessa istanza di destinazione. Questa opzione è valida solo per le code e interessa solo le code cluster. Se specificata per una coda che non è una coda cluster, l'opzione viene ignorata.

- I gruppi vengono instradati solo a una destinazione singola quando MQPMO_LOGICAL_ORDER viene specificato su MQPUT. Quando si specifica MQOO_BIND_ON_GROUP, ma un messaggio non fa parte di un gruppo logico, viene utilizzato il comportamento BIND_NOT_FIXED.

MQOO_BIND_AS_Q_DEF
Se non si specifica MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED o MQOO_BIND_ON_GROUP, l'opzione predefinita è MQOO_BIND_AS_Q_DEF. L'utilizzo di MQOO_BIND_AS_Q_DEF fa in modo che il bind utilizzato per l'handle della coda venga preso dall'attributo della coda DefBind .

Pertinenza delle opzioni MQOPEN

Le MQOPEN opzioni MQOO_BROWSE, MQOO_INPUT_*o MQOO_SET richiedono un'istanza locale della coda del cluster perché MQOPEN abbia esito positivo.

Le MQOPEN opzioni MQOO_OUTPUT, MQOO_BIND_*o MQOO_INQUIRE non richiedono un'istanza locale della coda del cluster per avere esito positivo.

Nome del gestore code risolto

Quando un nome gestore code viene risolto in MQOPEN , il nome risolto viene restituito all'applicazione. Se l'applicazione tenta di utilizzare questo nome in una chiamata MQOPEN successiva, potrebbe scoprire che non è autorizzata ad accedere al nome.

MQPUT, MQPUT1 e cluster

Se MQOO_BIND_NOT_FIXED è specificato su MQOPEN le routine di gestione del carico di lavoro scelgono quale destinazione MQPUT o MQPUT1 selezionare.

Se MQOO_BIND_NOT_FIXED viene specificato su una chiamata MQOPEN , ogni chiamata MQPUT successiva richiama la routine di gestione del carico di lavoro per stabilire a quale gestore code inviare il messaggio. La destinazione e l'instradamento da prendere vengono selezionati messaggio per messaggio. La destinazione e l'instradamento potrebbero cambiare dopo che il messaggio è stato inserito se cambiano le condizioni nella rete. La chiamata MQPUT1 funziona sempre come se MQOO_BIND_NOT_FIXED fosse attiva, ossia richiama sempre la routine di gestione del carico di lavoro.

Quando la routine di gestione del carico di lavoro ha selezionato un gestore code, il gestore code locale completa l'operazione di inserimento. Il messaggio può essere inserito in code differenti:

1. Se la destinazione è l'istanza locale della coda, il messaggio viene inserito nella coda locale.
2. Se la destinazione è un gestore code in un cluster, il messaggio viene inserito in una coda di trasmissione cluster.
3. Se la destinazione è un gestore code esterno a un cluster, il messaggio viene inserito in una coda di trasmissione con lo stesso nome del gestore code di destinazione.

Se MQOO_BIND_ON_OPEN è specificato nella chiamata MQOPEN , le chiamate MQPUT non richiamano la routine di gestione del workload perché la destinazione e l'instradamento sono già stati selezionati.

MQINQ e cluster

La coda cluster da interrogare dipende dalle opzioni che si combinano con MQOO_INQUIRE.

Prima di poter analizzare una coda, aprirla utilizzando la chiamata MQOPEN e specificare MQOO_INQUIRE.

Per analizzare una coda cluster, utilizzare la chiamata MQOPEN e combinare altre opzioni con MQOO_INQUIRE. Gli attributi che possono essere interrogati dipendono dalla presenza di un'istanza locale della coda del cluster e dalla modalità di apertura della coda:

- La combinazione di MQOO_BROWSE, MQOO_INPUT_*o MQOO_SET con MQOO_INQUIRE richiede un'istanza locale della coda del cluster perché l'apertura abbia esito positivo. In questo caso, è possibile esaminare tutti gli attributi validi per le code locali.
- Combinando MQOO_OUTPUT con MQOO_INQUIREe non specificando alcuna delle opzioni precedenti, l'istanza aperta è:
  - L'istanza sul gestore code locale, se presente. In questo caso, è possibile esaminare tutti gli attributi validi per le code locali.
  - Un'istanza altrove nel cluster, se non è presente alcuna istanza del gestore code locale. In questo caso è possibile interrogare solo i seguenti attributi. In questo caso l'attributo QType ha il valore MQQT_CLUSTER .
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QTYPE

Per analizzare l'attributo DefBind di una coda cluster, utilizzare la chiamata MQINQ con il selettore MQIA_DEF_BIND. Il valore restituito è MQBND_BIND_ON_OPEN o MQBND_BIND_NOT_FIXEDo MQBND_BIND_ON_GROUP. È necessario specificare MQBND_BIND_ON_OPEN o MQBND_BIND_ON_GROUP quando si utilizzano gruppi con cluster.

Per analizzare gli attributi CLUSTER e CLUSNL dell'istanza locale di una coda, utilizzare la chiamata MQINQ con il selettore MQCA_CLUSTER_NAME o il selettore MQCA_CLUSTER_NAMELIST.

Nota: Se si apre una coda cluster senza correggere la coda a cui MQOPEN ha eseguito il bind, le successive chiamate MQINQ potrebbero interrogare diverse istanze della coda cluster.

Concetti correlati

“Opzione MQOPEN per la coda cluster” a pagina 749

Il bind utilizzato per l'handle della coda viene preso dall'attributo della coda **DefBind** , che può assumere il valore MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDo MQBND_BIND_ON_GROUP.

MQSET e cluster

L'opzione MQOPEN option MQOO_SET richiede che ci sia un'istanza locale di una coda cluster perché MQSET abbia esito positivo.

Non è possibile utilizzare la chiamata MQSET per impostare gli attributi di una coda altrove nel cluster.

È possibile aprire un alias locale o una coda remota definita con l'attributo cluster e utilizzare la chiamata MQSET . È possibile impostare gli attributi dell'alias locale o della coda remota. Non importa se la coda di destinazione è una coda cluster definita su un gestore code differente.

Codici di ritorno

Codici di ritorno specifici per i cluster

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

Viene emessa una chiamata MQOPEN, MQPUTo MQPUT1 per aprire una coda cluster o inserire un messaggio su di essa. L'uscita del carico di lavoro cluster, definita dall'attributo ClusterWorkloadExit di un gestore code, ha esito negativo in modo imprevisto o non risponde in tempo.

Viene scritto un messaggio nel log di sistema su IBM MQ for z/OS che fornisce ulteriori informazioni su questo errore.

Le successive chiamate MQOPEN, MQPUTe MQPUT1 per questo handle di coda vengono elaborate come se l'attributo ClusterWorkloadExit fosse vuoto.

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

Su z/OS, non è possibile caricare l'uscita del carico di lavoro del cluster.

Un messaggio viene scritto nel log di sistema e l'elaborazione continua come se l'attributo `ClusterWorkloadExit` fosse vuoto.

Multi Su Multiplatforms, viene emessa una chiamata MQCONN o MQCONNX per connettersi a un gestore code. La chiamata ha esito negativo perché non è possibile caricare l'uscita del workload del cluster, definita dall'attributo `ClusterWorkloadExit` del gestore code.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Viene emessa una chiamata MQOPEN con le opzioni MQOO_OUTPUT e MQOO_BIND_ON_OPEN attive per una coda cluster. Tutte le istanze della coda nel cluster sono attualmente inibite dall'inserimento, poiché l'attributo `InhibitPut` è impostato su MQQA_PUT_INHIBITED. Poiché non ci sono istanze di coda disponibili per ricevere messaggi, la chiamata MQOPEN ha esito negativo.

Questo codice di errore si verifica solo quando entrambe le seguenti istruzioni sono vere:

- Non esiste alcuna istanza locale della coda. Se è presente un'istanza locale, la chiamata MQOPEN ha esito positivo, anche se l'istanza locale è di tipo put - inibito.
- Non esiste alcuna uscita del carico di lavoro del cluster per la coda oppure esiste un'uscita del carico di lavoro del cluster ma non sceglie un'istanza della coda. (Se l'uscita del carico di lavoro del cluster sceglie un'istanza della coda, la chiamata MQOPEN ha esito positivo, anche se tale istanza è inibita dall'inserimento.)

Se l'opzione MQOO_BIND_NOT_FIXED viene specificata nella chiamata MQOPEN , la chiamata può avere esito positivo anche se tutte le code nel cluster sono bloccate dall'inserimento. Tuttavia, una chiamata MQPUT successiva potrebbe non riuscire se tutte le code sono ancora immesse al momento di tale chiamata.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Viene emessa una chiamata MQOPEN, MQPUTo MQPUT1 per aprire una coda cluster o inserire un messaggio su di essa. La definizione della coda non può essere risolta correttamente perché è richiesta una risposta dal gestore code del repository completo, ma non ne è disponibile alcuna.
2. Viene emessa una chiamata MQOPEN, MQPUT, MQPUT1 o MQSUB per un oggetto argomento che specifica PUBSCOPE (ALL) o SUBSCOPE (ALL). La definizione dell'argomento del cluster non può essere risolta correttamente perché è richiesta una risposta dal gestore code del repository completo, ma non è disponibile.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Viene emessa una chiamata MQOPEN, MQPUTo MQPUT1 per una coda cluster. Si è verificato un errore durante il tentativo di utilizzare una risorsa richiesta per il clustering.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Viene emessa una chiamata MQPUT o MQPUT1 per inserire un messaggio in una coda cluster. Al momento della chiamata, non ci sono più istanze della coda nel cluster. MQPUT ha esito negativo e il messaggio non viene inviato.

L'errore può verificarsi se MQOO_BIND_NOT_FIXED viene specificata sulla chiamata MQOPEN che apre la coda oppure se viene utilizzato MQPUT1 per inserire il messaggio.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Viene emessa una chiamata MQOPEN, MQPUTo MQPUT1 per aprire o inserire un messaggio in una coda cluster. L'uscita del carico di lavoro del cluster rifiuta la chiamata.

z/OS Utilizzo e scrittura di applicazioni su IBM MQ for z/OS

Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

Queste informazioni descrivono le funzioni IBM MQ disponibili per i programmi in esecuzione in ciascuno degli ambienti supportati. Inoltre,

- Per informazioni sull'utilizzo di IBM MQ-CICS bridge, consultare Utilizzo di IBM MQ con CICS .
- Per informazioni sull'utilizzo di IMS e del bridge IMS , vedere “Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71.

Utilizzare i seguenti collegamenti per ulteriori informazioni sull'utilizzo e la scrittura di applicazioni su IBM MQ for z/OS:

- “Funzioni IBM MQ for z/OS dipendenti dall'ambiente” a pagina 891
- “Funzioni di debug, supporto syncpoint e supporto di ripristino” a pagina 892
- “L'interfaccia IBM MQ for z/OS con l'ambiente dell'applicazione” a pagina 893
- “Scrittura di applicazioni z/OS UNIX System Services” a pagina 894
- “Programmazione di applicazioni con code condivise” a pagina 898

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722
Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Applicazioni bridge IMS e IMS su IBM MQ for z/OS” a pagina 71
Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

z/OS *Funzioni IBM MQ for z/OS dipendenti dall'ambiente*

Utilizzare queste informazioni quando si considerano le funzioni IBM MQ for z/OS .

Le principali differenze da considerare tra le funzioni IBM MQ negli ambienti in cui viene eseguito IBM MQ for z/OS sono:

- IBM MQ for z/OS fornisce i seguenti controlli trigger:
  - CKTI per l'utilizzo nell'ambiente CICS
  - CSQQTRMN da utilizzare nell'ambiente IMS

  È necessario scrivere il proprio modulo per avviare le applicazioni in altri ambienti.

- La sincronizzazione mediante il commit a due fasi è supportata negli ambienti CICS e IMS . È inoltre supportato nell'ambiente batch z/OS utilizzando la gestione delle transazioni e RRS (recoverable resource manager services). Il commit a fase singola è supportato nell'ambiente z/OS da IBM MQ .
- Per gli ambienti batch e IMS , MQI fornisce chiamate per connettere programmi e per disconnetterli da un gestore code. I programmi possono connettersi a più di un gestore code.
- Un sistema CICS può connettersi a un solo gestore code. Ciò può verificarsi quando CICS viene avviato se il nome del sistema secondario è definito nel lavoro di avvio del sistema CICS . Le chiamate di connessione e disconnessione MQI sono tollerate, ma non hanno alcun effetto, nell'ambiente CICS .
- L'uscita API - crossing consente a un programma di intervenire nell'elaborazione di tutte le chiamate MQI. Questa uscita è disponibile solo nell'ambiente CICS .
- In CICS sui sistemi multiprocessore, si ottiene un vantaggio in termini di prestazioni poiché le chiamate MQI possono essere eseguite in più TCB z/OS . Per ulteriori informazioni, consultare Pianificazione dei messaggi z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

Queste funzioni sono riepilogate in Tabella 130 a pagina 892.

Tabella 130. Funzioni dell'ambiente z/OS

	CICS	**IMS**	**Batch / TSO**
Controllo trigger fornito	Sì	Sì	No
commit a due fasi	Sì	Sì	Sì
commit a singola fase	Sì	No	Sì
Connetti / disconnetti chiamate MQI	Tollerato	Sì	Sì
uscita incrociata API	Sì	No	No

Nota: Il commit a due fasi è supportato nell'ambiente Batch/TSO utilizzando RRS.

z/OS *Funzioni di debug, supporto syncpoint e supporto di ripristino*

Utilizzare queste informazioni per acquisire informazioni sulle funzioni di debug del programma, sul supporto del punto di sincronizzazione e sul supporto di ripristino.

Funzioni di debug del programma

IBM MQ for z/OS fornisce una funzione di traccia che è possibile utilizzare per eseguire il debug dei programmi in tutti gli ambienti.

Inoltre, nell'ambiente CICS è possibile utilizzare:

- CDF (CICS Execution Diagnostics Facility)
- CETR (CICS Trace Control Transaction)
- L'uscita incrociata API IBM MQ for z/OS

Sulla piattaforma z/OS , è possibile utilizzare qualsiasi strumento di debug interattivo disponibile supportato dal linguaggio di programmazione che si sta utilizzando.

Supporto punto di sincronizzazione

La sincronizzazione dell'inizio e della fine delle unità di lavoro è necessaria in un ambiente di elaborazione delle transazioni in modo che l'elaborazione delle transazioni possa essere utilizzata in modo sicuro.

È completamente supportato da IBM MQ for z/OS negli ambienti CICS e IMS . Il supporto completo significa cooperazione tra i gestori risorse in modo che le unità di lavoro possano essere sottoposte a commit o a backout all'unisono, sotto il controllo di CICS o IMS. Esempi di gestori risorse sono Db2, CICS File Control IMSe IBM MQ for z/OS.

Le applicazioni batch z/OS possono utilizzare le chiamate IBM MQ for z/OS per fornire una funzione di commit a fase singola. Ciò significa che è possibile eseguire il commit o il backout di una serie di operazioni della coda definita dall'applicazione senza fare riferimento ad altri gestori risorse.

Il commit a due fasi è supportato anche in ambiente batch z/OS utilizzando la gestione delle transazioni e RRS (recoverable resource manager services). Per ulteriori informazioni, consultare Syncpoints in z/OS batch applications.

Supporto di ripristino

Se la connessione tra un gestore code e un sistema CICS o IMS viene interrotta durante una transazione, alcune unità di lavoro potrebbero non essere ripristinate correttamente.

Tuttavia, queste unità di lavoro vengono risolte dal gestore code (sotto il controllo del gestore del punto di sincronizzazione) quando la connessione con il sistema CICS o IMS viene ristabilita.

z/OS *L'interfaccia IBM MQ for z/OS con l'ambiente dell'applicazione*

Per consentire alle applicazioni in esecuzione in ambienti differenti di inviare e ricevere messaggi attraverso una rete di accodamento messaggi, IBM MQ for z/OS fornisce un *adattatore* per ciascuno degli ambienti supportati.

Questi adattatori sono l'interfaccia tra programmi applicativi e sottosistemi di IBM MQ for z/OS . Consentono ai programmi di utilizzare l'MQI.

z/OS *L'adattatore batch*

Utilizzare queste informazioni per informazioni sull'adattatore batch e sul protocollo di commit supportato.

L' *adattatore batch* fornisce l'accesso alle risorse IBM MQ for z/OS per i programmi in esecuzione in:

- Modalità attività (TCB)
- Stato del problema o del supervisore
- Modalità di controllo spazio di indirizzo principale

I programmi non devono essere in modalità memoria incrociata.

Le connessioni tra i programmi applicativi e IBM MQ for z/OS sono a livello di attività. L'adattatore fornisce un singolo thread di connessione da un TCB (task control block) dell'applicazione a IBM MQ for z/OS.

L'adattatore supporta un protocollo di commit a fase singola per le modifiche apportate alle risorse di IBM MQ for z/OS ; non supporta i protocolli di commit a più fasi.

z/OS *L'adattatore batch RRS*

Utilizzare queste informazioni per informazioni sull'adattatore batch RRS e sui due adattatori batch RRS forniti da IBM MQ.

La gestione delle transazioni e l'adattatore RRS (recoverable resource manager services):

- Utilizza z/OS RRS per il controllo del commit.
- Supporta connessioni simultanee a più sottosistemi IBM MQ gestiti su una singola istanza z/OS da una singola attività.
- Fornisce il controllo del commit coordinato a livello di z/OS(utilizzando z/OS RRS) per le risorse recuperabili a cui si accede tramite z/OS RRS - compliant recoverable managers per:
  - Applicazioni che si collegano a IBM MQ utilizzando l'adattatore batch RRS.
  - Db2-stored procedure in esecuzione in uno spazio di indirizzo Db2-stored procedure gestito da un WLM (workload manager) su z/OS.
- Supporta la possibilità di commutare un thread batch IBM MQ tra TCB.

IBM MQ for z/OS fornisce due adattatori batch RRS:

CSQBRSTB
Questo adattatore richiede di modificare qualsiasi istruzione MQCMIT in SRRCMIT e qualsiasi istruzione MQBACK in SRRBACK nell'applicazione IBM MQ . (Se si codifica MQCMIT o MQBACK in un'applicazione collegata a CSQBRSTB, si riceve MQRC_ENVIRONMENT_ERROR.)

CSQBRRSI
Questo adattatore consente all'applicazione IBM MQ di utilizzare MQCMIT e MQBACK o SRRCMIT e SRRBACK.

Nota: CSQBRSTB e CSQBRRSI vengono forniti con attributi di collegamento AMODE (31) RMODE (ANY). Se l'applicazione carica uno stub al di sotto della linea di 16 MB, ricollegare lo stub con RMODE (24).

Migrazione

È possibile migrare le applicazioni Batch/TSO IBM MQ esistenti per utilizzare il coordinamento RRS con poche o nessuna modifica.

Se si collega - modifica l'applicazione IBM MQ con l'adattatore CSQBRRSI, MQCMIT e MQBACK syncpoint dell'unità di lavoro in IBM MQ e in tutti gli altri gestori risorse abilitati RRS. Se si collega - modifica l'applicazione IBM MQ con l'adattatore CSQBRSTB, modificare MQCMIT in SRRCMIT e MQBACK in SRRBACK. Quest' ultimo approccio è preferibile; indica chiaramente che il punto di sincronizzazione non è limitato solo alle risorse IBM MQ .

z/OS *L'adattatore IMS*

Se si utilizza l'adattatore IMS da un sistema IBM MQ for z/OS , assicurarsi che IMS possa ottenere memoria sufficiente per contenere i messaggi fino a 100 MB di lunghezza.

Nota per gli utenti

L' *adattatore IMS* fornisce l'accesso alle risorse IBM MQ for z/OS per:

- MPP (Online Message Processing Program)
- IFP (fast path programs) interattivi
- BMP (batch message processing programs)

Per utilizzare queste risorse, i programmi devono essere in esecuzione in modalità task (TCB) e in stato di problema; non devono essere in modalità memoria incrociata o in modalità registro di accesso.

L'adattatore fornisce un thread di collegamento da un TCB (task control block) dell'applicazione a IBM MQ. L'adattatore supporta un protocollo di commit a due fasi per le modifiche apportate alle risorse di proprietà di IBM MQ for z/OS, con IMS che agisce come coordinatore del punto di sincronizzazione.

L'adattatore fornisce anche un programma di controllo dei trigger che può avviare i programmi automaticamente quando vengono soddisfatte determinate condizioni di trigger su una coda. Per ulteriori informazioni, consultare “Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866.

Se si stanno scrivendo programmi batch DL/I, seguire le istruzioni fornite in questo argomento per i programmi batch z/OS .

z/OS *Scrittura di applicazioni z/OS UNIX System Services*

L'adattatore batch supporta le connessioni del gestore code dagli spazi di indirizzi batch e TSO.

Per uno spazio di indirizzo batch, l'adattatore supporta connessioni da più TCB all'interno di tale spazio di indirizzo come segue:

- Ogni TCB può connettersi a più gestori code utilizzando la chiamata MQCONN o MQCONNX (ma un TCB può avere una sola istanza di connessione a un determinato gestore code alla volta).
- Più TCB possono connettersi allo stesso gestore code (ma l'handle del gestore code restituito in qualsiasi chiamata MQCONN o MQCONNX è collegato al TCB di emissione e non può essere utilizzato da altri TCB).

z/OS UNIX System Services supporta due tipi di chiamata pthread_create:

1. Thread massimi, eseguirli uno per ogni TCB, che sono ATTACHed e DETACHed all'inizio e alla fine del thread da z/OS.
2. Thread di peso medio, eseguire uno per ogni TCB, ma il TCB può essere uno di un pool di TCB di lunga durata. L'applicazione deve eseguire tutte le operazioni di ripulitura dell'applicazione necessarie, poiché, se è connessa a un server, la terminazione del thread predefinita che potrebbe essere fornita dal server alla terminazione del task (TCB), **non** è sempre guidata.

I thread lightweight non sono supportati. (Se un'applicazione crea thread permanenti che inviano le proprie richieste di lavoro, l'applicazione è responsabile della ripulitura di tutte le risorse prima di avviare la successiva richiesta di lavoro.)

IBM MQ for z/OS supporta i thread z/OS UNIX System Services utilizzando l'adattatore batch nel modo seguente:

1. I thread massimi sono completamente supportati come connessioni batch. Ogni thread viene eseguito nel proprio TCB, collegato e scollegato all'inizio e alla fine del thread. Se il thread termina prima di emettere una chiamata MQDISC, IBM MQ for z/OS esegue la ripulitura delle attività standard, che include il commit di qualsiasi unità di lavoro in sospeso se il thread è terminato normalmente o il backout se il thread è terminato in modo anomalo.
2. I thread di peso medio sono completamente supportati, ma se il TCB verrà riutilizzato da un altro thread, l'applicazione deve garantire che una chiamata MQDISC, preceduta da MQCMIT o MQBACK, venga emessa prima del successivo avvio del thread. Ciò implica che se l'applicazione ha stabilito un Program Interrupt Handler e l'applicazione si interrompe in modo anomalo, il Gestore interrupt deve emettere chiamate MQCMIT e MQDISC prima di riutilizzare il TCB per un altro thread.

Nota: I modelli di thread **non** supportano l'accesso alle risorse IBM MQ comuni da più thread.

z/OS *L'API - crossing exit per z/OS*

Questo argomento contiene informazioni sull'interfaccia di programmazione sensibile al prodotto.

Un'uscita è un punto nel codice fornito da IBMin cui è possibile eseguire il proprio codice. IBM MQ for z/OS fornisce un' *API - crossing exit* che puoi utilizzare per intercettare le chiamate alla MQI e per monitorare o modificare la funzione delle chiamate MQI. Questa sezione descrive come utilizzare l'uscita incrociata API e descrive il programma di uscita di esempio fornito con IBM MQ for z/OS.

Questa sezione è applicabile solo per gli utenti di CICS TS V3.1 e versioni precedenti. Gli utenti di CICS TS V3.2 e versioni successive devono fare riferimento alla sezione CICS Integration with IBM MQ nella documentazione del prodotto CICS .

Nota

L'uscita API - crossing viene richiamata solo dall'adattatore CICS di IBM MQ for z/OS. Il programma di uscita viene eseguito nello spazio di indirizzo CICS .

z/OS *Scrittura del proprio programma di uscita*

È possibile utilizzare il programma di uscita API - crossing di esempio (CSQCAPX) fornito con IBM MQ for z/OS come framework per il proprio programma.

Ciò è descritto in "Il programma di uscita API - crossing di esempio, CSQCAPX" a pagina 896.

Quando si scrive un programma di uscita, per trovare il nome di una chiamata MQI emessa da una applicazione, esaminare il campo *ExitCommand* della struttura MQXP. Per individuare il numero di parametri sulla chiamata, esaminare il campo *ExitParmCount* . È possibile utilizzare il campo *ExitUserArea* a 16 byte per memorizzare l'indirizzo di qualsiasi storage dinamico ottenuto dall'applicazione. Questo campo viene conservato nelle chiamate dell'uscita e ha la stessa durata di un'attività CICS .

Se si utilizza CICS Transaction Server V3.2, è necessario scrivere il programma di uscita in modo che sia protetto dal sottoprocesso e dichiarare il programma di uscita come protetto dal sottoprocesso. Se

si utilizzano release precedenti di CICS , si consiglia anche di scrivere e dichiarare i programmi di uscita come thread - safe per essere pronti per la migrazione a CICS Transaction Server V3.2.

Il programma di uscita può sopprimere l'esecuzione di una chiamata MQI restituendo MQXCC_SUPPRESS_FUNCTION o MQXCC_SKIP_FUNCTION nel campo *ExitResponse* . Per consentire l'esecuzione della chiamata (e il richiamo del programma di uscita dopo che la chiamata è stata completata), il programma di uscita deve restituire MQXCC_OK.

Quando viene richiamato dopo una chiamata MQI, un programma di uscita può esaminare e modificare i codici di completamento e motivo impostati dalla chiamata.

Note d'utilizzo

Di seguito sono riportati alcuni punti generali da considerare quando si scrive il programma di uscita:

- Per motivi di prestazioni, scrivere il programma in linguaggio assembler. Se lo si scrive in una delle altre lingue supportate da IBM MQ for z/OS, è necessario fornire il proprio file di definizione dati.
- Link - modificare il programma come AMODE (31) e RMODE (ANY).
- Per definire il blocco del parametro di uscita nel proprio programma, utilizzare la macro del linguaggio assembler, CMQXPA.
- Specificare CONCURRENCY (THREADSAFE) quando si definisce il programma di uscita e tutti i programmi richiamati dal programma di uscita.
- Se si utilizza la funzione di protezione della memoria di CICS Transaction Server per z/OS , il proprio programma deve essere eseguito nella chiave di esecuzione CICS . Ovvero, è necessario specificare EXECKEY (CICS) quando si definisce sia il programma di uscita che i programmi a cui passa il controllo. Per informazioni sui programmi di uscita CICS e sulla funzione di protezione della memoria CICS , consultare *CICS Customization Guide*.
- Il programma può utilizzare tutte le API (ad esempio, IMS, Db2e CICS) che può essere utilizzato da un programma user exit correlato al task CICS . Può anche utilizzare qualsiasi chiamata MQI tranne MQCONN, MQCONNX e MQDISC. Tuttavia, le chiamate MQI all'interno del programma di uscita non richiamano il programma di uscita una seconda volta.
- Il proprio programma può emettere i comandi EXEC CICS SYNCPOINT o EXEC CICS SYNCPOINT ROLLBACK. Tuttavia, questi comandi eseguono il commit o il rollback ***di tutti*** gli aggiornamenti effettuati dall'attività fino al punto in cui l'uscita è stata utilizzata e quindi il loro utilizzo non è consigliato.
- Il programma deve terminare immettendo un comando EXEC CICS RETURN. Non deve trasferire il controllo con un comando XCTL.
- Le uscite vengono scritte come estensioni del codice IBM MQ for z/OS . Assicurarsi che l'uscita non interrompe i programmi o le transazioni IBM MQ for z/OS che utilizzano MQI. Questi sono generalmente indicati con un prefisso CSQ o CK.
- Se CSQCAPX è definito in CICS, il sistema CICS tenta di caricare il programma di uscita quando CICS si connette a IBM MQ for z/OS. Se questo tentativo ha esito positivo, viene inviato il messaggio CSQC301I al pannello CKQC o alla console di sistema. Se il caricamento non riesce (ad esempio, se il modulo di caricamento non esiste in nessuna delle librerie nella concatenazione DFHRPL), il messaggio CSQC315 viene inviato al pannello CKQC o alla console di sistema.
- Poiché i parametri nell'area di comunicazione sono indirizzi, il programma di uscita deve essere definito come locale per il sistema CICS (ossia, non come programma remoto).

z/OS *Il programma di uscita API - crossing di esempio, CSQCAPX*

Il programma di uscita di esempio viene fornito come programma in linguaggio assembler. Il file di origine (CSQCAPX) viene fornito nella libreria **thlqual**.SCSQASMS (dove **thlqual** è il qualificatore di alto livello utilizzato dall'installazione). Questo file origine include pseudocodice che descrive la logica del programma.

Il programma di esempio contiene il codice di inizializzazione e un layout che è possibile utilizzare quando si scrivono i programmi di uscita.

L'esempio mostra come:

- Imposta il blocco del parametro di uscita
- Indirizzare i blocchi di parametri di chiamata e uscita
- Determinare per quale chiamata MQI viene richiamata l'exit
- Determinare se l'uscita viene richiamata prima o dopo l'elaborazione della chiamata MQI
- Inserire un messaggio in una coda di storage temporaneo CICS
- Utilizzare la macro DFHEIENT per l'acquisizione di storage dinamico per mantenere la rientranza
- Utilizzare DFHEIBLK per il blocco di controllo dell'interfaccia exec CICS
- Condizioni di errore trap
- Restituisci il controllo al chiamante

Progettazione del programma di uscita di esempio

Il programma di uscita di esempio scrive i messaggi in una coda di storage temporaneo CICS (CSQ1EXIT) per mostrare l'operazione dell'uscita.

I messaggi mostrano se l'uscita viene richiamata prima o dopo la chiamata MQI. Se l'uscita viene richiamata dopo la chiamata, il messaggio contiene il codice di completamento e il codice motivo restituiti dalla chiamata. L'esempio utilizza costanti denominate dalla macro CMQXPA per controllare il tipo di voce (ovvero, prima o dopo la chiamata).

L'esempio non esegue alcuna funzione di controllo, ma inserisce semplicemente i messaggi con data / ora in una coda CICS che indica il tipo di chiamata che sta elaborando. Ciò fornisce un'indicazione delle prestazioni dell'MQI e del funzionamento corretto del programma di uscita.

Nota: Il programma di uscita di esempio emette sei chiamate EXEC CICS per ciascuna chiamata MQI effettuata durante l'esecuzione del programma. Se si utilizza questo programma di uscita, le prestazioni di IBM MQ for z/OS sono ridotte.

z/OS *Preparazione e utilizzo dell'uscita di attraversamento API*

L'uscita di esempio viene fornita solo in formato origine.

Per utilizzare l'uscita di esempio o un programma di uscita scritto, creare una libreria di caricamento, come per qualsiasi altro programma CICS , come descritto in “Creazione di applicazioni CICS in z/OS” a pagina 1029.

- Per CICS Transaction Server per z/OS e CICS per MVS/ESA, quando si aggiorna il dataset CSD (CICS system definition), le definizioni necessarie si trovano nel membro **thlqual**.SCSQPROC (CSQ4B100).

  Nota: Le definizioni utilizzano un suffisso di MQ. Se questo suffisso è già utilizzato nell'azienda, è necessario modificarlo prima della fase di assemblaggio.

Se si utilizzano le definizioni di programma CICS predefinite, il programma di uscita CSQCAPX viene installato in uno stato ***disabilitato*** . Questo perché l'utilizzo del programma di uscita può produrre una riduzione significativa delle prestazioni.

Per attivare temporaneamente l'uscita di attraversamento API:

1. Immettere il comando **CEMT S PROGRAM(CSQCAPX) ENABLED** dal terminale master CICS .
2. Eseguire la transazione CKQC e utilizzare l'opzione 3 nel menu a discesa Connessione per modificare lo stato dell'uscita di attraversamento API in **Abilitato**.

Se si desidera eseguire IBM MQ for z/OS con l'uscita API - crossing abilitata in modo permanente, con CICS Transaction Server per z/OS e CICS per MVS/ESA, effettuare una delle seguenti operazioni:

- Modificare la definizione CSQCAPX nel membro CSQ4B100, modificando STATUS (DISABLED) in STATUS (ENABLED). È possibile aggiornare la definizione CSD di CICS utilizzando il programma batch DFHCSDUP fornito da CICS.
- Modificare la definizione CSQCAPX nel gruppo CSQCAT1 modificando lo stato da DISABLED a ENABLED.

In entrambi i casi, è necessario reinstallare il gruppo. È possibile eseguire questa operazione avviando a freddo il sistema CICS o utilizzando la transazione CEDA CICS per reinstallare il gruppo mentre CICS è in esecuzione.

Nota: L'utilizzo di CEDA potrebbe causare un errore se una delle voci nel gruppo è attualmente in uso.

Informazioni sull'interfaccia di programmazione sensibile al prodotto.

z/OS *Programmazione di applicazioni con code condivise*

Questo argomento fornisce informazioni su alcuni dei fattori che è necessario considerare quando si progettano nuove applicazioni per utilizzare le code condivise e quando si migrano le applicazioni esistenti nell'ambiente della coda condivisa.

z/OS *Serializzazione delle applicazioni*

Alcuni tipi di applicazioni potrebbero dover garantire che i messaggi vengano richiamati da una coda esattamente nello stesso ordine in cui sono arrivati sulla coda.

Ad esempio, se IBM MQ viene utilizzato per eseguire l'shadow degli aggiornamenti del database su un sistema remoto, un messaggio che descrive l'aggiornamento di un record deve essere elaborato dopo un messaggio che descrive l'inserimento di tale record. In un ambiente di accodamento locale, ciò è spesso ottenuto dall'applicazione che sta ricevendo i messaggi che aprono la coda con l'opzione MQOO_INPUT_EXCLUSIVE, impedendo così a qualsiasi altra applicazione che ottiene di elaborare la coda contemporaneamente.

IBM MQ consente alle applicazioni di aprire le code condivise esclusivamente nello stesso modo. Tuttavia, se l'applicazione sta funzionando da una partizione di una coda (ad esempio, tutti gli aggiornamenti del database si trovano sulla stessa coda, ma quelli per la tabella A hanno un identificativo di correlazione di A e quelli per la tabella B un identificativo di correlazione di B) e le applicazioni desiderano ottenere i messaggi per gli aggiornamenti della tabella A e della tabella B contemporaneamente, il semplice meccanismo di apertura della coda in modo esclusivo non è possibile.

Se questo tipo di applicazione deve sfruttare l'alta disponibilità delle code condivise, è possibile decidere che un'altra istanza dell'applicazione che accede alle stesse code condivise, in esecuzione su un gestore code secondario, prenda il sopravvento se l'applicazione o il gestore code di ricezione primario ha esito negativo.

Se il gestore code primario ha esito negativo, si verificano due situazioni:

- Il ripristino peer della coda condivisa garantisce che tutti gli aggiornamenti incompleti dall'applicazione primaria vengano completati o sottoposti a backout.
- L'applicazione secondaria assume il controllo dell'elaborazione della coda.

L'applicazione secondaria potrebbe essere avviata prima che siano state gestite tutte le unità di lavoro incomplete, il che potrebbe portare l'applicazione secondaria a richiamare i messaggi fuori sequenza. Per risolvere questo tipo di problema, l'applicazione può scegliere di essere un' *applicazione serializzata*.

Un'applicazione serializzata utilizza la chiamata MQCONNX per connettersi al gestore code, specificando una tag di connessione quando si connette univoca per tale applicazione. Tutte le unità di lavoro eseguite dall'applicazione sono contrassegnate con il tag di connessione. IBM MQ garantisce che le unità di lavoro all'interno del gruppo di condivisione code con la stessa tag di connessione vengano serializzate (in base alle opzioni di serializzazione sulla chiamata MQCONNX).

Ciò significa che, se l'applicazione primaria utilizza la chiamata MQCONNX con una tag di connessione `Database shadow retrievere` l'applicazione di takeover secondaria tenta di utilizzare una chiamata MQCONNX con una tag di connessione identica, l'applicazione secondaria non può connettersi alla seconda IBM MQ fino a quando non sono state completate le unità di lavoro principali in sospeso, in questo caso dal ripristino peer.

Considerare l'utilizzo della tecnica dell'applicazione serializzata per le applicazioni che dipendono dalla sequenza esatta dei messaggi su una coda. In particolare:

- Le applicazioni che non devono essere riavviate dopo un errore dell'applicazione o del gestore code fino al completamento di tutte le operazioni di commit e backout per la precedente esecuzione dell'applicazione.

  In questo caso, la tecnica dell'applicazione serializzata è applicabile solo se l'applicazione funziona nel punto di sincronizzazione.
- Applicazioni che non devono essere avviate mentre è già in esecuzione un'altra istanza della stessa applicazione.

  In questo caso, la tecnica dell'applicazione serializzata è richiesta solo se l'applicazione non può aprire la coda per l'input esclusivo.

Nota: IBM MQ garantisce solo la conservazione della sequenza di messaggi quando vengono soddisfatti determinati criteri. Questi sono descritti nella descrizione di MQGET.

z/OS *Applicazioni non adatte per l'utilizzo con code condivise*

Alcune funzioni di IBM MQ non sono supportate quando si utilizzano le code condivise, pertanto le applicazioni che utilizzano tali funzioni non sono adatte all'ambiente della coda condivisa.

Considerare i seguenti punti quando si progettano le applicazioni della coda condivisa:

- L'indicizzazione della coda è limitata per code condivise. Se si desidera utilizzare l'identificativo del messaggio o l'identificativo di correlazione per selezionare il messaggio che si desidera richiamare dalla coda, la coda deve essere indicizzata con il valore corretto. Se si selezionano i messaggi solo in base all'identificatore del messaggio, la coda necessita di un tipo di indice MQIT_MSG_ID (anche se è possibile utilizzare MQIT_NONE). Se si stanno selezionando messaggi solo in base all'identificativo di correlazione, la coda deve avere un tipo di indice MQIT_CORREL_ID.
- Non è possibile utilizzare code dinamiche temporanee come code condivise. Tuttavia, è possibile utilizzare code dinamiche permanenti. I modelli per le code dinamiche condivise hanno un DEFTYPE di SHAREDYN (dinamica condivisa) anche se vengono create ed eliminate allo stesso modo delle code PERMDYN (dinamiche permanenti).

z/OS *Decidere se condividere code non di applicazione*

Utilizzare queste informazioni quando si considera la condivisione di code non di applicazione.

Esistono code diverse dalle code dell'applicazione che si potrebbe considerare di condividere:

Code di iniziazione

Se si definisce una coda di iniziazione condivisa, non è necessario avere un controllo trigger in esecuzione su ogni gestore code nel gruppo di condivisione code, purché sia in esecuzione almeno un controllo trigger. È inoltre possibile utilizzare una coda di iniziazione condivisa anche se è presente un controllo trigger in esecuzione su ciascun gestore code nel gruppo di condivisione code.

Se si dispone di una coda di applicazioni condivisa e si utilizza il tipo di trigger EVERY (o un tipo di trigger FIRST con un intervallo di trigger ridotto, che si comporta come un tipo di trigger EVERY), la coda di iniziazione deve essere sempre una coda condivisa. Per ulteriori informazioni su quando utilizzare una coda di avvio condivisa, consultare Tabella 131 a pagina 900.

SYSTEM.* Code

È possibile definire il SISTEMA SYSTEM.ADMIN.* code utilizzate per conservare i messaggi di eventi come code condivise. Ciò può essere utile per controllare il bilanciamento del carico se si verifica un'eccezione. Ogni messaggio di eventi creato da IBM MQ contiene un identificativo di correlazione che indica quale gestore code lo ha prodotto.

È necessario definire il SISTEMA SYSTEM.QSG.* code utilizzate per i canali condivisi e l'accodamento all'interno del gruppo come code condivise.

È anche possibile modificare le definizioni di SYSTEM.DEFAULT.LOCAL.QUEUE da condividere o definire la propria definizione di coda condivisa predefinita. Consultare Definizione degli oggetti di sistema per IBM MQ for z/OS per ulteriori informazioni.

Non è possibile definire altri SYSTEM.* code condivise.

z/OS *Migrazione delle proprie applicazioni esistenti per utilizzare le code condivise*

I codici di origine errore, il trigger e la chiamata API MQINQ possono funzionare in modo diverso in un ambiente di code condivise.

Consultare Migrazione di code non condivise in code condivise per informazioni sulla migrazione delle code esistenti in code condivise.

Quando si migrano le applicazioni esistenti, considerare quanto segue, che potrebbe funzionare in modo diverso nell'ambiente della coda condivisa:

Codici di origine

Quando si migrano le applicazioni esistenti per utilizzare le code condivise, verificare la presenza di nuovi codici motivo che possono essere emessi.

Triggering

Se si utilizza una coda di applicazioni condivisa, l'attivazione funziona solo sui messaggi di cui è stato eseguito il commit (su una coda di applicazioni non condivisa, l'attivazione funziona su tutti i messaggi).

Se si utilizza il trigger per avviare le applicazioni, è possibile utilizzare una coda di avvio condivisa. Tabella 131 a pagina 900 descrive cosa è necessario considerare quando si decide quale tipo di coda di avvio utilizzare.

Tabella 131. Quando utilizzare una coda di iniziazione condivisa

	Coda applicazione non condivisa	**Coda applicazione condivisa**
Coda di avvio non condivisa	Come per le release precedenti.	Se si utilizza un tipo di trigger FIRST o DEPTH, è possibile utilizzare una coda di avvio non condivisa con una coda di applicazione condivisa. Potrebbero essere generati ulteriori messaggi di trigger, ma questa configurazione è utile per attivare applicazioni di lunga durata (come CICS bridge) e fornisce l'alta disponibilità. Per il tipo di trigger FIRST o DEPTH, un messaggio di trigger attiva un'istanza dell'applicazione su ogni gestore code che sta eseguendo un controllo dei trigger e che non ha già la coda dell'applicazione aperta per l'input. Viene generato un messaggio trigger per ogni gestore code; se è presente più di un controllo trigger in esecuzione rispetto alla coda di avvio locale non condivisa, su un particolare gestore code, essi competeranno per elaborare il messaggio.

Tabella 131. Quando utilizzare una coda di iniziazione condivisa (Continua)

	Coda applicazione non condivisa	Coda applicazione condivisa
Coda di iniziazione condivisa	Non utilizzare una coda di iniziazione condivisa con una coda di applicazione non condivisa.	Per il tipo di trigger EVERY, quando un'applicazione inserisce un messaggio in una coda di applicazione condivisa, il gestore code di inserimento determina quali gestori code sono interessati al trigger - ogni evento e invia una notifica a uno di questi gestori code. Sul gestore code notificato, l'azione risultante è generare un messaggio trigger per la coda di iniziazione. **Nota:** Se si dispone di una coda di applicazioni condivisa con un tipo di trigger EVERY, utilizzare una coda di iniziazione condivisa oppure si potrebbero perdere i messaggi di trigger in determinate circostanze; ad esempio, un errore del gestore code. Per il tipo di trigger FIRST o DEPTH, viene generato un messaggio trigger da ogni gestore code che ha la coda di avvio denominata aperta per l'input. **Nota:** Per il tipo di trigger FIRST o DEPTH, se un'istanza del controllo trigger è occupata, ciò lascia la possibilità per i controlli trigger meno occupati di elaborare più di un messaggio trigger dalla coda di avvio condivisa. Pertanto, è possibile avviare più istanze dell'applicazione server rispetto a uno specifico gestore code. Si noti che queste istanze multiple vengono avviate come risultato dell'elaborazione di più messaggi trigger. Normalmente, per il tipo di trigger FIRST o DEPTH, se un'istanza dell'applicazione sta già servendo una coda dell'applicazione, un altro messaggio di trigger non verrà generato dal gestore code a cui è connessa l'applicazione.

MQINQ

Quando si utilizza la chiamata MQ per visualizzare le informazioni su una coda condivisa, i valori del numero di chiamate MQOPEN che hanno la coda aperta per l'input e l'output si riferiscono solo al gestore code che ha emesso la chiamata. Non vengono prodotte informazioni sugli altri gestori code nel gruppo di condivisione code che hanno la coda aperta.

z/OS Applicazioni bridge IMS e IMS su IBM MQ for z/OS

Queste informazioni consentono di scrivere applicazioni IMS utilizzando IBM MQ.

- Per utilizzare i syncpoint e le chiamate MQI nelle applicazioni IMS , consultare “Scrittura di applicazioni IMS utilizzando IBM MQ” a pagina 72.
- Per scrivere le applicazioni che utilizzano il bridge IBM MQ - IMS , consultare “Scrittura delle applicazioni bridge IMS” a pagina 76.

Utilizzare i seguenti link per ulteriori informazioni sulle applicazioni bridge IMS e IMS su IBM MQ for z/OS:

- “Scrittura di applicazioni IMS utilizzando IBM MQ” a pagina 72
- “Scrittura delle applicazioni bridge IMS” a pagina 76

Concetti correlati

“Panoramica su Message Queue Interface” a pagina 722

Informazioni sui componenti MQI (Message Queue Interface).

“Connessione e disconnessione da un gestore code” a pagina 735
Per utilizzare i servizi di programmazione IBM MQ , un programma deve disporre di una connessione a un gestore code. Utilizzare queste informazioni per informazioni su come connettersi e disconnettersi da un gestore code.

“Apertura e chiusura di oggetti” a pagina 742
Queste informazioni forniscono informazioni sull'apertura e la chiusura degli oggetti IBM MQ .

“Inserimento di messaggi in una coda” a pagina 753
Utilizzare queste informazioni per informazioni su come inserire i messaggi su una coda.

“Richiamo dei messaggi da una coda” a pagina 768
Utilizzare queste informazioni per ottenere messaggi da una coda.

“Richiesta di informazioni e impostazione degli attributi dell'oggetto” a pagina 851
Gli attributi sono proprietà che definiscono le caratteristiche di un oggetto IBM MQ .

“Commit e backout delle unità di lavoro” a pagina 854
Queste informazioni descrivono come eseguire il commit e il backout di tutte le operazioni get e put recuperabili che si sono verificate in un'unità di lavoro.

“Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866
Informazioni sui trigger e su come avviare le applicazioni IBM MQ utilizzando i trigger.

“Utilizzo di MQI e cluster” a pagina 886
Esistono opzioni speciali sulle chiamate e sui codici di ritorno relativi al cluster.

“Utilizzo e scrittura di applicazioni su IBM MQ for z/OS” a pagina 890
Le applicazioni IBM MQ for z/OS possono essere composte da programmi eseguiti in molti ambienti differenti. Ciò significa che possono usufruire delle strutture disponibili in più di un ambiente.

z/OS *Scrittura di applicazioni IMS utilizzando IBM MQ*

Ci sono ulteriori considerazioni quando si utilizza IBM MQ nelle applicazioni IMS , tra cui quali chiamate API di MQ possono essere utilizzate e il meccanismo utilizzato per il punto di sincronizzazione.

Utilizzare i seguenti link per ulteriori informazioni sulla scrittura di applicazioni IMS su IBM MQ for z/OS:

- “Punti di sincronizzazione nelle applicazioni IMS” a pagina 72
- “Chiamate MQI nelle applicazioni IMS” a pagina 73

Limitazioni

Esistono delle limitazioni su cui le chiamate API IBM MQ possono essere utilizzate da un'applicazione utilizzando l'adattatore IMS .

Le seguenti chiamate API IBM MQ non sono supportate in una applicazione che utilizza l'adattatore IMS :

- MQCB
- MQCB_FUNZIONE
- MQCTL

Concetti correlati

“Scrittura delle applicazioni bridge IMS” a pagina 76
Questo argomento contiene informazioni sulla scrittura di applicazioni per utilizzare il bridge IBM MQ - IMS .

z/OS *Punti di sincronizzazione nelle applicazioni IMS*

In un'applicazione IMS , si stabilisce un punto di sincronizzazione utilizzando chiamate IMS come GU (get unique) a IOPCB e CHKP (checkpoint).

Per annullare tutte le modifiche dal punto di controllo precedente, è possibile utilizzare la chiamata IMS ROLB (rollback). Per ulteriori informazioni, vedi ROLB call nella documentazione di IMS .

Il gestore code è un partecipante in un protocollo di commit a due fasi; il gestore del punto di sincronizzazione IMS è il coordinatore.

Tutte le maniglie aperte vengono chiuse dall'adattatore IMS in un punto di sincronizzazione (tranne in un ambiente BMP batch o non basato su messaggi). Ciò è dovuto al fatto che un utente differente potrebbe avviare la successiva unità di lavoro e il controllo di sicurezza di IBM MQ viene eseguito quando vengono effettuate le chiamate MQCONN, MQCONNX e MQOPEN, non quando vengono effettuate le chiamate MQPUT o MQGET.

Tuttavia, in un ambiente WFI (Wait - for - Input) o PWFI (pseudo Wait - for - Input) IMS non notifica IBM MQ di chiudere gli handle fino a quando non arriva il messaggio successivo o non viene restituito un codice di stato QC all'applicazione. Se l'applicazione è in attesa nella regione IMS e uno di questi handle appartiene a code attivate, l'attivazione non si verificherà perché le code sono aperte. Per questo motivo, le applicazioni in esecuzione in un ambiente WFI o PWFI devono esplicitamente MQCLOSE che la coda gestisce prima di eseguire il GU all'IOPCB per il messaggio successivo.

Se un'applicazione IMS (BMP o MPP) emette la chiamata MQDISC, le code aperte vengono chiuse ma non viene utilizzato alcun punto di sincronizzazione implicito. Se l'applicazione termina normalmente, tutte le code aperte vengono chiuse e si verifica un commit implicito. Se l'applicazione termina in modo anomalo, tutte le code aperte vengono chiuse e si verifica un backout implicito.

z/OS *Chiamate MQI nelle applicazioni IMS*

Utilizzare queste informazioni per informazioni sull'utilizzo delle chiamate MQI sulle applicazioni Server e sulle applicazioni di interrogazione.

Questa sezione riguarda l'utilizzo delle chiamate MQI nei seguenti tipi di applicazioni IMS :

- “Applicazioni del server” a pagina 903
- “Applicazioni di richiesta” a pagina 905

Applicazioni del server

Di seguito viene riportato uno schema del modello di applicazione server MQI:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

Il programma di esempio CSQ4ICB3 mostra l'implementazione, in C/370, di un BMP che utilizza questo modello. Il programma stabilisce prima la comunicazione con IMS e poi con IBM MQ:

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
```

```
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

L'inizializzazione di IMS determina se il programma è stato richiamato come un BMP orientato ai messaggi o batch e controlla la connessione del gestore code IBM MQ e gli handle di coda di conseguenza:

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

L'inizializzazione IBM MQ si connette al gestore code e apre le code. In un BMP basato sui messaggi, questo viene richiamato dopo ogni punto di sincronizzazione IMS ; in un BMP orientato al batch, questo viene richiamato solo durante l'avvio del programma:

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

L'implementazione del modello server in un MPP è influenzata dal fatto che l'MPP elabora una singola unità di lavoro per chiamata. Ciò si verifica perché, quando viene utilizzato un punto di sincronizzazione (GU), gli handle di connessione e di coda vengono chiusi e viene consegnato il successivo messaggio IMS . Questa limitazione può essere parzialmente superata da uno dei seguenti:

- **Elaborazione di molti messaggi in una singola unità di lavoro**

  Ciò comporta:

  – Lettura di un messaggio
  – Elaborazione degli aggiornamenti richiesti
  – Inserimento della risposta

  in un loop fino a quando non sono stati elaborati tutti i messaggi o fino a quando non è stato elaborato un numero massimo impostato di messaggi, in cui viene utilizzato un punto di sincronizzazione.

  Solo alcuni tipi di applicazione (ad esempio, un semplice aggiornamento del database o una richiesta di informazioni) possono essere avvicinati in questo modo. Sebbene i messaggi di risposta MQI possano essere inseriti con l'autorizzazione del creatore del messaggio MQI gestito, le implicazioni di sicurezza di qualsiasi aggiornamento delle risorse IMS devono essere affrontate con attenzione.
- **Elaborazione di un messaggio per richiamo di MPP e garanzia della pianificazione multipla di MPP per elaborare tutti i messaggi disponibili.**

  Utilizzare il programma di controllo trigger IBM MQ IMS (CSQQTRMN) per pianificare la transazione MPP quando ci sono messaggi sulla coda IBM MQ e nessuna applicazione che la supporta.

  Se il controllo dei trigger avvia MPP, il nome del gestore code e il nome della coda vengono passati al programma, come mostrato nel seguente estratto di codice COBOL:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

Il modello server, che si prevede sia un'attività di lunga durata, è meglio supportato in una regione di elaborazione batch, anche se BMP non può essere attivato utilizzando CSQQTRMN.

Applicazioni di richiesta

Un'applicazione tipica IBM MQ che avvia un'interrogazione o un aggiornamento funziona nel modo seguente:

- Raccogliere dati dall'utente
- Inserire uno o più messaggi IBM MQ
- Richiamare i messaggi di risposta (potrebbe essere necessario attenderli)
- Fornire una risposta all'utente

Poiché i messaggi inseriti nelle code IBM MQ non diventano disponibili per altre applicazioni IBM MQ fino a quando non ne viene eseguito il commit, devono essere inseriti fuori dal punto di sincronizzazione oppure l'applicazione IMS deve essere suddivisa in due transazioni.

Se l'interrogazione implica l'inserimento di un singolo messaggio, è possibile utilizzare l'opzione *nessun punto di sincronizzazione* ; tuttavia, se l'interrogazione è più complessa o gli aggiornamenti delle risorse sono coinvolti, è possibile che si verifichino problemi di congruenza se si verifica un errore e non si utilizza il punto di sincronizzazione.

Per ovviare a questo problema, è possibile suddividere le transazioni IMS MPP utilizzando chiamate MQI utilizzando un commutatore di messaggi da programma a programma; per informazioni, consultare *IMS ISC (Intersystem Communication)* . Ciò consente a un programma di interrogazione di essere implementato in un MPP:

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS *Scrittura delle applicazioni bridge IMS*

Questo argomento contiene informazioni sulla scrittura di applicazioni per utilizzare il bridge IBM MQ - IMS .

Per informazioni sul bridge IBM MQ - IMS , vedere Il bridge IMS.

Utilizzare i seguenti link per ulteriori informazioni sulla scrittura di applicazioni bridge IMS su IBM MQ for z/OS:

- “Come il bridge IMS gestisce i messaggi” a pagina 76
- “Scrittura di programmi di transazione IMS tramite IBM MQ” a pagina 913

Concetti correlati

“Scrittura di applicazioni IMS utilizzando IBM MQ” a pagina 72
Ci sono ulteriori considerazioni quando si utilizza IBM MQ nelle applicazioni IMS , tra cui quali chiamate API di MQ possono essere utilizzate e il meccanismo utilizzato per il punto di sincronizzazione.

z/OS *Come il bridge IMS gestisce i messaggi*

Quando si utilizza il bridge IBM MQ - IMS per inviare i messaggi a un'applicazione IMS , è necessario creare i messaggi in un formato speciale.

È inoltre necessario inserire i messaggi nelle code IBM MQ che sono state definite con una classe di memoria che specifica il gruppo XCF e il nome membro del sistema IMS di destinazione. Queste sono note come code bridge MQ-IMS o semplicemente code **bridge** .

Il bridge IBM MQ-IMS richiede l'accesso di immissione esclusivo (MQOO_INPUT_EXCLUSIVE) alla coda bridge se è definita con QSGDISP (QMGR) o se è definita con QSGDISP (SHARED) insieme all'opzione NOSHARE.

Un utente non ha bisogno di collegarsi a IMS prima di inviare messaggi a una applicazione IMS . L'ID utente nel campo *UserIdentifier* della struttura MQMD viene utilizzato per il controllo di sicurezza. Il livello di controllo viene determinato quando IBM MQ si connette a IMSed è descritto in Application access control for IMS bridge. Ciò consente l'implementazione di uno pseudo accesso.

Il bridge di IBM MQ - IMS accetta i seguenti tipi di messaggio:

- Messaggi contenenti i dati della transazione IMS e una struttura MQIIH (descritta in MQIIH):

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

Nota:

1. Le parentesi quadre, [], rappresentano segmenti multipli facoltativi.
2. Impostare il campo *Format* della struttura di MQMD su MQFMT_IMS per utilizzare la struttura MQIIH.

- Messaggi contenenti dati di transazione IMS ma nessuna struttura MQIIH:

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ convalida i dati del messaggio per garantire che la somma dei byte LL più la lunghezza di MQIIH (se presente) sia uguale alla lunghezza del messaggio.

Quando il bridge IBM MQ - IMS richiama i messaggi dalle code bridge, li elabora nel modo seguente:

- Se il messaggio contiene una struttura MQIIH, il bridge verifica MQIIH (vedere MQIIH), crea le intestazioni OTMA e invia il messaggio a IMS. Il codice di transazione è specificato nel messaggio di input. Se si tratta di un LTERM, IMS risponde con un messaggio DFS1288E . Se il codice della transazione rappresenta un comando, IMS esegue il comando; altrimenti il messaggio viene accodato in IMS per la transazione.
- Se il messaggio contiene dati di transazione IMS , ma non una struttura MQIIH, il bridge IMS fa i seguenti presupposti:
  - Il codice transazione è espresso in byte da 5 a 12 dei dati utente
  - La transazione è in modalità non conversazionale
  - La transazione è in modalità di commit 0 (commit - e - invio)
  - Il *Format* in MQMD viene utilizzato come *MFSMapName* (in input)
  - La modalità di protezione è MQISS_CHECK

  Anche il messaggio di risposta viene creato senza una struttura MQIIH, prendendo *Format* per MQMD dall' *MFSMapName* dell'output IMS .

Il bridge IBM MQ - IMS utilizza uno o due Tpipe per ogni coda IBM MQ :

- Un Tpipe sincronizzato viene utilizzato per tutti i messaggi che utilizzano la modalità Commit 0 (COMMIT_THEN_SEND) (questi vengono visualizzati con SYN nel campo di stato del comando TPIPE xxxx del client TMEMBER IMS /DIS)
- Un Tpipe non sincronizzato viene utilizzato per tutti i messaggi che utilizzano la modalità Commit 1 (SEND_THEN_COMMIT)

I Tpipe vengono creati da IBM MQ quando vengono utilizzati per la prima volta. Esiste un Tpipe non sincronizzato fino a quando IMS non viene riavviato. I Tpipe sincronizzati esistono fino a quando IMS non viene avviato a freddo. Non è possibile eliminare questi Tpipe.

Per ulteriori informazioni su come il bridge IBM MQ - IMS gestisce i messaggi, consultare i seguenti argomenti:

- “Associazione di messaggi IBM MQ a tipi di transazioni IMS” a pagina 77
- “Se il messaggio non può essere inserito nella coda IMS” a pagina 78
- “Codici di feedback del bridge IMS” a pagina 78
- “I campi MQMD nei messaggi dal bridge IMS .” a pagina 79
- “I campi MQIIH nei messaggi dal bridge IMS” a pagina 80

- “Messaggi di risposta da IMS” a pagina 81
- “Utilizzo di PCB di risposta alternativi in transazioni IMS” a pagina 81
- “Invio di messaggi non richiesti da IMS” a pagina 81
- “Segmentazione del messaggio” a pagina 82
- “Conversione dei dati per i messaggi da e verso il bridge IMS” a pagina 82

Concetti correlati

“Scrittura di programmi di transazione IMS tramite IBM MQ” a pagina 913
La codifica richiesta per gestire le transazioni IMS tramite IBM MQ dipende dal formato del messaggio richiesto dalla transazione IMS e dall'intervallo di risposte che può restituire. Tuttavia, ci sono diversi punti da considerare quando la tua applicazione gestisce le informazioni di formattazione dello schermo IMS .

z/OS *Associazione di messaggi IBM MQ a tipi di transazioni IMS*

Una tabella che descrive l'associazione dei messaggi IBM MQ ai tipi di transazione IMS .

Tabella 132. Modalità di associazione dei messaggi IBM MQ ai tipi di transazione IMS

IBM MQTipo messaggio	**Commit - then - send (modalità 0) - utilizza Tpipe IMS sincronizzati**	**Send - then - commit (modalità 1) - utilizza Tpipe IMS non sincronizzati**
Messaggi IBM MQ persistenti	• Transazioni con funzioni complete recuperabili • Le transazioni non recuperabili vengono rifiutate da IMS	• Transazioni Fastpath • Transazioni conversazionali • Transazioni con funzioni complete
Messaggi IBM MQ non persistenti	• Transazioni con funzioni complete non recuperabili • Le transazioni ripristinabili sono consentite con IMS V8 e APAR PQ61404 e tutte le versioni successive di IMS	• Transazioni Fastpath • Transazioni conversazionali • Transazioni con funzioni complete

Nota: I comandi IMS non possono utilizzare messaggi IBM MQ persistenti con modalità di commit 0. Per ulteriori informazioni, consultare Modalità commit (commitMode) .

z/OS *Se il messaggio non può essere inserito nella coda IMS*

Informazioni sulle azioni da intraprendere se il messaggio non può essere inserito nella coda IMS .

Se il messaggio non può essere inserito nella coda IMS , la seguente azione viene eseguita da IBM MQ:

- Se un messaggio non può essere inserito in IMS perché non è valido, viene inserito nella coda di messaggi non recapitabili e un messaggio viene inviato alla console di sistema.
- Se il messaggio è valido, ma è rifiutato da IMS, IBM MQ invia un messaggio di errore alla console di sistema, il messaggio include il codice di condizione IMS e il messaggio IBM MQ viene inserito nella coda dei messaggi non instradabili. Se il codice di rilevamento IMS è 001A, IMS invia un messaggio IBM MQ contenente il motivo dell'esito negativo alla coda di risposta.

  Nota: Nelle circostanze elencate in precedenza, se IBM MQ non è in grado di inserire il messaggio nella coda di messaggi non instradabili per qualsiasi motivo, il messaggio viene restituito alla coda IBM MQ di origine. Un messaggio di errore viene inviato alla console di sistema e non vengono inviati ulteriori messaggi da tale coda.

  Per inviare di nuovo i messaggi, eseguire ***uno*** dei seguenti:

  – Arrestare e riavviare i Tpipe in IMS corrispondenti alla coda

- Modificare la coda in GET (DISABLED) e di nuovo in GET (ENABLED)
- Arrestare e riavviare IMS o OTMA
- Arrestare e riavviare il sottosistema IBM MQ

- Se il messaggio viene rifiutato da IMS per un messaggio diverso da un errore, il messaggio IBM MQ viene restituito alla coda di origine, IBM MQ arresta l'elaborazione della coda e viene inviato un messaggio di errore alla console di sistema.

  Se è richiesto un messaggio di report di eccezioni, il bridge lo inserisce nella coda di risposta con l'autorizzazione del creatore. Se il messaggio non può essere inserito nella coda, il messaggio di prospetto viene inserito nella coda di messaggi non recapitabili con l'autorizzazione del bridge. Se non è possibile inserirlo nella DLQ, viene eliminato.

z/OS *Codici di feedback del bridge IMS*

I codici di rilevamento IMS vengono generalmente emessi in formato esadecimale nei messaggi della console IBM MQ come CSQ2001I (ad esempio, codice di rilevamento 0x001F). I codici di feedback IBM MQ come visualizzati nell'intestazione dei messaggi non instradabili inseriti nella coda dei messaggi non instradabili sono numeri decimali.

I codici di feedback del bridge IMS sono compresi tra 301 e 399 o tra 600 e 855 per il codice di rilevamento NACK 0x001A. Vengono associati dai codici di rilevamento IMS-OTMA come segue:

1. Il codice di rilevamento IMS-OTMA viene convertito da un numero esadecimale a un numero decimale.
2. 300 viene aggiunto al numero risultante dal calcolo in 1, fornendo il codice IBM MQ *Feedback* .
3. Il codice di rilevamento IMS-OTMA 0x001A, decimale 26 è un caso speciale. Viene generato un codice *Feedback* compreso tra 600 e 855.

   a. Il codice motivo IMS-OTMA viene convertito da un numero esadecimale a un numero decimale.

   b. 600 viene aggiunto al numero risultante dal calcolo in a, fornendo il codice IBM MQ *Feedback* .

Per informazioni sui codici di rilevamento IMS-OTMA, consultare Sense code OTMA per i messaggi NAK.

z/OS *I campi MQMD nei messaggi dal bridge IMS .*

Informazioni sui campi MQMD nei messaggi dal bridge IMS .

L'MQMD del messaggio di origine viene trasmesso da IMS nella sezione Dati utente delle intestazioni OTMA. Se il messaggio ha origine in IMS, viene creato da IMS Destination Resolution Exit. L'MQMD di un messaggio ricevuto da IMS viene creato come segue:

StrucID
"MG"

Versione
MQMD_VERSION_1

Prospetto
MQRO_NONE

MsgType
MQMT_REPLY

Scadenza
Se MQIIH_PASS_EXPIRATION è impostato nel campo Indicatori di MQIIH, questo campo contiene il tempo di scadenza rimanente, altrimenti è impostato su MQEI_UNLIMITED

Feedback
MQFB_NONE

Codifica
MQENC.Native (la codificazione del sistema z/OS)

CodedCharSetId
MQCCSI_Q_MGR (ilSetID CodedChardel sistema z/OS)

Formato
MQFMT_IMS se MQMD.Format del messaggio di input è MQFMT_IMS, altrimenti IOPCB.MODNAME

Priorità
MQMD.Priority del messaggio di input

Persistenza
Dipende dalla modalità di commit: MQMD.Persistence del messaggio di input se CM-1; persistenza corrisponde alla recuperabilità del messaggio IMS se CM-0

MsgId
MQMD.MsgId se MQRO_PASS_MSG_ID, altrimenti New MsgId (impostazione predefinita)

CorrelId
MQMD.CorrelId dal messaggio di input se MQRO_PASS_CORREL_ID, altrimenti MQMD.MsgId dal messaggio di input (impostazione predefinita)

BackoutCount
0

ReplyToQ
Spazi

ReplyToQMgr
Spazi (impostati sul nome qmgr locale dal gestore code durante MQPUT)

UserIdentifier
MQMD.UserIdentifier del messaggio di input

AccountingToken
MQMD.AccountingToken del messaggio di input

ApplIdentityData
MQMD.ApplIdentityData del messaggio di input

PutApplType
MQAT_XCF se non si verifica alcun errore, altrimenti MQAT_BRIDGE

PutApplName
<XCFgroupName> <XCFmemberName> se non si verifica alcun errore, altrimenti il nome QMGR

PutDate
Data in cui il messaggio è stato inserito

PutTime
Ora in cui il messaggio è stato inserito

ApplOriginData
Spazi

z/OS *I campi MQIIH nei messaggi dal bridge IMS*

Informazioni sui campi MQIIH nei messaggi dal bridge IMS .

L'MQIIH di un messaggio ricevuto da IMS viene creato come segue:

StrucId
"IIH"

Versione
1

StrucLength
84

Codifica
MQEN_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Formato
MQIIH.ReplyToFormat del messaggio di input se MQIIH.ReplyToFormat non è vuoto, altrimenti IOPCB.MODNAME

Indicatori
0

LTermOverride
Nome LTERM (Tpipe) dall'intestazione OTMA

MFSMapName
Nome associazione dall'intestazione OTMA

ReplyToFormat
Spazi

Programma di autenticazione
MQIIH.Authenticator del messaggio di input se il messaggio di risposta viene inserito in una coda bridge MQ-IMS , altrimenti spazi vuoti.

TranInstanceId
ID conversazione / Token server dall'intestazione OTMA se in conversazione. Nelle versioni di IMS precedenti a V14, questo campo è sempre null se non è in conversazione. Da IMS V14 in poi, questo campo può essere impostato da IMS anche se non è in conversazione.

TranState
"C" se nella conversazione, altrimenti vuoto

CommitMode
Modalità di commit dall'intestazione OTMA ("0" o "1")

SecurityScope
Spazio

Riservato
Spazio

z/OS *Messaggi di risposta da IMS*

Quando una transazione IMS ISRTs al suo IOPCB, il messaggio viene reinstradato all'LTERM o TPIPE di origine.

Questi sono visualizzati in IBM MQ come messaggi di risposta. I messaggi di risposta da IMS vengono inseriti nella coda di risposta specificata nel messaggio originale. Se il messaggio non può essere inserito nella coda di risposta, viene inserito nella coda di messaggi non recapitabili utilizzando l'autorità del bridge. Se il messaggio non può essere inserito nella coda di messaggi non recapitabili, viene inviato un riconoscimento negativo a IMS per indicare che non è possibile ricevere il messaggio. La responsabilità del messaggio viene quindi restituita a IMS. Se si utilizza la modalità di commit 0, i messaggi da tale Tpipe non vengono inviati al bridge e rimangono nella coda IMS ; in altre parole, non vengono inviati ulteriori messaggi fino al riavvio. Se si sta utilizzando la modalità di commit 1, l'altro lavoro può continuare.

Se la risposta ha una struttura MQIIH, il suo tipo di formato è MQFMT_IMS; in caso contrario, il suo tipo di formato è specificato dal nome MOD IMS utilizzato durante l'inserimento del messaggio.

z/OS *Utilizzo di PCB di risposta alternativi in transazioni IMS*

Quando una transazione IMS utilizza PCB a risposta alternativa (ISRT a ALTPCB o emette una chiamata CHNG a un PCB modificabile), viene richiamata l'exit di pre - instradamento (DFSYPRX0) per stabilire se il messaggio deve essere reinstradato.

Se il messaggio deve essere reinstradato, l'uscita di risoluzione della destinazione (DFSYDRU0) viene richiamata per confermare la destinazione e preparare le informazioni di intestazione. Consultare Utilizzo delle uscite OTMA in IMS e L'uscita di pre - instradamento DFSYPRX0 per informazioni su questi programmi di uscita.

A meno che non venga eseguita un'azione nelle uscite, tutti gli output dalle IMS transazioni avviate da un gestore code IBM MQ , sia nell'IOPCB che in un ALTPCB, verranno restituiti allo stesso gestore code.

z/OS *Invio di messaggi non richiesti da IMS*

Per inviare i messaggi da IMS a una coda IBM MQ , è necessario richiamare una transazione IMS ISRT a un ALTPCB.

È necessario scrivere uscite di pre - instradamento e di risoluzione della destinazione per instradare i messaggi non richiesti da IMS e creare i dati utente OTMA, in modo che l'MQMD del messaggio possa essere creato correttamente. Consultare L'uscita di preinstradamento DFSYPRX0 e L'uscita utente di risoluzione di destinazione per informazioni su questi programmi di uscita.

Nota: Il bridge IBM MQ - IMS non sa se un messaggio ricevuto è una risposta o un messaggio non richiesto. Gestisce il messaggio nello stesso modo in ogni singolo caso, creando MQMD e MQIIH della risposta in base all'OTMA UserData arrivato con il messaggio

I messaggi non richiesti possono creare nuovi Tpipe. Ad esempio, se una transazione IMS esistente passa a un nuovo LTERM (ad esempio PRINT01), ma l'implementazione richiede che l'output venga distribuito tramite OTMA, viene creato un nuovo Tpipe (denominato PRINT01 in questo esempio). Per impostazione predefinita, questo è un Tpipe non sincronizzato. Se l'implementazione richiede che il messaggio sia ripristinabile, impostare l'indicatore di output dell'uscita di risoluzione di destinazione. Per ulteriori informazioni, consultare il manuale *IMS Customization Guide* .

z/OS *Segmentazione del messaggio*

È possibile definire le transazioni IMS come se prevedesse un input a segmento singolo o multiplo.

L'applicazione IBM MQ di origine deve creare l'input utente che segue la struttura MQIIH come uno o più segmenti LLZZ - data. Tutti i segmenti di un messaggio IMS devono essere contenuti in un singolo messaggio IBM MQ inviato con un singolo MQPUT.

La lunghezza massima di un segmento di dati LLZZ è definita da IMS/OTMA (32767 byte). La lunghezza totale del messaggio IBM MQ è la somma dei byte LL, più la lunghezza della struttura MQIIH.

Tutti i segmenti della risposta sono contenuti in un unico messaggio IBM MQ .

Esiste un'ulteriore limitazione alla limitazione di 32 KB sui messaggi con formato MQFMT_IMS_VAR_STRING. Quando i dati in un messaggio CCSID misto ASCII vengono convertiti in un messaggio CCSID misto EBCDIC, viene aggiunto un byte di fine stringa o un byte di inizio stringa ogni volta che si verifica una transizione tra i caratteri SBCS e DBCS. La limitazione di 32 KB si applica alla dimensione massima del messaggio. Ciò significa che, poiché il campo LL nel messaggio non può superare i 32 KB, il messaggio non deve superare i 32 KB inclusi tutti i caratteri di inizio e fine stringa. L'applicazione che crea il messaggio deve consentirlo.

z/OS *Conversione dei dati per i messaggi da e verso il bridge IMS*

La conversione dei dati viene eseguita dalla funzione di accodamento distribuito (che può richiamare le uscite necessarie) o dall'agente di accodamento all'interno del gruppo (che non supporta l'utilizzo delle uscite) quando inserisce un messaggio in una coda di destinazione che ha le informazioni XCF definite per la relativa classe di memoria. La conversione dei dati non si verifica quando un messaggio viene consegnato a una coda mediante pubblicazione / sottoscrizione.

Tutte le uscite necessarie devono essere disponibili per la funzione di accodamento distribuito nel dataset a cui fa riferimento l'istruzione CSQXLIB DD. Ciò significa che è possibile inviare messaggi a una applicazione IMS utilizzando il bridge IBM MQ - IMS da qualsiasi piattaforma IBM MQ .

Se si verificano errori di conversione, il messaggio viene inserito nella coda non convertita; ciò fa sì che venga considerato come un errore dal bridge IBM MQ - IMS , poiché il bridge non è in grado di riconoscere il formato dell'intestazione. Se si verifica un errore di conversione, viene inviato un messaggio di errore alla console z/OS .

Consultare “Scrittura delle uscite di conversione dati” a pagina 985 per informazioni dettagliate sulla conversione dei dati in generale.

Invio di messaggi al bridge IBM MQ - IMS

Per garantire che la conversione venga eseguita correttamente, è necessario indicare al gestore code il formato del messaggio.

Se il messaggio ha una struttura MQIIH, il *Format* in MQMD deve essere impostato sul formato integrato MQFMT_IMS e il *Format* in MQIIH deve essere impostato sul nome del formato che descrive i propri dati del messaggio. Se non è presente alcun MQIIH, impostare il *Format* in MQMD sul proprio nome formato.

Se i dati (diversi da LLZZs) sono tutti dati carattere (MQCHAR), utilizzare come nome formato (in MQIIH o MQMD, a seconda dei casi) il formato integrato MQFMT_IMS_VAR_STRING. Altrimenti, utilizzare il proprio nome formato, nel cui caso è necessario fornire anche un'uscita di conversione dati per il formato. L'uscita deve gestire la conversione degli LLZZs nel tuo messaggio, oltre ai dati stessi (ma non deve gestire alcun MQIIH all'inizio del messaggio).

Se l'applicazione utilizza *MFSMapName*, è possibile utilizzare i messaggi con MQFMT_IMS e definire il nome della mappa passato alla transazione IMS nel campo MFSMapName di MQIIH.

Ricezione di messaggi dal bridge IBM MQ - IMS

Se una struttura MQIIH è presente sul messaggio originale che si sta inviando a IMS, ne è presente anche uno sul messaggio di replica.

Per assicurarsi che la propria risposta sia convertita correttamente:

- Se si dispone di una struttura MQIIH sul messaggio originale, specificare il formato che si desidera per il messaggio di risposta nel campo MQIIH *ReplytoFormat* del messaggio originale. Questo valore viene inserito nel campo MQIIH *Format* del messaggio di risposta. Ciò è particolarmente utile se tutti i dati di output sono nel formato LLZZ < dati carattere>.
- Se non si dispone di una struttura MQIIH sul messaggio originale, specificare il formato che si desidera per il messaggio di risposta come nome MFS MOD nell'ISRT dell'applicazione IMS per IOPCB.

z/OS *Scrittura di programmi di transazione IMS tramite IBM MQ*

La codifica richiesta per gestire le transazioni IMS tramite IBM MQ dipende dal formato del messaggio richiesto dalla transazione IMS e dall'intervallo di risposte che può restituire. Tuttavia, ci sono diversi punti da considerare quando la tua applicazione gestisce le informazioni di formattazione dello schermo IMS .

Quando una transazione IMS viene avviata da uno schermo 3270, il messaggio passa attraverso IMS Message Format Services. In questo modo è possibile rimuovere tutte le dipendenze del terminale dal flusso di dati visualizzato dalla transazione. Quando una transazione viene avviata tramite OTMA, MFS non viene coinvolto. Se la logica dell'applicazione è implementata in MFS, è necessario ricrearla nella nuova applicazione.

In alcune transazioni IMS , l'applicazione dell'utente finale può modificare alcuni comportamenti dello schermo 3270, ad esempio, evidenziando un campo in cui sono stati immessi dati non validi. Questo tipo di informazioni viene comunicato aggiungendo un campo attributo a due byte al messaggio IMS per ogni campo dello schermo che deve essere modificato dal programma.

Pertanto, se si sta codificando un'applicazione per imitare un 3270, è necessario tenere conto di questi campi quando si creano o si ricevono messaggi.

Potrebbe essere necessario codificare le informazioni nel programma per elaborare:

- Il tasto premuto (ad esempio, Invio e PF1)
- Dove si trova il cursore quando il messaggio viene passato all'applicazione
- Se i campi attributo sono stati impostati dall'applicazione IMS
  - Intensità alta, normale o zero
  - Colore
  - Se IMS prevede che il campo torni indietro la volta successiva che viene premuto Invio

- Se l'applicazione IMS ha utilizzato caratteri null (X'3F') in qualsiasi campo.

Se il messaggio IMS contiene solo dati carattere (tranne il segmento LLZZ - data) e si sta utilizzando una struttura MQIIH, impostare il formato MQMD su MQFMT_IMS e il formato MQIIH su MQFMT_IMS_VAR_STRING.

Se il messaggio IMS contiene solo dati carattere (ad eccezione del segmento LLZZ - data) e **non** si utilizza una struttura MQIIH, impostare il formato MQMD su MQFMT_IMS_VAR_STRING e assicurarsi che l'applicazione IMS specifichi MODname MQFMT_IMS_VAR_STRING durante la risposta. Se si verifica un problema (ad esempio, l'utente non è autorizzato a utilizzare la transazione) e IMS invia un messaggio di errore, questo ha un MODname nel formato DFSMOx, dove x è un numero compreso tra 1 e 5. Viene inserito in MQMD.Format.

Se il messaggio IMS contiene dati binari, compressi o a virgola mobile (ad eccezione del segmento dati LLZZ), codificare le proprie routine di conversione dati. Fare riferimento a *IMS/ESA Application Programming: Transaction Manager* per informazioni sulla formattazione delle schermate IMS .

Considerare i seguenti argomenti quando si scrive il codice per gestire le transazioni IMS tramite IBM MQ.

- "Scrittura di applicazioni IBM MQ per richiamare transazioni di conversazione IMS" a pagina 914
- "Scrittura di programmi contenenti comandi IMS" a pagina 914
- "Triggering" a pagina 915

Scrittura di applicazioni IBM MQ per richiamare transazioni di conversazione IMS

Utilizzare queste informazioni come guida per le considerazioni sulla scrittura dell'applicazione IBM MQ per richiamare le transazioni conversazionali IMS .

Quando scrivi un'applicazione che richiama una conversazione IMS , considera quanto segue:

- Includere una struttura MQIIH con il messaggio dell'applicazione.
- Impostare *CommitMode* in MQIIH su MQICM_SEND_THEN_COMMIT.
- Per richiamare una nuova conversazione, impostare *TranState* in MQIIH su MQITS_NOT_IN_CONVERSATION.
- Per richiamare la seconda e le fasi successive di una conversazione, impostare *TranState* su MQITS_IN_CONVERSATION e impostare *TranInstanceId* sul valore di tale campo restituito nel passo precedente della conversazione.
- Non esiste un modo semplice in IMS per trovare il valore di un *TranInstanceId*, se si perde il messaggio originale inviato da IMS.
- L'applicazione deve controllare il *TranState* dei messaggi da IMS per controllare se la transazione IMS ha terminato la conversazione.
- È possibile utilizzare /EXIT per terminare una conversazione. È inoltre necessario racchiudere tra virgolette *TranInstanceId*, impostare *TranState* su MQITS_IN_CONVERSATION e utilizzare la coda IBM MQ su cui viene eseguita la conversazione.
- Non è possibile utilizzare /HOLD o /REL per congelare o rilasciare una conversazione.
- Le conversazioni richiamate tramite il bridge IBM MQ - IMS vengono terminate se IMS viene riavviato.

Scrittura di programmi contenenti comandi IMS

Un programma applicativo può creare un IBM MQ messaggio nel formato LLZZ*comando*, invece di una transazione, dove *comando* è nel formato /DIS TRAN PART o /DIS POOL ALL.

La maggior parte dei comandi IMS può essere eseguita in questo modo; per i dettagli, consultare *IMS V11 Communications and Connections* . L'output del comando viene ricevuto nel messaggio di replica IBM MQ nel formato di testo come verrebbe inviato a un terminale 3270 per la visualizzazione.

OTMA ha implementato una forma speciale del comando di visualizzazione della transazione IMS , che restituisce una forma architettata dell'output. Il formato esatto è definito in *IMS V11 Comunicazioni e connessioni*. Per richiamare questo modulo da un messaggio IBM MQ , creare i dati del messaggio come

prima, ad esempio /DIS TRAN PART, e impostare il campo TranState in MQIIH su MQITS_ARCHITECTED. IMS elabora il comando e restituisce la risposta nel modulo progettato. Una risposta progettata contiene tutte le informazioni che possono essere trovate nel formato di testo dell'output e un'altra parte di informazioni: se la transazione è definita come recuperabile o non recuperabile.

Triggering

Il bridge IBM MQ - IMS non supporta i messaggi trigger.

Se si definisce una coda di avvio che utilizza una classe di archiviazione con parametri XCF, i messaggi inseriti in tale coda vengono rifiutati quando si arriva al bridge.

Scrittura di applicazioni procedurali client

Cosa devi sapere per scrivere le applicazioni client su IBM MQ utilizzando un linguaggio procedurale.

Le applicazioni possono essere create ed eseguite nell'ambiente client IBM MQ . L'applicazione deve essere creata e collegata al IBM MQ MQI client utilizzato. Il modo in cui le applicazioni vengono create e collegate varia in base alla piattaforma e al linguaggio di programmazione utilizzato. Per informazioni su come creare applicazioni client, consultare “Creazione di applicazioni per IBM MQ MQI clients” a pagina 921.

Puoi eseguire un'applicazione IBM MQ sia in un ambiente IBM MQ completo che in un ambiente IBM MQ MQI client senza modificare il codice, purché vengano soddisfatte determinate condizioni. Per ulteriori informazioni sull'esecuzione delle applicazioni nell'ambiente client IBM MQ , consultare “Esecuzione di applicazioni nell'ambiente di IBM MQ MQI client” a pagina 923.

Se si utilizza l'interfaccia MQI (message queue interface) per scrivere le applicazioni da eseguire in un ambiente IBM MQ MQI client , esistono alcuni controlli aggiuntivi da imporre durante una chiamata MQI per garantire che l'elaborazione dell'applicazione IBM MQ non venga interrotta. Per ulteriori informazioni su questi controlli, consultare “Utilizzo di MQI in un'applicazione client” a pagina 916.

Consultare i seguenti argomenti per informazioni sulla preparazione ed esecuzione di altri tipi di applicazione come applicazioni client:

- “Preparazione ed esecuzione di applicazioni CICS e Tuxedo” a pagina 935
- “Preparazione ed esecuzione di applicazioni di Microsoft Transaction Server” a pagina 50
- “Preparazione ed esecuzione delle applicazioni IBM MQ JMS” a pagina 938

Concetti correlati

“Concetti dello sviluppo di applicazioni” a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

“Sviluppo di applicazioni per IBM MQ” a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

“Considerazioni sulla progettazione per applicazioni IBM MQ” a pagina 51
Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

“Scrittura di un'applicazione procedurale per l'accodamento” a pagina 721
Utilizzare queste informazioni per informazioni sulla scrittura delle applicazioni di accodamento, la connessione e la disconnessione da un gestore code, la pubblicazione / sottoscrizione e l'apertura e la chiusura di oggetti.

“Scrittura di applicazioni di pubblicazione / sottoscrizione” a pagina 809
Avviare la scrittura delle applicazioni IBM MQ di pubblicazione / sottoscrizione.

“Creazione di un'applicazione procedurale” a pagina 1002

È possibile scrivere un'applicazione IBM MQ in uno dei diversi linguaggi procedurali ed eseguire l'applicazione su diverse piattaforme.

“Gestione degli errori del programma procedurale” a pagina 1040
Queste informazioni spiegano gli errori associati alle chiamate MQI delle applicazioni quando effettua una chiamata o quando il suo messaggio viene consegnato alla destinazione finale.

Attività correlate

“Utilizzo dei programmi procedurali di esempio IBM MQ” a pagina 1060
Questi programmi di esempio sono scritti in linguaggi procedurali e dimostrano gli usi tipici di MQI (Message Queue Interface). Programmi IBM MQ su diverse piattaforme.

Utilizzo di MQI in un'applicazione client

Questa raccolta di argomenti considera le differenze tra la scrittura dell'applicazione IBM MQ da eseguire in un ambiente client MQI (Message Queue Interface) e da eseguire nell'intero ambiente del gestore code IBM MQ .

Quando si progetta un'applicazione, considerare quali controlli è necessario imporre durante una chiamata MQI per assicurarsi che l'elaborazione dell'applicazione IBM MQ non venga interrotta.

Prima di poter eseguire applicazioni che utilizzano MQI, è necessario creare determinati oggetti IBM MQ . Per ulteriori informazioni, consultare Programmi applicativi che utilizzano MQI.

Limitazione della dimensione di un messaggio in una applicazione client

Un gestore code ha una lunghezza massima del messaggio, ma la dimensione massima del messaggio che è possibile trasmettere da una applicazione client è limitata dalla definizione del canale.

L'attributo lunghezza massima del messaggio (MaxMsgLength) di un gestore code è la lunghezza massima di un messaggio che può essere gestito da tale gestore code.

Multi Su Multiplatforms, è possibile incrementare l'attributo della lunghezza massima dei messaggi di un gestore code. Per ulteriori informazioni, consultare ALTER QMGR.

È possibile rilevare il valore di MaxMsgLength per un gestore code utilizzando la chiamata MQINQ.

Se l'attributo di lunghezza MaxMsgviene modificato, non viene effettuato alcun controllo che non vi siano già code e nemmeno messaggi con una lunghezza superiore al nuovo valore. Dopo aver modificato questo attributo, riavviare le applicazioni e i canali per essere certi che la modifica sia diventata effettiva. Non è quindi possibile generare nuovi messaggi che superano la lunghezza MaxMsgdel gestore code o della coda (a meno che non sia consentita la segmentazione del gestore code).

La lunghezza massima del messaggio in una definizione del canale limita la dimensione di un messaggio che è possibile trasmettere lungo una connessione client. Se un'applicazione IBM MQ tenta di utilizzare la chiamata MQPUT o MQGET con un messaggio più grande di questo, viene restituito un codice di errore all'applicazione. Il parametro della dimensione massima del messaggio della definizione del canale non influenza la dimensione massima del messaggio che può essere utilizzata utilizzando MQBC su una connessione client.

Concetti correlati

“Utilizzo di MQCONNX” a pagina 920
È possibile utilizzare la chiamata MQCONNX per specificare una struttura MQCD (channel definition) nella struttura MQCNO.

Riferimenti correlati

Lunghezza massima messaggio (MAXMSGL)
MODIFICA CANALE
2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

Scelta del CCSID del client o del server

Utilizzare il CCSID (coded character set identifier) locale per il client. Il gestore code esegue la conversione necessaria. È possibile utilizzare la variabile di ambiente **MQCCSID** per sovrascrivere il CCSID.

Se l'applicazione esegue più PUT, i campi CCSID e di codifica di MQMD possono essere sovrascritti dopo il completamento del primo PUT.

I dati trasmessi attraverso l'interfaccia MQI (message queue interface) dall'applicazione allo stub client devono essere nel CCSID locale, codificato per IBM MQ MQI client. Se il gestore code connesso richiede la conversione dei dati, la conversione viene eseguita dal codice di supporto client sul gestore code.

In IBM WebSphere MQ 7.0 e versioni successive, il client Java può eseguire la conversione se il gestore code non è in grado di farlo. Consultare “IBM MQ classes for Java Connessioni client” a pagina 375.

Il codice client presuppone che i dati carattere che attraversano l'MQI nel client si trovano nel CCSID configurato per quella workstation. Se questo CCSID non è supportato o non è il CCSID richiesto, può essere sovrascritto con la variabile di ambiente **MQCCSID** utilizzando uno dei seguenti comandi:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Se questo parametro è impostato nel profilo, si presuppone che tutti i dati MQI si trovano nella codepage 850.

Nota: Il presupposto relativo alla codepage 850 non si applica ai dati dell'applicazione nel messaggio.

Se l'applicazione sta eseguendo più PUT che includono le intestazioni IBM MQ dopo il descrittore del messaggio (MQMD), tenere presente che i campi CCSID e di codifica di MQMD vengono sovrascritti dopo il completamento del primo PUT.

Dopo il primo PUT, questi campi contengono il valore utilizzato dal gestore code connesso per convertire le intestazioni IBM MQ . Verificare che l'applicazione reimposti i valori sui valori richiesti.

Utilizzo di MQINQ in un'applicazione client

Alcuni valori interrogati utilizzando MQINQ vengono modificati dal codice client.

CCSID
è impostato sul CCSID client, non su quello del gestore code.

MaxMsgLength
viene ridotto se è limitato dalla definizione di canale. Questo sarà il valore più basso tra:

- Il valore definito nella definizione della coda oppure
- Il valore definito nella definizione di canale

Per ulteriori informazioni, consultare MQINQ.

Utilizzo del coordinamento del punto di sincronizzazione in un'applicazione client

Un'applicazione in esecuzione su un client di base può emettere MQCMIT e MQBACK, ma l'ambito del controllo del punto di sincronizzazione è limitato alle risorse MQI. È possibile utilizzare un gestore transazioni esterno con un client transazionale esteso.

All'interno di IBM MQ, uno dei ruoli del gestore code è il controllo del punto di sincronizzazione all'interno di un'applicazione. Se un'applicazione viene eseguita su un client di base IBM MQ , può emettere MQCMIT e MQBACK, ma l'ambito del controllo del punto di sincronizzazione è limitato alle risorse MQI. Il comando IBM MQ MQBEGIN non è valido in un ambiente client di base.

Le applicazioni in esecuzione nell'ambiente del gestore code completo sul server possono coordinare più risorse (ad esempio i database) tramite un monitoraggio delle transazioni. Sul server è possibile utilizzare il monitoraggio transazioni fornito con i prodotti IBM MQ o un altro monitoraggio transazioni come CICS. Non è possibile utilizzare un monitoraggio transazioni con un'applicazione client base.

È possibile utilizzare un gestore transazioni esterno con un IBM MQ client transazionale esteso. Consultare Cos' è un client transazionale esteso? per ulteriori dettagli.

Utilizzo della lettura anticipata in un'applicazione client

È possibile utilizzare la lettura anticipata su un client per consentire l'invio di messaggi non persistenti a un client senza che l'applicazione client debba richiedere i messaggi.

Quando un client richiede un messaggio da un server, invia una richiesta al server. Invia una richiesta separata per ciascuno dei messaggi che utilizza. Per migliorare le prestazioni di un client che utilizza messaggi non persistenti evitando di dover inviare questi messaggi di richiesta, è possibile configurare un client per utilizzare la lettura anticipata. La lettura anticipata consente l'invio di messaggi a un cliente senza che un'applicazione debba richiederli.

L'utilizzo della lettura anticipata può migliorare le prestazioni quando si utilizzano messaggi non persistenti da un'applicazione client. Questo miglioramento delle prestazioni è disponibile per le applicazioni MQI e JMS . Le applicazioni client che utilizzano MQGET o l'utilizzo asincrono traggono vantaggio dai miglioramenti delle prestazioni quando si utilizzano messaggi non persistenti.

Quando si richiama MQOPEN con MQOO_READ_AHEAD, il client IBM MQ abilita la lettura anticipata solo se sono soddisfatte determinate condizioni. Queste condizioni includono:

- L'applicazione client deve essere compilata e collegata rispetto alle librerie client MQI IBM MQ in thread.
- Il canale del client deve utilizzare il protocollo TCP/IP
- Il canale deve avere un'impostazione SharingConversations (SHARECNV) diversa da zero nelle definizioni di canale del client e del server.

Quando la lettura anticipata è abilitata, i messaggi vengono inviati a un buffer di memoria sul client denominato buffer di lettura anticipata. Il client dispone di un buffer di lettura anticipata per ogni coda aperta con lettura anticipata abilitata. I messaggi nel buffer di lettura anticipata non sono persistenti. Il client aggiorna periodicamente il server con informazioni sulla quantità di dati che ha utilizzato.

Non tutte le progettazioni di applicazioni client sono adatte per l'utilizzo della lettura anticipata poiché non tutte le opzioni sono supportate per l'utilizzo. È necessario che alcune opzioni siano congruenti tra le chiamate MQGET quando la lettura anticipata è abilitata. Se un client modifica i criteri di selezione tra le chiamate MQGET, i messaggi memorizzati nel buffer di lettura anticipata rimangono bloccati nel buffer di lettura anticipata del client. Per ulteriori informazioni, vedi “Miglioramento delle prestazioni dei messaggi non persistenti” a pagina 787

La configurazione read ahead è controllata da tre attributi, MaximumSize, PurgeTimee UpdatePercentage, specificati nella stanza MessageBuffer del file di configurazione client IBM MQ .

Utilizzo dell'inserimento asincrono in un'applicazione client

Utilizzando l'inserimento asincrono, un'applicazione può inserire un messaggio in una coda senza attendere una risposta dal gestore code. È possibile utilizzarlo per migliorare le prestazioni della messaggistica in alcune situazioni.

Di solito, quando un'applicazione inserisce uno o più messaggi in una coda, utilizzando MQPUT o MQPUT1, l'applicazione deve attendere che il gestore code confermi di aver elaborato la richiesta MQI. È possibile migliorare le prestazioni della messaggistica, in modo particolare per le applicazioni che utilizzano i bind del client e per le applicazioni che inserono un numero elevato di messaggi di piccole dimensioni in una coda, scegliendo invece di inserire i messaggi in modo asincrono. Quando un'applicazione inserisce un messaggio in maniera asincrona, il gestore code non restituisce l'esito positivo o negativo di ciascuna chiamata, ma è possibile controllare periodicamente la presenza di errori.

Per inserire un messaggio in una coda in modo asincrono, utilizzare l'opzione MQPMO_ASYNC_RESPONSE nel campo *Options* della struttura MQPMO.

Se un messaggio non è idoneo per l'inserimento asincrono, viene inserito in una coda in modo sincrono.

Quando si richiede una risposta di inserimento asincrono per MQPUT o MQPUT1, un CompCode e motivo di MQCC_OK e MQRC_NONE non significa necessariamente che il messaggio è stato inserito correttamente in una coda. Anche se l'esito positivo o negativo di ogni singola chiamata MQPUT o MQPUT1 potrebbe non essere restituito immediatamente, il primo errore che si è verificato in una chiamata asincrona può essere determinato successivamente tramite una chiamata a MQSTAT.

Per ulteriori dettagli su MQPMO_ASYNC_RESPONSE, consultare Opzioni MQPMO.

Il programma di esempio Put asincrono dimostra alcune funzioni disponibili. Per i dettagli sulle caratteristiche e la progettazione del programma e su come eseguirlo, consultare "Il programma di esempio Put asincrono" a pagina 1080.

Utilizzo delle conversazioni di condivisione in un'applicazione client

In un ambiente in cui la condivisione delle conversazioni è consentita, le conversazioni possono condividere un'istanza del canale MQI.

La condivisione delle conversazioni è controllata da due campi, entrambi denominati SharingConversations, uno dei quali fa parte della struttura MQCD (channel definition) e uno dei quali fa parte della struttura MQCXP (channel exit parameter). Il campo SharingConversations in MQCD è un valore intero che determina il numero massimo di conversazioni che possono condividere un'istanza del canale associata al canale. Il campo SharingConversations in MQCXP è un valore booleano, che indica se l'istanza del canale è attualmente condivisa.

In un ambiente in cui la condivisione delle conversazioni non è consentita, le nuove connessioni client che specificano MQCD identici non condivideranno un'istanza del canale.

Una nuova connessione dell'applicazione client condividerà l'istanza del canale quando si verificano le seguenti condizioni:

- Le estremità di connessione client e server dell'istanza del canale sono configurate per la condivisione delle conversazioni e questi valori non vengono sovrascritti dalle uscite del canale.
- Il valore MQCD della connessione client (fornito sulla chiamata MQCONNX del client o dalla tabella di definizione del canale client (CCDT)) corrisponde esattamente al valore MQCD della connessione client fornito sulla chiamata MQCONNX del client o dalla CCDT quando è stata stabilita per la prima volta l'istanza del canale esistente. Notare che l'MQCD originale potrebbe essere stato successivamente modificato dalle uscite o dalla negoziazione del canale, ma che la corrispondenza viene effettuata rispetto al valore fornito al sistema client prima che queste modifiche siano state apportate.
- Il limite di conversazioni di condivisione sul lato server non è stato superato.

Se una nuova connessione dell'applicazione client corrisponde ai criteri per eseguire la condivisione di un'istanza del canale con altre conversazioni, questa decisione viene presa prima che vengano richiamate le uscite su tale conversazione. Le uscite su una conversazione di questo tipo non possono alterare il fatto che sta condividendo l'istanza del canale con altre conversazioni. Se non ci sono istanze di canale esistenti che corrispondono alla nuova definizione di canale, viene connessa una nuova istanza di canale.

La negoziazione del canali si verifica solo per la prima conversazione su un'istanza del canale; i valori negoziati per l'istanza del canale sono fissi in quella fase e non possono essere modificati all'avvio delle conversazioni successive. L'autenticazione TLS si verifica anche solo per la prima conversazione.

Se il valore di MQCD SharingConversations viene modificato durante l'inizializzazione di qualsiasi uscita di sicurezza, di invio o di ricezione per la prima conversazione sul socket alla connessione client o all'estremità di connessione server dell'istanza del canale, il nuovo valore che ha dopo l'inizializzazione di tutte queste uscite viene utilizzato per determinare il valore delle conversazioni di condivisione per l'istanza del canale (il valore più basso ha la precedenza).

Se il valore negoziato per la condivisione delle conversazioni è zero, l'istanza del canale non viene mai condivisa. Ulteriori programmi di uscita che impostano questo campo su zero vengono eseguiti in modo simile sulla propria istanza del canale.

Se il valore negoziato per la condivisione delle conversazioni è maggiore di zero, MQCXP SharingConversations è impostato su TRUE per le chiamate successive alle uscite, indicando che altri

programmi di uscita su questa istanza del canale possono essere immessi contemporaneamente a questo.

Quando si scrive un programma di uscita del canale, considerare se verrà eseguito su un'istanza del canale che potrebbe implicare la condivisione delle conversazioni. Se l'istanza del canale potrebbe implicare la condivisione di conversazioni, considerare l'effetto sulle altre istanze dell'uscita del canale della modifica dei campi MQCD; tutti i campi MQCD hanno valori comuni in tutte le conversazioni di condivisione. Una volta stabilita l'istanza del canale, se i programmi di uscita tentano di modificare i campi MQCD, potrebbero riscontrare dei problemi perché altre istanze di programmi di uscita in esecuzione sull'istanza del canale potrebbero tentare di modificare gli stessi campi contemporaneamente. Se questa situazione può verificarsi con i programmi di uscita, è necessario serializzare l'accesso al MQCD nel codice di uscita.

Se si utilizza un canale definito per condividere le conversazioni, ma non si desidera che la condivisione avvenga su una particolare istanza del canale, impostare il valore MQCD di SharingConversations su 1 o 0 quando si inizializza un'uscita del canale sulla prima conversazione sull'istanza del canale. Consultare SharingConversations per una spiegazione dei valori di SharingConversations.

Esempio

La condivisione delle conversazioni è abilitata.

Si sta utilizzando una definizione di canale di collegamento client che specifica un programma di uscita.

La prima volta che questo canale viene avviato, il programma di uscita modifica alcuni parametri MQCD quando viene inizializzato. Questi sono gestiti dal canale, quindi la definizione con cui è in esecuzione il canale è ora differente da quella fornita originariamente. Il parametro MQCXP SharingConversations è impostato su TRUE.

La volta successiva in cui l'applicazione si connette utilizzando questo canale, la conversazione viene eseguita sull'istanza del canale che è stata avviata precedentemente, perché ha la stessa definizione di canale originale. L'istanza del canale a cui l'applicazione si connette la seconda volta è la stessa istanza della prima volta che si connette. Di conseguenza, utilizza le definizioni che sono state modificate dal programma di uscita. Quando il programma di uscita viene inizializzato per la seconda conversazione, anche se può modificare i campi MQCD, non vengono utilizzati dal canale. Queste stesse caratteristiche si applicano a tutte le conversazioni successive che condividono l'istanza del canale.

Utilizzo di MQCONNX

È possibile utilizzare la chiamata MQCONNX per specificare una struttura MQCD (channel definition) nella struttura MQCNO.

Ciò consente all'applicazione client chiamante di specificare la definizione del canale di connessione client al runtime. Per ulteriori informazioni, consultare Creazione di un canale di connessione client su IBM MQ MQI client utilizzando MQCNO. Quando si utilizza MQCONNX, la chiamata emessa sul server dipende dal livello server e dalla configurazione del listener.

Quando si utilizza MQCONNX da un client, le seguenti opzioni vengono ignorate:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

La struttura MQCD che è possibile utilizzare dipende dal numero di versione MQCD che si sta utilizzando. Per informazioni sulle versioni MQCD (MQCD_VERSION), consultare Versione MQCD. È possibile utilizzare la struttura MQCD, ad esempio, per passare i programmi di uscita canale al server. Se si utilizza MQCD Versione 3 o successiva, è possibile utilizzare la struttura per passare un array di uscite al server. È possibile utilizzare questa funzione per eseguire più di un'operazione sullo stesso messaggio, come la codifica e la compressione, aggiungendo un'uscita per ciascuna operazione, piuttosto che modificare un'uscita esistente. Se non si specifica un array nella struttura MQCD, verranno controllati i singoli campi di uscita. Per ulteriori informazioni sui programmi di uscita del canale, consultare “Programmi di uscita canale per canali di messaggistica” a pagina 963.

Handle di connessione condivisi su MQCONNX
È possibile condividere handle tra thread differenti all'interno dello stesso processo, utilizzando handle di connessione condivisi.

Quando si specifica un handle di connessione condiviso, l'handle di connessione restituito dalla chiamata MQCONNX può essere passato nelle chiamate MQI successive su qualsiasi thread nel processo.

Nota: È possibile utilizzare un handle di connessione condiviso su un IBM MQ MQI client per connettersi a un gestore code del server che non supporta gli handle di connessione condivisi.

Per ulteriori informazioni, consultare “Utilizzo di MQCONNX” a pagina 920.

Creazione di applicazioni per IBM MQ MQI clients

Le applicazioni possono essere create ed eseguite in ambiente IBM MQ MQI client . L'applicazione deve essere creata e collegata al IBM MQ MQI client utilizzato. Il modo in cui le applicazioni vengono create e collegate varia in base alla piattaforma e al linguaggio di programmazione utilizzato.

Se un'applicazione deve essere eseguita in un ambiente client, è possibile scriverla nelle lingue mostrate nella seguente tabella:

Tabella 133. Linguaggi di programmazione supportati in ambienti client

Piattaforma client	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Sì	Sì	Sì			
IBM i	Sì		Sì		Sì	
Linux	Sì	Sì	Sì			
Windows	Sì	Sì	Sì			Sì

Multi *Collegamento di applicazioni C con il codice IBM MQ MQI client*

Dopo aver scritto la tua applicazione IBM MQ che vuoi eseguire su IBM MQ MQI client, devi collegarla al codice IBM MQ MQI client .

È possibile collegare l'applicazione al codice IBM MQ MQI client in due modi:

1. Direttamente, collegando l'applicazione a un gestore code, nel qual caso il gestore code deve essere sulla stessa macchina dell'applicazione.
2. In un file della libreria client, che fornisce l'accesso ai gestori code sulla stessa macchina o su una macchina diversa.

IBM MQ fornisce un file di libreria client per ciascun ambiente:

AIX **AIX**
La libreria libmqic.a per le applicazioni non con thread o la libreria libmqic_r.a per le applicazioni con thread.

Linux **Linux**
La libreria libmqic.so per le applicazioni senza thread o la libreria libmqic_r.so per le applicazioni con thread.

IBM i **IBM i**
Eseguire il bind dell'applicazione client con il programma di servizio client LIBMQIC per le applicazioni senza thread o il programma di servizio LIBMQIC_R per le applicazioni con thread.

Windows **Windows**
MQIC32.LIB.

ALW *Collegamento di applicazioni C+ + con il codice IBM MQ MQI client*

È possibile scrivere le applicazioni da eseguire sul client in C + +. I metodi di creazione variano a seconda dell'ambiente.

Per informazioni su come collegare le applicazioni C + +, vedere Creazione di programmi C + + IBM MQ.

Per i dettagli completi di tutti gli aspetti relativi all'utilizzo di C + +, consultare Utilizzo di C + +

Multi *Collegamento delle applicazioni COBOL con il codice IBM MQ MQI client*

Dopo aver scritto un'applicazione COBOL che si desidera eseguire su IBM MQ MQI client, è necessario collegarla a una libreria appropriata.

IBM MQ fornisce un file di libreria client per ciascun ambiente:

AIX **AIX**
: Collegare l'applicazione COBOL non sottoposta a thread con la libreria libmqicb.a o l'applicazione COBOL sottoposta a thread con libmqicb_r.a.

IBM i **IBM i**
: Eseguire il bind dell'applicazione client COBOL con il programma di servizio AMQCSTUB per le applicazioni senza thread o con il programma di servizio AMQCSTUB_R per le applicazioni con thread.

Windows **Windows**
: Collegare il codice dell'applicazione alla libreria MQICCBB per COBOL a 32 bit. IBM MQ MQI client for Windows non supporta COBOL a 16 bit.

Windows *Collegamento delle applicazioni Visual Basic con il codice IBM MQ MQI client*

È possibile collegare le applicazioni Microsoft Visual Basic con il codice di IBM MQ MQI client su Windows.

Deprecated

Da IBM MQ 9.0, il supporto per Microsoft Visual Basic 6.0 è obsoleto. Le classi IBM MQ per .NET sono la tecnologia sostitutiva consigliata. Per ulteriori informazioni, consultare Sviluppo di applicazioni .NET.

Collegare l'applicazione di Visual Basic con i seguenti file di inclusione:

CMQB.bas
: MQI

CMQBB.bas
: MQAI

CMQCFB.bas
: Comandi PCF

CMQXB.bas
: Canali

Impostare `mqtype=2` per il client nel compilatore Visual Basic , per garantire la selezione automatica corretta della dll client:

MQIC32.dll
: Windows 7, Windows 8, Windows 2008 e Windows 2012

Concetti correlati

“Codifica in Visual Basic” a pagina 1055
Informazioni da considerare quando si codificano i programmi IBM MQ in Microsoft Visual Basic. Visual Basic è supportata solo su Windows.

“Preparazione dei programmi Visual Basic in Windows” a pagina 1022
Informazioni da considerare quando si utilizzano programmi Microsoft Visual Basic su Windows.

Esecuzione di applicazioni nell'ambiente di IBM MQ MQI client

Puoi eseguire un'applicazione IBM MQ sia in un ambiente IBM MQ completo che in un ambiente IBM MQ MQI client senza modificare il codice, purché vengano soddisfatte determinate condizioni.

Queste condizioni sono:

- Non è necessario che l'applicazione si connetta a più di un gestore code contemporaneamente.
- Il nome del gestore code non è preceduto da un asterisco (*) su una chiamata `MQCONN` o `MQCONNX` .
- L'applicazione non deve utilizzare alcuna delle eccezioni elencate in Quali applicazioni vengono eseguite su un IBM MQ MQI client?

Nota: Le librerie utilizzate in fase di modifica del collegamento determinano l'ambiente in cui deve essere eseguita l'applicazione.

Quando si utilizza l'ambiente IBM MQ MQI client , tenere presente che:

- Ogni applicazione in esecuzione nell'ambiente IBM MQ MQI client ha le proprie connessioni ai server. Un'applicazione stabilisce una connessione a un server ogni volta che emette una chiamata `MQCONN` o `MQCONNX` .
- Un'applicazione invia e riceve i messaggi in modo sincrono. Ciò implica un'attesa tra il momento in cui la chiamata viene emessa sul client e il ritorno di un codice di completamento e di un codice motivo sulla rete.
- Tutta la conversione dei dati viene effettuata dal server, ma consultare anche MQCCSID per informazioni sulla sostituzione del CCSID configurato della macchina.

Connessione delle applicazioni IBM MQ MQI client ai gestori code

Un'applicazione in esecuzione in un ambiente IBM MQ MQI client può connettersi a un gestore code in vari modi. È possibile utilizzare le variabili di ambiente, la struttura MQCNO o una tabella di definizione client.

Quando un'applicazione in esecuzione in un ambiente client IBM MQ emette una chiamata MQCONN o MQCONNX, il client identifica come effettuare la connessione. Quando viene emessa una chiamata MQCONNX da un'applicazione su un client IBM MQ , la libreria del client MQI ricerca le informazioni del canale client nel seguente ordine:

1. Utilizzo del contenuto dei campi `ClientConnOffset` o `ClientConnPtr` della struttura MQCNO (se fornita). Questi campi identificano la struttura di definizioni del canale (MQCD) da utilizzare come definizione del canale di connessione client. I dettagli di connessione possono essere sovrascritti utilizzando un'uscita di preconnessione. Per ulteriori informazioni, consultare “Riferimento alle definizioni di connessione mediante un'uscita di pre - connessione da un repository” a pagina 995.
2. Se la variabile di ambiente **MQSERVER** è impostata, viene utilizzato il canale che definisce.
3. Se è definito un file `mqclient.ini` e la stanza Channels contiene un attributo **ServerConnectionParms** , viene utilizzato il canale che definisce. Per ulteriori informazioni, consultare IBM MQ MQI client configuration file, `mqclient.ini` e Channels stanza of the client configuration file.
4. Se sono impostate le variabili di ambiente **MQCHLLIB** e **MQCHLTAB** , viene utilizzata la tabella di definizione del canale client a cui fanno riferimento. In alternativa, da IBM MQ 9.0, la variabile di ambiente **MQCCDTURL** fornisce la capacità equivalente di impostare una combinazione di variabili di ambiente **MQCHLLIB** e **MQCHLTAB** . Se **MQCCDTURL** è impostato, viene utilizzata la tabella di definizione del canale client a cui punta. Per ulteriori informazioni, consultare AccessoURL a CCDT.
5. Se un file `mqclient.ini` è definito e la stanza Channels contiene **ChannelDefinitionDirectory** e **ChannelDefinitionFile** attributi, questi attributi vengono utilizzati per individuare la tabella di definizione del canale client. Per ulteriori informazioni, consultare IBM MQ MQI client configuration file, `mqclient.ini` e Channels stanza of the client configuration file.
6. Infine, se le variabili di ambiente non sono impostate, il client ricerca una tabella di definizione del canale client con un percorso e un nome stabiliti dall'attributo **DefaultPrefix** della

stanza AllQueueManagers nel file `mqs.ini` . Per ulteriori informazioni, consultare la sezione AllQueueManagers stanza of the mqs.ini file.

Se la ricerca di una tabella di definizione del canale client ha esito negativo, il client utilizza i seguenti percorsi:

- Linux AIX Su AIX and Linux: `/var/mqm/AMQCLCHL.TAB`
- Windows Su Windows: `C:\Program Files\IBM\MQ\amqclchl.tab`
- IBM i Su IBM i: `/QIBM/UserData/mqm/@ipcc`
- MQ Appliance Su IBM MQ Appliance: *QMname*`_AMQCLCHL.TAB`. Vengono visualizzati in `mqbackup:// URI`.

La prima delle opzioni descritte nell'elenco precedente (utilizzando i campi `ClientConnOffset` o `ClientConnPtr` di MQCNO) è supportata solo dalla chiamata MQCONNX. Se l'applicazione utilizza MQCONN anziché MQCONNX, le informazioni sul canale vengono ricercate nei restanti cinque modi nell'ordine mostrato nell'elenco. Se il client non riesce a trovare le informazioni sul canale, la chiamata MQCONN o MQCONNX non riesce.

Il nome del canale (per la connessione client) deve corrispondere al nome del canale di connessione server definito sul server perché la chiamata MQCONN o MQCONNX abbia esito positivo.

Concetti correlati

Accesso indirizzabile Web alla tabella di definizione del canale client

Attività correlate

Configurazione delle connessioni tra il server e il client

Riferimenti correlati

Tabella definizione canale client

MQCNO - Opzioni di connessione

Connessione delle applicazioni client ai gestori code utilizzando le variabili di ambiente

Le informazioni sul canale client possono essere fornite a una applicazione in esecuzione in un ambiente client dalle variabili di ambiente.

Un'applicazione in esecuzione in un ambiente IBM MQ MQI client può connettersi a un gestore code utilizzando le seguenti variabili di ambiente:

SERVER MQT

La variabile di ambiente **MQSERVER** viene utilizzata per definire un canale minimo. **MQSERVER** specifica l'ubicazione del server IBM MQ e il metodo di comunicazione da utilizzare.

MQCHLLIB

La variabile di ambiente **MQCHLLIB** specifica il percorso della directory del file contenente la CCDT (client channel definition table). Il file viene creato sul server, ma può essere copiato sulla workstation IBM MQ MQI client .

MQCHLTAB

La variabile di ambiente **MQCHLTAB** specifica il nome del file contenente la CCDT (client channel definition table).

Da IBM MQ 9.0, la variabile di ambiente **MQCCDTURL** fornisce la capacità equivalente di impostare una combinazione delle variabili di ambiente **MQCHLLIB** e **MQCHLTAB** . **MQCCDTURL** consente di fornire un URL di file, ftp o http come un singolo valore da cui è possibile ottenere una tabella di definizione di canale client. Per ulteriori informazioni, vedere Accesso indirizzabile Web alla tabella di definizione del canale client.

Connessione delle applicazioni client ai gestori code utilizzando la struttura MQCNO

È possibile specificare la definizione del canale in una struttura di definizione del canale (MQCD), che viene fornito utilizzando la struttura MQCNO della chiamata MQCONNX.

Per ulteriori informazioni, consultare Creazione di un canale di connessione client su IBM MQ MQI client utilizzando MQCNO.

Connessione di applicazioni client ai gestori code utilizzando una tabella di definizione del canale client
Se si utilizza il comando MQSC DEFINE CHANNEL, i dettagli forniti vengono inseriti nella tabella di definizione del canale client (ccdt). Il contenuto del parametro **QMgrName** della chiamata MQCONN o MQCONNX determina il gestore code a cui si connette il client.

Il client accede a questo file per determinare il canale che verrà utilizzato da un'applicazione. Se è presente più di una definizione di canale adatta, la scelta del canale è influenzata dagli attributi del canale client (CLNTWGHT) e di affinità di connessione (AFFINITY).

Utilizzo della riconnessione client automatica
È possibile riconnettere automaticamente le proprie applicazioni client, senza scrivere alcun codice aggiuntivo, configurando un certo numero di componenti.

La riconnessione automatica del client è *in linea*. La connessione viene ripristinata automaticamente in qualsiasi punto del programma applicativo client e vengono ripristinati anche tutti gli handle per aprire gli oggetti.

Al contrario, la riconnessione manuale richiede l'applicazione client per ricreare una connessione utilizzando MQCONN o MQCONNX e riaprire gli oggetti. La riconnessione automatica del client è adatta a molte ma non a tutte le applicazioni client.

Per ulteriori informazioni, consultare Ricarica client automatica.

Ruolo della tabella di definizione del canale client
La tabella di definizione del canale client (CCDT) contiene le definizioni dei canali di connessione client. È particolarmente utile se le applicazioni client potrebbero dover connettersi a diversi gestori code alternativi.

La tabella di definizione del canale client viene creata quando si definisce un gestore code. Lo stesso file può essere utilizzato da più di un client IBM MQ .

Esistono diversi modi per un'applicazione client per utilizzare una CCDT. La CCDT può essere copiata sul computer client. È possibile copiare la CCDT in un'ubicazione condivisa da più di un client. È possibile rendere il CCDT accessibile per il client come un file condiviso, mentre rimane sul server.

Da IBM MQ 9.0, la CCDT può essere ospitata in un'ubicazione centralizzata accessibile tramite un URI, eliminando la necessità di aggiornare singolarmente la CCDT per ogni client distribuito.

Concetti correlati
Accesso indirizzabile Web alla tabella di definizione del canale client
Attività correlate
Accesso alle definizioni del canale di connessione client
Riferimenti correlati
Tabella definizione canale client

Gruppi di gestori code in CCDT
È possibile definire una serie di connessioni nella tabella di definizione del canale client (CCDT) come un gruppo di gestori code . È possibile connettere un'applicazione a un gestore code che fa parte di un gruppo di gestori code. Questa operazione può essere eseguita prefissando il nome del gestore code su una chiamata MQCONN o MQCONNX con un asterisco.

È possibile scegliere di definire le connessioni a più di una macchina server perché:

- Si desidera connettere un client a uno qualsiasi di una serie di gestori code in esecuzione, per migliorare la disponibilità.
- Si desidera riconnettere un client allo stesso gestore code a cui si è connesso correttamente l'ultima volta, ma connettersi a un altro gestore code se la connessione non riesce.
- Si desidera essere in grado di ritentare una connessione client a un gestore code differente se la connessione non riesce, immettendo nuovamente MQCONN nel programma client.

- Se la connessione non riesce, si desidera riconnettere automaticamente una connessione client a un altro gestore code, senza scrivere alcun codice client.
- Si desidera riconnettere automaticamente una connessione client a un'altra istanza di un gestore code a più istanze se un'istanza in standby assume il controllo, senza scrivere alcun codice client.
- Si desidera bilanciare le connessioni client tra diversi gestori code, con più client che si collegano ad alcuni gestori code rispetto ad altri.
- Si desidera estendere la riconnessione di molte connessioni client su più gestori code e nel corso del tempo, nel caso in cui l'elevato volume di connessioni causi un malfunzionamento.
- Si desidera spostare i gestori code senza modificare il codice dell'applicazione client.
- Si desidera scrivere programmi di applicazione client che non devono conoscere i nomi dei gestori code.

Non è sempre appropriato connettersi a gestori code differenti. Un client transazionale esteso o un client Java in WebSphere Application Server, ad esempio, potrebbe dover connettersi a un'istanza del gestore code prevedibile. La riconnessione automatica del client non è supportata da IBM MQ classes for Java.

Un gruppo di gestori code è una serie di connessioni definite nella tabella di definizione del canale client (CCDT). La serie è definita dai membri che hanno lo stesso valore dell'attributo **QMNAME** nelle loro definizioni di canale.

Figura 97 a pagina 927 è una rappresentazione grafica di una tabella di connessione client, che mostra tre gruppi di gestori code, due gruppi di gestori code denominati scritti in CCDT come **QMNAME** (QM1) e **QMNAME** (QMGrp1)e un gruppo vuoto o predefinito scritto come **QMNAME** ('').

1. Il gruppo di gestori code QM1 dispone di tre canali di connessione client, che lo collegano ai gestori code QM1 e QM2. QM1 potrebbe essere un gestore code a più istanze ubicato su due server differenti.
2. Il gruppo di gestori code predefinito ha sei canali di connessione client che lo collegano a tutti i gestori code.
3. QMGrp1 dispone di canali di connessione client a due gestori code, QM4 e QM5.

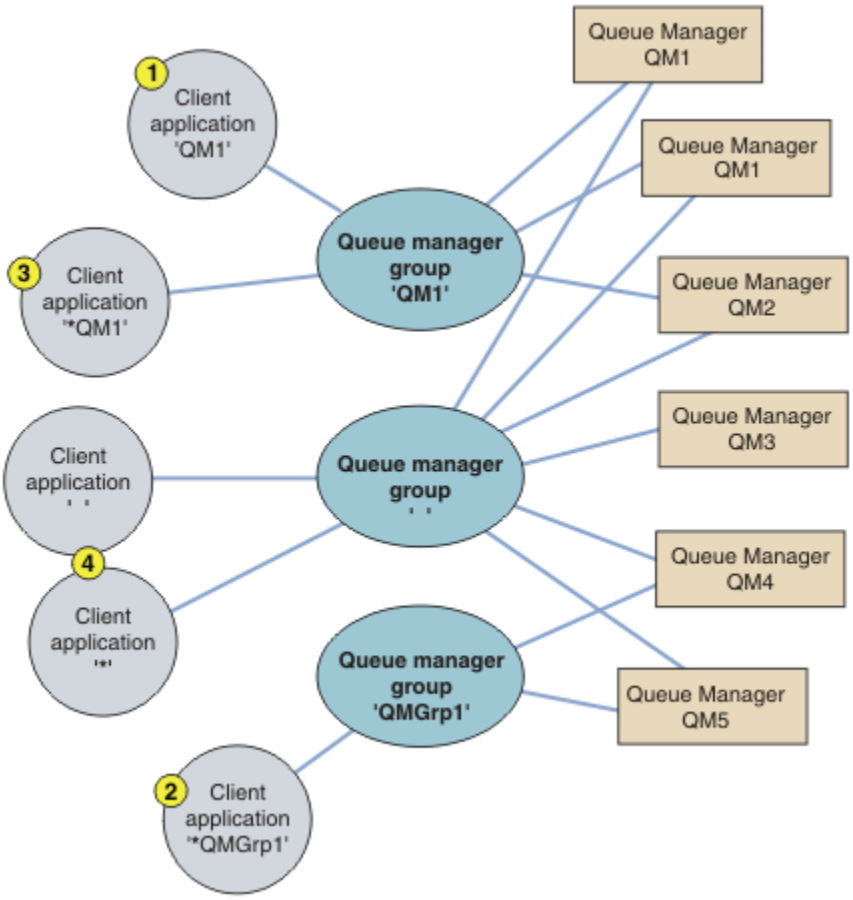


Figura 97. Gruppi gestore code

Quattro esempi di utilizzo di questa tabella di connessioni client sono descritti con l'aiuto delle applicazioni client numerate in Figura 97 a pagina 927.

1. Nel primo esempio, l'applicazione client passa un nome gestore code, `QM1`, come parametro **`QmgrName`** alla sua chiamata `MQCONN` o `MQCONNX` MQI. Il codice client IBM MQ seleziona il gruppo di gestori code corrispondente, QM1. Il gruppo contiene tre canali di connessione e IBM MQ MQI client prova a connettersi a `QM1` utilizzando ciascuno di questi canali a turno finché non trova un listener IBM MQ per la connessione collegata a un gestore code in esecuzione denominato `QM1`.

   L'ordine dei tentativi di connessione dipende dal valore dell'attributo AFFINITY della connessione client e dalle ponderazioni del canale client. All'interno di questi vincoli, l'ordine dei tentativi di connessione viene randomizzato, sia sulle tre connessioni possibili, sia nel tempo, al fine di distribuire il carico di effettuare connessioni.

   La chiamata `MQCONN` o `MQCONNX` emessa dall'applicazione client ha esito positivo quando viene stabilita una connessione a un'istanza in esecuzione di `QM1`.
2. Nel secondo esempio, l'applicazione client passa un nome gestore code con un prefisso con un asterisco, `*QMGrp1` come parametro **`QmgrName`** alla sua chiamata MQI `MQCONN` o `MQCONNX` . Il client IBM MQ seleziona il gruppo di gestori code corrispondente, QMGrp1. Questo gruppo contiene due canali di connessione client e IBM MQ MQI client tenta di connettersi a *qualsiasi* gestore code utilizzando ciascun canale a turno. In questo esempio, IBM MQ MQI client deve stabilire correttamente una connessione; il nome del gestore code a cui si connette non è importante.

   La regola per l'ordine dei tentativi di connessione è la stessa di prima. L'unica differenza è che prefissando il nome del gestore code con un asterisco, il client indica che il nome del gestore code non è rilevante.

La chiamata MQCONN o MQCONNX emessa dall'applicazione client ha esito positivo quando viene stabilita una connessione a un'istanza in esecuzione di qualsiasi gestore code connesso dai canali nel gruppo di gestori code QMGrp1 .

3. Il terzo esempio è essenzialmente lo stesso del secondo perché il parametro **QmgrName** è preceduto da un asterisco, *QM1. L'esempio illustra che non è possibile determinare a quale gestore code si connetterà una connessione del canale client ispezionando l'attributo QMNAME in un'unica definizione di canale. Il fatto che l'attributo **QMNAME** della definizione di canale sia QM1, non è sufficiente per richiedere una connessione a un gestore code denominato QM1. Se l'applicazione client antepone il parametro **QmgrName** con un asterisco, qualsiasi gestore code è una destinazione di connessione possibile.

   In questo caso, le chiamate MQCONN o MQCONNX emesse dall'applicazione client hanno esito positivo quando viene stabilita una connessione a un'istanza in esecuzione di QM1 o QM2.

4. Il quarto esempio illustra l'utilizzo del gruppo predefinito. In questo caso, l'applicazione client passa un asterisco, '*'o uno spazio vuoto ' ', come parametro **QmgrName** alla sua chiamata MQI MQCONN o MQCONNX . Per convenzione nella definizione di canale client, un attributo **QMNAME** vuoto indica il gruppo di gestori code predefinito e un parametro **QmgrName** vuoto o asterisco corrisponde a un attributo **QMNAME** vuoto.

   In questo esempio, il gruppo di gestori code predefinito ha connessioni del canale client a tutti i gestori code. Selezionando il gruppo di gestori code predefinito, l'applicazione potrebbe essere connessa a qualsiasi gestore code del gruppo.

   La chiamata MQCONN o MQCONNX emessa dall'applicazione client ha esito positivo quando viene stabilita una connessione a un'istanza in esecuzione di qualsiasi gestore code.

   Nota: Il gruppo predefinito è diverso da un gestore code predefinito, anche se un'applicazione utilizza un parametro **QmgrName** vuoto per connettersi al gruppo di gestori code predefinito o al gestore code predefinito. Il concetto di gruppo di gestori code predefiniti è rilevante solo per un'applicazione client e per un gestore code predefinito per un'applicazione server.

Definire i canali di connessione client solo su un gestore code, inclusi i canali che si connettono a un secondo o terzo gestore code. Non definirli su due gestori code, quindi provare ad unire le due tabelle di definizione del canale client. Il client può accedere a una sola tabella di definizione del canale client.

Esempi

Esaminare nuovamente l' elenco dei motivi per cui si utilizzano i gruppi di gestori code all'inizio dell'argomento. In che modo l'uso di un gruppo di gestori code fornisce tali funzioni?

Connettersi a uno qualsiasi di una serie di gestori code.
Definire un gruppo di gestori code con le connessioni a tutti i gestori code nel set e connettersi al gruppo utilizzando il parametro **QmgrName** preceduto da un asterisco.

Riconnettersi allo stesso gestore code, ma connettersi a uno differente, se il gestore code connesso all'ultima volta non è disponibile.
Definire un gruppo di gestori code come prima, ma impostare l'attributo **AFFINITY** (PREFERRED) su ciascuna definizione di canale client.

Riprovare una connessione a un altro gestore code se una connessione non riesce.
Connettersi a un gruppo di gestori code ed emettere nuovamente la chiamata MQCONN o MQCONNX MQI se la connessione è interrotta o se il gestore code ha esito negativo.

Riconnettersi automaticamente ad un altro gestore code se una connessione non riesce.
Connettersi a un gruppo di gestori code utilizzando l'opzione MQCONNX **MQCNO** MQCNO_RECONNECT.

Riconnettersi automaticamente a un'altra istanza di un gestore code a più istanze.
Eseguire la stessa operazione dell'esempio precedente. In questo caso, se si desidera limitare il gruppo di gestori code per connettersi alle istanze di un particolare gestore code a più istanze, definire il gruppo con le connessioni solo alle istanze del gestore code a più istanze.

È anche possibile richiedere all'applicazione client di emettere la relativa chiamata MQCONN o MQCONNX MQI senza un asterisco come prefisso al parametro **QmgrName** . In questo modo l'applicazione client può connettersi solo a un gestore code denominato. Infine, è possibile impostare

l'opzione **MQCNO** su MQCNO_RECONNECT_Q_MGR. Questa opzione accetta le riconnessioni allo stesso gestore code precedentemente connesso. È anche possibile utilizzare questo valore per limitare le riconnessioni alla stessa istanza di un gestore code normale.

Bilanciare le connessioni client tra i gestori code, con più client connessi ad alcuni gestori code rispetto ad altri.

Definire un gruppo di gestori code e impostare l'attributo **CLNTWGHT** su ciascuna definizione di canale client per distribuire le connessioni in modo non uniforme.

Distribuire il carico di riconnessione del client in modo non uniforme, e diffonderlo nel tempo, dopo un errore di connessione o di gestore code.

Eseguire la stessa operazione dell'esempio precedente. IBM MQ MQI client rende casuali le riconnessioni tra i gestori code e le ridistribuisce nel tempo.

Spostare i gestori code senza modificare il codice client.

CDT isola l'applicazione client dall'ubicazione del gestore code. CCDT è un file di dati che può essere definito sul client, letto da una posizione condivisa o recuperato da un server Web. Per ulteriori informazioni, consultare Tabella di definizione di canale client.

Scrivere un'applicazione client che non conosca i nomi dei gestori code.

Utilizzare i nomi dei gruppi di gestori code e definire una convenzione di denominazione per i nomi dei gruppi di gestori code che sia rilevante per le applicazioni client nella propria organizzazione e che rifletta l'architettura delle soluzioni piuttosto che la denominazione dei gestori code.

z/OS *Connessione ai gruppi di condivisione code*

È possibile connettere l'applicazione a un gestore code che fa parte di un gruppo di condivisione code. Questa operazione può essere eseguita utilizzando il nome del gruppo di condivisione code invece del nome del gestore code nella chiamata MQCONN o MQCONNX.

I gruppi di condivisione code hanno un nome composto da un massimo di quattro caratteri. Il nome deve essere univoco nella rete e diverso da qualsiasi altro nome di gestore code.

La definizione di canale client deve utilizzare l'interfaccia generica del gruppo di condivisione code per connettersi a un gestore code disponibile nel gruppo. Per ulteriori informazioni, consultare Connessione di un client a un gruppo di condivisione code. Viene eseguito un controllo per assicurarsi che il gestore code a cui si connette il listener sia un membro del gruppo di condivisione code.

Per ulteriori informazioni sulle code condivise, consultare Code condivise e gruppi di condivisione code.

Esempi di importanza e affinità del canale

Questi esempi illustrano come vengono selezionati i canali di connessione client quando vengono utilizzati `ClientChannelWeights` diversi da zero.

Gli attributi `ClientChannelPeso` e `ConnectionAffinity` controllano il modo in cui vengono selezionati i canali di connessione client quando è disponibile più di un canale adatto per una connessione. Questi canali sono configurati per connettersi a diversi gestori code in modo da fornire maggiore disponibilità, bilanciamento del carico di lavoro o entrambi. Le chiamate MQCONN che potrebbero risultare in una connessione a uno dei diversi gestori code devono anteporre un asterisco al nome del gestore code, come descritto in: Esempi di chiamate MQCONN: Esempio 1. Il nome del gestore code include un asterisco (*).

I canali candidati applicabili per una connessione sono quelli in cui l'attributo QMNAME corrisponde al nome gestore code specificato nella chiamata MQCONN. Se tutti i canali applicabili per una connessione hanno un `ClientChannelWeight` uguale a zero (impostazione predefinita), vengono selezionati in ordine alfabetico come nell'esempio: Esempi di chiamate MQCONN: Esempio 1. Il nome del gestore code include un asterisco (*).

I seguenti esempi illustrano cosa accade quando vengono utilizzati `ClientChannelWeights` diversi da zero. Si noti che, poiché questa funzione implica la selezione di canali pseudo - casuali, gli esempi mostrano una sequenza di azioni che potrebbero verificarsi piuttosto che ciò che sicuramente accadrà.

Esempio 1. Selezione dei canali quando ConnectionAffinity è impostata su PREFERRED
Questo esempio illustra come IBM MQ MQI client seleziona un canale da una CCDT, dove ConnectionAffinity è impostato su PREFERRED.

In questo esempio, diverse macchine client utilizzano una CCDT (Client Channel Definition Table) fornita da un gestore code. CDT include canali di connessione client con i seguenti attributi (visualizzati utilizzando la sintassi del comando DEFINE CHANNEL):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

L'applicazione emette MQCONN (*CORE)

Il canale A non è un candidato per questa connessione, perché l'attributo QMNAME non corrisponde. I canali B, C e D sono identificati come candidati e sono posti in ordine di preferenza in base alla loro ponderazione. In questo esempio l'ordine potrebbe essere C, B, D. Il client tenta di collegarsi al gestore code in core2.ops.company.example. Il nome del gestore code a tale indirizzo non viene controllato perché la chiamata MQCONN includeva un asterisco nel nome gestore code.

È importante notare che, con `AFFINITY(PREFERRED)`, ogni volta che questa particolare macchina client si collega, posizionerà i canali nello stesso ordine iniziale di preferenza. Ciò si applica anche quando le connessioni provengono da processi diversi o in momenti diversi.

In questo esempio, non è possibile raggiungere il gestore code in core.2.ops.company.example . Il client tenta di connettersi a core1.ops.company.example perché il canale B è il successivo nell'ordine di preferenza. Inoltre, il canale C viene degradato per diventare il meno preferito.

Una seconda chiamata MQCONN (*CORE) viene emessa dalla stessa applicazione. Il canale C è stato retrocesso dalla connessione precedente, quindi il canale più preferito è ora B. Questa connessione viene effettuata a core1.ops.company.example.

Una seconda macchina che condivide la stessa tabella di definizione canale client posiziona i canali in un ordine di preferenza iniziale diverso. Ad esempio, D, B, C. In circostanze normali, con tutti i canali che funzionano, le applicazioni su questa macchina sono connesse a core3.ops.company.example mentre quelle sulla prima macchina sono connesse a core2.ops.company.example. Ciò consente il bilanciamento del carico di lavoro di un numero elevato di client su più gestori code, consentendo a ciascun client di connettersi allo stesso gestore code, se disponibile.

Esempio 2. Selezione dei canali quando ConnectionAffinity è impostata su NONE
Questo esempio illustra come IBM MQ MQI client seleziona un canale da una CCDT, dove ConnectionAffinity è impostato su NONE.

In questo esempio, alcuni client utilizzano una CCDT (Client Channel Definition Table) fornita da un gestore code. CDT include canali di connessione client con i seguenti attributi (visualizzati utilizzando la sintassi del comando DEFINE CHANNEL):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

L'applicazione emette MQCONN (*CORE). Come nell'esempio precedente, il canale A non viene considerato perché QMNAME non corrisponde. I canali B, C o D sono selezionati in base al loro peso, con probabilità del 50%, 30% o 20%. In questo esempio, il canale B potrebbe essere selezionato. Non è stato creato alcun ordine di preferenza persistente.

Viene effettuata una seconda chiamata MQCONN (*CORE). Ancora una volta, viene selezionato uno dei tre canali applicabili, con le stesse probabilità. In questo esempio, viene scelto il canale C. Tuttavia, core2.ops.company.example non risponde, quindi viene effettuata un'altra scelta tra i rimanenti canali candidati. Il canale B è selezionato e l'applicazione è connessa a core1.ops.company.example.

Con AFFINITY (NONE), ogni chiamata MQCONN è indipendente da qualsiasi altra. Pertanto, quando questa applicazione di esempio effettua un terzo MQCONN (*CORE), potrebbe ancora una volta tentare la connessione tramite il canale interrotto C, prima di scegliere uno tra B o D.

Esempi di chiamate MQCONN
Esempi di utilizzo di MQCONN per la connessione a un gestore code specifico o a uno di un gruppo di gestori code.

In ognuno degli esempi riportati di seguito, la rete è la stessa; esiste una connessione definita a due server dallo stesso IBM MQ MQI client. In questi esempi, è possibile utilizzare la chiamata MQCONNX al posto della chiamata MQCONN.

Esistono due gestori code in esecuzione sulle macchine server, uno denominato SALE e l'altro denominato SALE_BACKUP.

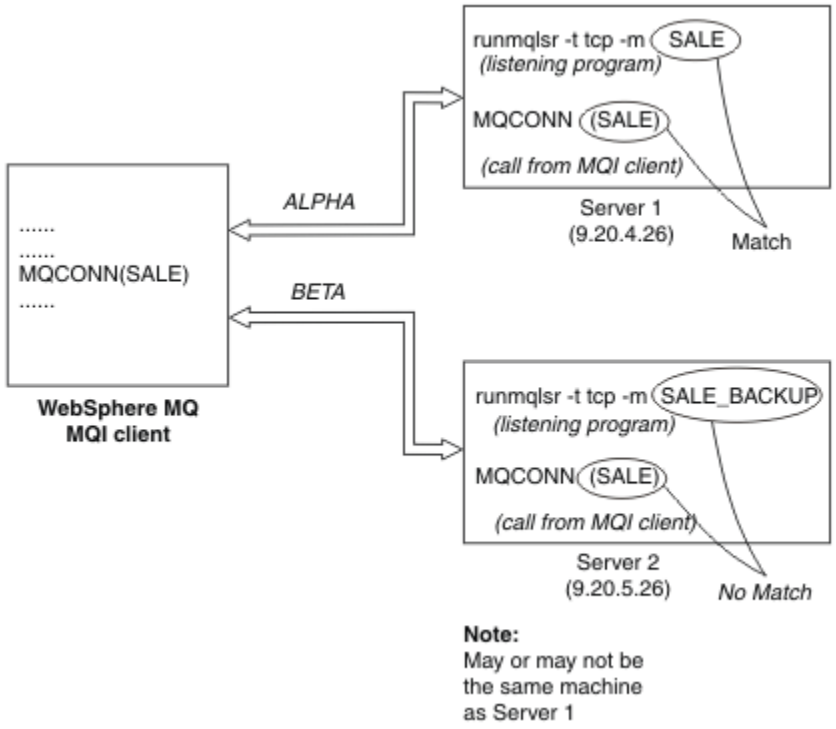


Figura 98. Esempio MQCONN

Le definizioni per i canali in questi esempi sono:

Definizioni SALDI:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Definizione SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

Le definizioni del canale client possono essere riepilogate come segue:

Nome	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	VENDITA
BETA	CLNTCONN	TCP	9.20.5.26	VENDITA

Cosa dimostrano gli esempi MQCONN
Gli esempi mostrano l'utilizzo di più gestori code come sistema di backup.

Si supponga che il collegamento di comunicazione al server 1 sia temporaneamente interrotto. Viene dimostrato l'utilizzo di più gestori code come sistema di backup.

Ogni esempio copre una chiamata MQCONN differente e fornisce una spiegazione di ciò che accade nell'esempio specifico presentato, applicando le seguenti regole:

1. La CCDT (client channel definition table) viene scansionata in ordine alfabetico per un nome gestore code (campo QMNAME) corrispondente a quello fornito nella chiamata MQCONN.
2. Se viene trovata una corrispondenza, viene utilizzata la definizione di canale.
3. Si è tentato di avviare il canale sulla macchina identificata dal nome connessione (CONNAME). Se l'operazione ha esito positivo, l'applicazione continua. Richiede:
   - Un listener in esecuzione sul server.
   - Il listener da connettere allo stesso gestore code di quello a cui il client desidera connettersi (se specificato).
4. Se il tentativo di avviare il canale non riesce e c'è più di una voce nella tabella di definizione del canale client (in questo esempio ci sono due voci), viene ricercata un'ulteriore corrispondenza nel file. Se viene trovata una corrispondenza, l'elaborazione continua al passo 1.
5. Se non viene trovata alcuna corrispondenza o non ci sono più voci nella tabella di definizione del canale client e il canale non è stato avviato, l'applicazione non è in grado di connettersi. Nella chiamata MQCONN vengono restituiti un codice motivo e un codice di completamento appropriati. L'applicazione può intraprendere un'azione in base ai codici motivo e di completamento restituiti.

Esempio 1. Il nome del gestore code include un asterisco (*)
In questo esempio, l'applicazione non è interessata a quale gestore code si connette. L'applicazione emette una chiamata MQCONN per un nome gestore code che include un asterisco. Viene scelto un canale adatto.

L'applicazione ha i seguenti problemi:

```
MQCONN (*SALE)
```

Seguendo le regole, questo è ciò che accade in questa istanza:

1. La CCDT (client channel definition table) viene sottoposta a scansione per il nome del gestore code SALE, corrispondente alla chiamata MQCONN dell'applicazione.
2. Sono state trovate definizioni di canale per ALPHA e BETA .
3. Se un canale ha un valore CLNTWGHT pari a 0, questo canale viene selezionato. Se entrambi hanno un valore CLNTWGHT pari a 0, il canale ALPHA viene selezionato perché è il primo in sequenza alfabetica. Se entrambi i canali hanno un valore CLNTWGHT diverso da zero, un canale viene selezionato in modo casuale, in base al suo peso.
4. Viene effettuato un tentativo di avviare il canale.
5. Se è stato selezionato il canale BETA , il tentativo di avviarlo ha esito positivo.

6. Se è stato selezionato il canale ALPHA , il tentativo di avviarlo NON ha esito positivo perché il collegamento di comunicazione è interrotto. Si applicano quindi le seguenti operazioni:
   a. L'unico altro canale per il nome gestore code SALE è BETA.
   b. È stato effettuato un tentativo di avviare questo canale - l'operazione ha esito positivo.
7. Un controllo per verificare che un listener sia in esecuzione indica che ne esiste uno in esecuzione. Non è connesso al gestore code SALE , ma poiché il parametro della chiamata MQI contiene un asterisco (*), non viene eseguito alcun controllo. L'applicazione è connessa al gestore code SALE_BACKUP e continua l'elaborazione.

Esempio 2. Nome gestore code specificato
In questo esempio l'applicazione deve connettersi a uno specifico gestore code. L'applicazione emette una chiamata MQCONN per tale nome gestore code. Viene scelto un canale adatto.

L'applicazione richiede una connessione a un gestore code specifico, denominato SALE, come visualizzato nella chiamata MQI:

```
MQCONN (SALE)
```

Seguendo le regole, questo è ciò che accade in questa istanza:

1. La tabella di definizione del canale client (CCDT) viene scansionata in ordine alfabetico per il nome del gestore code SALE, corrispondente alla chiamata MQCONN dell'applicazione.
2. La prima definizione di canale trovata per la corrispondenza è ALPHA.
3. È stato effettuato un tentativo di avvio del canale - l'operazione non ha avuto esito positivo perché il collegamento di comunicazione è interrotto.
4. La tabella di definizione del canale client viene nuovamente sottoposta a scansione per il nome gestore code SALE e viene trovato il nome canale BETA .
5. È stato effettuato un tentativo di avviare il canale - l'operazione ha esito positivo.
6. Un controllo per verificare che un listener sia in esecuzione mostra che ne esiste uno in esecuzione, ma non è connesso al gestore code SALE .
7. Non ci sono ulteriori voci nella tabella di definizione del canale client. L'applicazione non può continuare e riceve il codice di ritorno MQRC_Q_MGR_NOT_AVAILABLE.

Esempio 3. Il nome del gestore code è vuoto o un asterisco (*)
In questo esempio, l'applicazione non è interessata a quale gestore code si connette. L'applicazione emette un MQCONN specificando un nome gestore code vuoto o un asterisco. Viene scelto un canale adatto.

Questo viene trattato come “Esempio 1. Il nome del gestore code include un asterisco (*)” a pagina 932.

Nota: Se questa applicazione fosse in esecuzione in un ambiente diverso da IBM MQ MQI cliente il nome fosse vuoto, tenterebbe di connettersi al gestore code predefinito. Ciò non si verifica quando viene eseguito da un ambiente client; il gestore code a cui si accede è quello associato al listener a cui si connette il canale.

L'applicazione ha i seguenti problemi:

```
MQCONN ("")
```

o

```
MQCONN (*)
```

Seguendo le regole, questo è ciò che accade in questa istanza:

1. La tabella di definizione del canale client (CCDT) viene sottoposta a scansione in ordine alfabetico in base alla sequenza dei nomi dei canali, alla ricerca di un nome gestore code vuoto, corrispondente alla chiamata MQCONN dell'applicazione.
2. La voce per il nome del canale ALPHA ha un nome gestore code nella definizione di SALE. Non corrisponde al parametro della chiamata MQCONN, che richiede che il nome del gestore code sia vuoto.
3. La voce successiva è per il nome canale BETA.
4. Il `queue manager name` nella definizione è SALE. Ancora una volta, non corrisponde al parametro della chiamata MQCONN, che richiede che il nome del gestore code sia vuoto.
5. Non ci sono ulteriori voci nella tabella di definizione del canale client. L'applicazione non può continuare e riceve il codice di ritorno MQRC_Q_MGR_NOT_AVAILABLE.

Attivazione nell'ambiente client

I messaggi inviati dalle applicazioni IBM MQ in esecuzione su IBM MQ MQI clients contribuiscono all'attivazione nello stesso modo di qualsiasi altro messaggio e possono essere utilizzati per attivare i programmi sia sul server che sul client.

L'attivazione viene spiegata in dettaglio in Trigger canali.

Il controllo trigger e l'applicazione da avviare devono trovarsi sullo stesso sistema.

Le caratteristiche predefinite della coda attivata sono le stesse dell'ambiente del server. In particolare, se non vengono specificate opzioni di controllo del punto di sincronizzazione MQPMO in un'applicazione client che inserisce i messaggi in una coda attivata locale per un gestore code z/OS , i messaggi vengono inseriti all'interno di un'unità di lavoro. Se la condizione di attivazione viene quindi soddisfatta, il messaggio del trigger viene inserito nella coda di avvio all'interno della stessa unità di lavoro e non può essere richiamato dal controllo del trigger fino a quando l'unità di lavoro non termina. Il processo che deve essere attivato non viene avviato fino al termine dell'unità di lavoro.

Definizione di processo

È necessario definire la definizione di processo sul server, poiché è associata alla coda su cui è impostato il trigger.

L'oggetto processo definisce cosa deve essere attivato. Se il client e il server non sono in esecuzione sulla stessa piattaforma, qualsiasi processo avviato dal controllo trigger deve definire *ApplType*, altrimenti il server utilizza le definizioni predefinite (ovvero, il tipo di applicazione normalmente associato alla macchina server) e causa un errore.

Ad esempio, se il controllo trigger è in esecuzione su un IBM MQ MQI client e si desidera inviare una richiesta a un server su un altro sistema operativo, MQAT_WINDOWS_NT deve essere definito altrimenti l'altro sistema operativo utilizza le definizioni predefinite e il processo ha esito negativo.

Controllo trigger

Il controllo dei trigger fornito da prodotti nonz/OS IBM MQ viene eseguito negli ambienti client per i sistemi IBM i IBM i, AIX, Linux, and Windows .

Per eseguire il controllo trigger, immettere uno dei seguenti comandi:

- IBM i Su IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW Su piattaforme AIX, Linux, and Windows :

```
runmqtmc [-m QMgrName] [-q InitQ]
```

La coda di avvio predefinita è SYSTEM.DEFAULT.INITIATION.QUEUE sul gestore code predefinito. La coda di iniziazione è il punto in cui il controllo del trigger cerca i messaggi del trigger. Successivamente,

richiama i programmi per i messaggi trigger appropriati. Questo controllo trigger supporta il tipo di applicazione predefinito ed è uguale a `runmqtrm` ad eccezione del fatto che collega le librerie client.

La stringa di comando, creata dal controllo trigger, è la seguente:

1. Il *ApplicId* dalla definizione del processo pertinente. *ApplicId* è il nome del programma da eseguire, come dovrebbe essere immesso sulla riga comandi.
2. La struttura MQTMC2 , racchiusa tra virgolette, ottenuta dalla coda di avvio. Viene avviata una stringa di comando che ha questa stringa, esattamente come fornita, tra virgolette in modo che il comando di sistema la accetti come un parametro.
3. Il *EnvrData* dalla definizione del processo pertinente.

Il controllo dei trigger non controlla se è presente un altro messaggio sulla coda di iniziazione fino al completamento dell'applicazione che ha avviato. Se l'applicazione ha molta elaborazione da fare, il controllo trigger potrebbe non tenere il passo con il numero di messaggi trigger in arrivo. Ci sono due modi per affrontare questa situazione:

1. Avere più controlli trigger in esecuzione

   Se si sceglie di avere più controlli trigger in esecuzione, è possibile controllare il numero massimo di applicazioni che possono essere eseguite contemporaneamente.
2. Eseguire le applicazioni avviate in background

   Se si sceglie di eseguire le applicazioni in background, IBM MQ non impone alcuna limitazione sul numero di applicazioni che possono essere eseguite.

Per eseguire l'applicazione avviata in background su sistemi AIX and Linux , è necessario inserire una & (e commerciale) alla fine del *EnvrData* della definizione del processo.

Applicazioni CICS (nonz/OS)
Un programma applicativo nonz/OS CICS che emette una chiamata MQCONN o MQCONNX deve essere definito in CEDA come RESIDENT. Se si ricollglia un'applicazione server CICS come client, si rischia di perdere il supporto del punto di sincronizzazione.

Un programma applicativo nonz/OS CICS che emette una chiamata MQCONN o MQCONNX deve essere definito in CEDA come RESIDENT. Per rendere il codice residente il più piccolo possibile, è possibile collegarsi a un programma separato per emettere la chiamata MQCONN o MQCONNX .

Se la variabile di ambiente MQSERVER viene utilizzata per definire la connessione client, deve essere specificata in CICSENV.CMD .

Le applicazioni IBM MQ possono essere eseguite in un ambiente server IBM MQ o su un client IBM MQ senza modificare il codice. Tuttavia, in un ambiente server IBM MQ , CICS può agire come coordinatore del punto di sincronizzazione e si utilizzano EXEC CICS SYNCPOINT e EXEC CICS SYNCPOINT ROLLBACK anziché **MQCMIT** e **MQBACK**. Se un'applicazione CICS viene semplicemente ricollegata come client, il supporto del punto di sincronizzazione viene perso. **MQCMIT** e **MQBACK** devono essere utilizzati per l'applicazione in esecuzione su un IBM MQ MQI client.

ALW Preparazione ed esecuzione di applicazioni CICS e Tuxedo

Per eseguire le applicazioni CICS e Tuxedo come applicazioni client, si utilizzano librerie diverse da quelle utilizzate con le applicazioni server. Anche l'ID utente con cui viene eseguita l'applicazione è diverso.

Per preparare le applicazioni CICS e Tuxedo per l'esecuzione come applicazioni IBM MQ MQI client , seguire le istruzioni in Configurazione di un client transazionale esteso.

Tenere presente, tuttavia, che le informazioni relative specificamente alla preparazione delle applicazioni CICS e Tuxedo, inclusi i programmi di esempio forniti con IBM MQ, presuppongono che si stiano preparando le applicazioni da eseguire su un sistema server IBM MQ . Di conseguenza, le informazioni si riferiscono solo a librerie di IBM MQ che sono destinate all'utilizzo su un sistema server. Durante la preparazione delle applicazioni client, è necessario effettuare le seguenti operazioni:

- Utilizzare la libreria di sistema client appropriata per i bind di lingua utilizzati dall'applicazione. Ad esempio:
  - Linux AIX Per le applicazioni scritte in C su AIX and Linux, utilizzare la libreria libmqic invece di libmqm.
  - Windows Su sistemi Windows , utilizzare la libreria mqic.lib invece di mqm.lib.
- Invece delle librerie di sistema del server mostrate in Tabella 134 a pagina 936 e Tabella 135 a pagina 936, utilizzare le librerie di sistema client equivalenti. Se una libreria di sistema server non è elencata in queste tabelle, utilizzare la stessa libreria su un sistema client.

Tabella 134. Librerie di sistema client su AIX and Linux

Libreria per un sistema server di IBM MQ	**Libreria equivalente da utilizzare su un sistema client IBM MQ**
libmqmxa	libmqcxa

Tabella 135. Librerie di sistema client su sistemi Windows

Libreria per un sistema server di IBM MQ	**Libreria equivalente da utilizzare su un sistema client IBM MQ**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

L'ID utente utilizzato da un'applicazione client

Quando si esegue un'applicazione server IBM MQ in CICS, normalmente passa dall'utente CICS all'ID utente della transazione. Tuttavia, quando si esegue un'applicazione IBM MQ MQI client in CICS, essa conserva l'autorizzazione privilegiata CICS .

ALW *Programmi di esempio CICS e Tuxedo*

Programmi di esempio CICS e Tuxedo da utilizzare su sistemi AIX, Linux, and Windows .

Tabella 136 a pagina 936 elenca i programmi di esempio CICS e Tuxedo forniti per l'utilizzo su sistemi client AIX and Linux . Tabella 137 a pagina 937 elenca le informazioni equivalenti per i sistemi client Windows . Le tabelle elencano anche i file utilizzati per preparare ed eseguire i programmi. Per una descrizione dei programmi di esempio, vedere “L'esempio di transazione CICS” a pagina 1084 e “Utilizzo degli esempi di TUXEDO su AIX, Linux, and Windows” a pagina 1127.

Tabella 136. Programmi di esempio per sistemi client AIX and Linux

Descrizione	**Origine**	**Modulo eseguibile**
CICSProgramma	amqscic0.ccs	amqscicc
File di intestazione per il programma CICS	amqscih0.h	-
Programma client Tuxedo per inserire messaggi	amqstxpx.c	-
Programma client Tuxedo per richiamare i messaggi	amqstxgx.c	-
Programma del server Tuxedo per i due programmi client	amqstxsx.c	-
File UBBCONFIG per i programmi Tuxedo	ubbstxcx.cfg	-
File di tabella dei campi per i programmi Tuxedo	amqstxvx.flds	-

Tabella 136. Programmi di esempio per sistemi client AIX and Linux (Continua)		
Descrizione	**Origine**	**Modulo eseguibile**
Visualizza file di descrizione per i programmi Tuxedo	amqstxvx.v	-

Tabella 137. Programmi di esempio per sistemi client Windows		
Descrizione	**Origine**	**Modulo eseguibile**
CICS transazione	amqscic0.ccs	amqscicc
File di intestazione per la transazione CICS	amqscih0.h	-
Programma client Tuxedo per inserire messaggi	amqstxpx.c	-
Programma client Tuxedo per richiamare i messaggi	amqstxgx.c	-
Programma del server Tuxedo per i due programmi client	amqstxsx.c	-
File UBBCONFIG per i programmi Tuxedo	ubbstxcx.cfg	-
File di tabella dei campi per i programmi Tuxedo	amqstxvx.fld	-
Visualizza file di descrizione per i programmi Tuxedo	amqstxvx.v	-
Makefile per i programmi Tuxedo	amqstxmc.mak	-
File ENVFILE per i programmi Tuxedo	amqstxen.env	-

ALW *Messaggio di errore AMQ5203, come modificato per le applicazioni CICS e Tuxedo*

Quando si eseguono applicazioni CICS o Tuxedo che utilizzano un client transazionale esteso, è possibile che vengano visualizzati messaggi di diagnostica standard. Uno di questi è stato modificato per essere utilizzato con un client transazionale esteso

I messaggi che potrebbero essere visualizzati nei file di log degli errori di IBM MQ sono documentati in Messaggi diagnostici: AMQ4000-9999. Il messaggio AMQ5203 è stato modificato per l'utilizzo con un client transazionale esteso. Di seguito è riportato il testo del messaggio modificato:

AMQ5203: Si è verificato un errore durante il richiamo dell'interfaccia XA.

Spiegazione

Il numero di errore è & 2 dove il valore 1 indica che il valore degli indicatori forniti di & 1 non era valido, 2 indica che si è tentato di utilizzare le librerie con thread e non con thread nello stesso processo, 3 indica che si è verificato un errore con il nome gestore code fornito '& 3', 4 indica che l'ID gestore risorse di & 1 non è valido, 5 indica che si è tentato di utilizzare un secondo gestore code denominato '& 3 'quando un altro gestore code era già connesso, 6 indica che il Gestore transazioni è stato richiamato quando l'applicazione non è connessa a un gestore code, 7 indica che la chiamata XA è stata effettuata mentre era in corso un'altra chiamata, 8 indica che la stringa xa_info' & 4 'nella chiamata xa_open conteneva un valore di parametro non valido per il nome parametro' & 5 'e 9 indica che alla stringa xa_info' & 4 'nella chiamata xa_open manca un parametro obbligatorio, il nome parametro' & 5 '.

Risposta utente

Correggere l'errore e ripetere l'operazione.

Windows Preparazione ed esecuzione di applicazioni di Microsoft Transaction Server

Per preparare un'applicazione MTS da eseguire come un'applicazione IBM MQ MQI client , seguire queste istruzioni come appropriato per il proprio ambiente.

Per informazioni generali su come sviluppare applicazioni Microsoft Transaction Server (MTS) che accedono a risorse IBM MQ , consultare la sezione su MTS nel Centro assistenza IBM MQ .

Per preparare un'applicazione MTS da eseguire come un'applicazione IBM MQ MQI client , effettuare una delle seguenti operazioni per ogni componente dell'applicazione:

- Se il componente utilizza i bind in linguaggio C per MQI, seguire le istruzioni in “Preparazione dei programmi C in Windows” a pagina 1018 ma collegare il componente alla libreria mqicxa.lib invece di mqic.lib.
- Se il componente utilizza le classi C++ di IBM MQ , seguire le istruzioni in “Creazione di programmi C++ su Windows” a pagina 551 ma collegare il componente alla libreria imqx23vn.lib invece di imqc23vn.lib.
- Se il componente utilizza i bind del linguaggio Visual Basic per MQI, seguire le istruzioni in “Preparazione dei programmi Visual Basic in Windows” a pagina 1022 , ma quando si definisce il progetto Visual Basic, immettere `MqType=3` nel campo **Argomenti di compilazione condizionale** .

Preparazione ed esecuzione delle applicazioni IBM MQ JMS

È possibile eseguire applicazioni IBM MQ JMS in modalità client, con WebSphere Application Server come gestore transazioni. Potrebbero essere visualizzati alcuni messaggi di avvertenza.

Per preparare ed eseguire le applicazioni IBM MQ JMS in modalità client, con WebSphere Application Server come gestore transazioni, seguire le istruzioni in “Utilizzo di IBM MQ classes for JMS/Jakarta Messaging” a pagina 84.

Quando si esegue un'applicazione client IBM MQ JMS , potrebbero essere visualizzati i seguenti messaggi di avvertenza:

MQJE080
Unità di licenza insufficienti - eseguire setmqcap

MQJE081
Il file che contiene le informazioni sull'unità di licenza è nel formato errato - eseguire setmqcap

MQJE082
Impossibile trovare il file contenente le informazioni sull'unità di licenza - eseguire setmqcap

Uscite utente, uscite API e servizi installabili IBM MQ

Questo argomento contiene collegamenti alle informazioni sull'utilizzo e lo sviluppo di questi programmi.

Per un'introduzione su come utilizzare le uscite utente, le uscite API e i servizi installabili per estendere le funzionalità del gestore code, consultare Estensione delle funzioni del gestore code.

Per informazioni sulla scrittura e la compilazione di uscite e servizi installabili, consultare i topic secondari.

Concetti correlati
Programmi di uscita canale per canali MQI
Riferimenti correlati
Riferimento uscita API
Informazioni di riferimento per l'interfaccia dei servizi installabili
IBM i Informazioni di riferimento sull'interfaccia dei servizi installabili su IBM i

ALW Scrittura di uscite e servizi installabili su AIX, Linux, and Windows

È possibile scrivere e compilare uscite senza collegarsi ad alcuna libreria IBM MQ su AIX, Linux, and Windows.

Informazioni su questa attività

Questo argomento si applica solo a sistemi AIX, Linux, and Windows . Per dettagli sulla scrittura di uscite e servizi installabili per altre piattaforme, consultare gli argomenti specifici della piattaforma.

Se IBM MQ è installato in un percorso non predefinito, è necessario scrivere e compilare le uscite senza collegarsi ad alcuna libreria IBM MQ .

È possibile scrivere e compilare uscite su sistemi AIX, Linux, and Windows senza collegare alcuna delle seguenti librerie IBM MQ :

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

Le uscite esistenti collegate a queste librerie continuano a funzionare, a condizione che sui AIX and Linux sistemi IBM MQ sia installato nell'ubicazione predefinita.

Procedura

1. Includere il file di intestazione cmqec.h .

   L'inclusione di questo file di intestazione include automaticamente i file di intestazione cmqc.h, cmqxc.h e cmqzc.h .
2. Scrivere l'uscita in modo che le chiamate MQI e DCI vengano effettuate tramite la struttura MQIEP. Per ulteriori informazioni sulla struttura MQIEP, consultare Struttura MQIEP.
   - Servizi installabili
     - Utilizzare il parametro **Hconfig** per puntare alla chiamata MQZEP.
     - È necessario verificare che i primi 4 byte di **Hconfig** corrispondano al **StrucId** della struttura MQIEP prima di utilizzare il parametro **Hconfig** .
     - Per ulteriori informazioni sulla scrittura dei componenti del servizio installabili, consultare MQIEP.
   - Uscite API
     - Utilizzare il parametro **Hconfig** per puntare alla chiamata MQXEP.
     - È necessario verificare che i primi 4 byte di **Hconfig** corrispondano al **StrucId** della struttura MQIEP prima di utilizzare il parametro **Hconfig** .
     - Per ulteriori informazioni sulla scrittura delle uscite API, consultare “Scrittura delle uscite API” a pagina 956.
   - Uscite canale
     - Utilizzare il parametro **pEntryPoints** della struttura MQCXP per puntare a chiamate MQI e DCI.
     - È necessario verificare che il numero di versione MQCXP sia alla versione 8 o superiore prima di utilizzare **pEntryPoints**.
     - Per ulteriori informazioni sulla scrittura delle uscite di canale, consultare “Scrittura di programmi di uscita canale” a pagina 966.
   - Uscite di conversione dati
     - Utilizzare il parametro **pEntryPoints** della struttura MQDXP per puntare a chiamate MQI e DCI.
     - È necessario verificare che il numero di versione di MQDXP sia alla versione 2 o superiore prima di utilizzare **pEntryPoints**.
     - È possibile utilizzare il comandi **crtmqcvx** e il file di origine amqsvfc0.c per creare il codice di conversione dati che utilizza il parametro **pEntryPoints** . Consultare “Scrittura di un'uscita di conversione dati per IBM MQ for Windows” a pagina 993 e “Scrittura di un'uscita di conversione dati per i sistemi IBM MQ for AIX or Linux” a pagina 990.
     - Se si dispone di uscite di conversione dati esistenti che sono state generate utilizzando il comando **crtmqcvx** , è necessario rigenerare l'uscita utilizzando il comando aggiornato.

- Per ulteriori informazioni sulla scrittura delle uscite di conversione dati, consultare "Scrittura delle uscite di conversione dati" a pagina 985.

- Uscite di preconnessione
  - Utilizzare il parametro **pEntryPoints** della struttura MQNXP per puntare a chiamate MQI e DCI.
  - È necessario verificare che il numero di versione MQNXP sia alla versione 2 o superiore prima di utilizzare **pEntryPoints**.
  - Per ulteriori informazioni sulla scrittura delle uscite di pre - connessione, consultare "Riferimento alle definizioni di connessione mediante un'uscita di pre - connessione da un repository" a pagina 995.
- Uscite di pubblicazione
  - Utilizzare il parametro **pEntryPoints** della struttura MQPSXP per puntare a chiamate MQI e DCI.
  - È necessario verificare che il numero di versione di MQPSXP sia alla versione 2 o superiore prima di utilizzare **pEntryPoints**.
  - Per ulteriori informazioni sulla scrittura delle uscite di pubblicazione, consultare "Scrittura e compilazione di uscite di pubblicazione" a pagina 996.
- Uscite carico di lavoro cluster
  - Utilizzare il parametro **pEntryPoints** della struttura MQWXP per puntare a chiamate MQXCLWLN.
  - È necessario verificare che il numero di versione MQWXP sia alla versione 4 o superiore prima di utilizzare **pEntryPoints**.
  - Per ulteriori informazioni sulla scrittura delle uscite del carico di lavoro del cluster, consultare "Scrittura e compilazione delle uscite del carico di lavoro del cluster" a pagina 998.

Ad esempio, in un'uscita canale che richiama MQPUT:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Ulteriori esempi possono essere visualizzati in "Utilizzo dei programmi procedurali di esempio IBM MQ" a pagina 1060.

3. Compilare l'uscita:
   - Non collegarsi alle librerie IBM MQ .
   - Non includere un RPath integrato in alcuna libreria IBM MQ nella propria uscita.
   - Per ulteriori informazioni sulla compilazione dell'uscita, consultare uno dei seguenti argomenti:
     - Uscite API: "Compilazione delle uscite API" a pagina 958.
     - Uscite del canale, uscite di pubblicazione, uscite del carico di lavoro del cluster: "Compilazione dei programmi di uscita del canale su sistemi AIX, Linux, and Windows" a pagina 984.
     - Uscite di conversione dati: "Scrittura delle uscite di conversione dati" a pagina 985.
4. Inserire l'uscita in uno dei seguenti posti:
   - Un percorso di propria scelta che si qualifica completamente quando si configura l'exit
   - Il percorso di uscita predefinito, in una specifica directory di installazione. Ad esempio, *MQ_DATA_PATH*/exits/installation2.
   - Il percorso di uscita predefinito

     Il percorso di uscita predefinito è *MQ_DATA_PATH*/exits per le uscite a 32 bit e *MQ_DATA_PATH*/exits64 per le uscite a 64 bit. È possibile modificare questi percorsi nel file qm.ini o mqclient.ini .

Per ulteriori informazioni, consultare Percorso di uscita. In Windows e Linux, è possibile utilizzare IBM MQ Explorer per modificare il percorso:

a. Fare clic con il pulsante destro del mouse sul nome del gestore code
b. Fare clic su **Proprietà ...**
c. Fare clic su **Uscite**
d. Nel campo del percorso predefinito delle uscite, specificare il percorso della directory che contiene il programma di uscita.

Se un'uscita viene collocata sia in una directory di installazione specifica che nella directory del percorso predefinito, l'uscita della directory di installazione specifica viene utilizzata dall'installazione di IBM MQ indicata nel percorso. Ad esempio, l'uscita è collocata in `/exits/installation2` e `/exits`, ma non in `/exits/installation1`. L' IBM MQ installazione `installation2` utilizza l'uscita da `/exits/installation2`. L' IBM MQ installazione `installation1` utilizza l'uscita dalla directory `/exits` .

5. Se necessario, configurare l'uscita:
   - Servizi installabili “Configurazione di servizi e componenti” a pagina 949.
   - Uscite API: “Configurazione uscite API” a pagina 960.
   - Uscite canale: “Configurazione delle uscite canale” a pagina 985.
   - Uscite di pubblicazione: “Configurazione delle uscite di pubblicazione” a pagina 998.
   - Uscite di preconnessione: stanza PreConnect del file di configurazione client.

ALW *Uscite API non collegate ad una libreria MQI*

In determinate circostanze, è necessario collegare l'uscita API esistente, che non può essere ricodificata per utilizzare i puntatori della funzione MQIEP, con una libreria API IBM MQ .

Ciò è necessario, in modo che l'uscita API esistente possa essere caricata con esito positivo, dal programma di collegamento di runtime del sistema, in programmi che non hanno già i puntatori di funzione caricati.

Nota: Queste informazioni sono limitate alle uscite API esistenti che effettuano chiamate MQI direttamente. Ossia, le uscite che non utilizzano, MQIEP. Laddove possibile, è necessario pianificare di codificare nuovamente l'uscita per utilizzare i punti di entrata di MQIEP.

Da IBM MQ 8.0, **runmqsc** è un esempio di programma che non si collega direttamente con una libreria MQI.

Pertanto, un'uscita API che non è stata collegata alla libreria API IBM MQ richiesta o che non è stata ricodificata per utilizzare MQIEP, non riesce a caricarsi in **runmqsc**.

Vengono visualizzati errori nel log degli errori del gestore code, ad esempio, AMQ6175: `Il sistema non è stato in grado di caricare dinamicamente la libreria condivisa`, insieme al testo di qualificazione, ad esempio `undefined symbol: MQCONN`.

e AMQ7214: `Non è possibile caricare il modulo per l'uscita API 'myexitname'.`

Attività correlate

“Scrittura di uscite e servizi installabili su AIX, Linux, and Windows” a pagina 938
È possibile scrivere e compilare uscite senza collegarsi ad alcuna libreria IBM MQ su AIX, Linux, and Windows.

ALW Componenti e servizi installabili per AIX, Linux, and Windows

Questa sezione introduce i servizi installabili e le funzioni e i componenti ad essi associati. L'interfaccia per queste funzioni è documentata in modo che l'utente, o i fornitori di software, possano fornire componenti.

I principali motivi per fornire i servizi installabili IBM MQ sono:

- Per fornire la flessibilità di scegliere se utilizzare i componenti forniti dai prodotti IBM MQ o sostituirli o aumentarli con altri.
- Per consentire ai fornitori di partecipare, fornendo componenti che potrebbero utilizzare nuove tecnologie, senza apportare modifiche interne ai prodotti IBM MQ .
- Per consentire a IBM MQ di sfruttare le nuove tecnologie in modo più rapido ed economico, in modo da fornire i prodotti prima e a prezzi più bassi.

I *servizi installabili* e *componenti del servizio* fanno parte della struttura del prodotto IBM MQ . Al centro di questa struttura si trova la parte del gestore code che implementa la funzione e le regole associate alla MQI (Message Queue Interface). Questa parte centrale richiede una serie di funzioni di servizio, denominate *servizi installabili*, per eseguire il suo lavoro. I servizi installabili sono:

- Servizio di autorizzazione
- Servizio di denominazione

Ogni servizio installabile è una serie correlata di funzioni implementate utilizzando uno o più *componenti del servizio*. Ogni componente viene richiamato utilizzando un'interfaccia correttamente strutturata e disponibile al pubblico. Ciò consente ai fornitori di software indipendenti e ad altre terze parti di fornire componenti installabili per aumentare o sostituire quelli forniti dai prodotti IBM MQ . Tabella 138 a pagina 942 riepiloga i servizi e i componenti che possono essere utilizzati.

Tabella 138. Riepilogo dei componenti del servizio installabili

servizio installabile	Componente fornito	Funzione	Requisiti
Servizio di autorizzazione	object authority manager (OAM)	Fornisce il controllo di autorizzazione sui comandi e sulle chiamate MQI. Gli utenti possono scrivere il proprio componente per aumentare o sostituire l'OAM. Ad esempio, per verificare che un ID utente disponga dell'autorità per aprire una coda.	(Sono presupponenti le funzioni di autorizzazione della piattaforma appropriate)
Servizio di denominazione	Nessuna	Fornisce supporto al gestore code per la ricerca del nome del gestore code proprietario di una coda specificata. • Definito dall'utente	• Un gestore di nomi di terze parti o scritti dall'utente

L'interfaccia dei servizi installabili è descritta in Informazioni di riferimento sull'interfaccia dei servizi installabili.

Attività correlate

Configurazione dei servizi installabili

Scrittura di un componente del servizio

Questa sezione descrive la relazione tra servizi, componenti, punti di ingresso e codici di ritorno.

Funzioni e componenti

Ogni servizio è costituito da una serie di funzioni correlate. Ad esempio, il servizio dei nomi contiene la funzione per:

- Ricerca di un nome coda e restituzione del nome del gestore code in cui è definita la coda
- Inserimento di un nome coda nella directory del servizio

- Eliminazione di un nome coda dalla directory del servizio

Contiene anche funzioni di inizializzazione e di terminazione.

Un servizio installabile è fornito da uno o più componenti del servizio. Ogni componente può eseguire alcune o tutte le funzioni definite per quel servizio. Ad esempio, in IBM MQ for AIX, il componente del servizio di autorizzazione fornito, OAM, esegue tutte le funzioni disponibili. Per ulteriori informazioni, fare riferimento a "Interfaccia servizio di autorizzazione" a pagina 946. Il componente è anche responsabile della gestione di qualsiasi risorsa o software sottostante (ad esempio, una directory LDAP) di cui ha bisogno per implementare il servizio. I file di configurazione forniscono un modo standard di caricare il componente e determinare gli indirizzi delle routine funzionali che esso fornisce.

Figura 99 a pagina 943 mostra come i servizi e i componenti sono correlati:

- Un servizio viene definito per un gestore code dalle stanze in un file di configurazione.
- Ogni servizio è supportato dal codice fornito nel gestore code. Gli utenti non possono modificare questo codice e quindi non possono creare i propri servizi.
- Ogni servizio è implementato da uno o più componenti, che possono essere forniti con il prodotto o scritti dall'utente. È possibile richiamare più componenti per un servizio, ognuno dei quali supporta diverse funzioni all'interno del servizio.
- I punti di ingresso collegano i componenti di servizio al codice supportato nel gestore code.

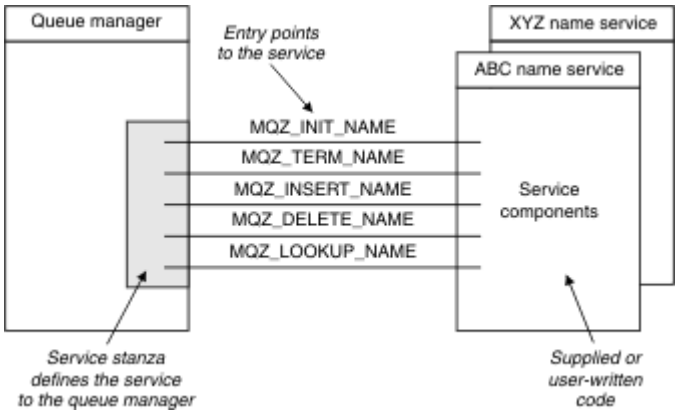


Figura 99. Informazioni su servizi, componenti e punti di ingresso

Punti di ingresso

Ciascun componente del servizio è rappresentato da un elenco di indirizzi del punto di ingresso delle routine che supportano un particolare servizio installabile. Il servizio installabile definisce la funzione che deve essere eseguita da ciascuna routine.

L'ordine dei componenti del servizio quando vengono configurati definisce l'ordine in cui i punti di ingresso vengono richiamati nel tentativo di soddisfare una richiesta per il servizio.

Nel file di intestazione fornito `cmqzc.h`, i punti di ingresso forniti per ogni servizio hanno un prefisso MQZID_.

Se i servizi sono presenti, vengono caricati in un ordine predefinito. Il seguente elenco mostra i servizi e l'ordine in cui vengono inizializzati.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` è il unico servizio configurato per impostazione predefinita. Configurare manualmente `NameService` e `UserIdentifierService` se si desidera utilizzarli.

I servizi e i componenti del servizio hanno una mappatura uno - a - uno o uno - a - molti. È possibile definire più componenti del servizio per ciascun servizio. Sui sistemi

AIX and Linux , il valore Servizio della sezione `ServiceComponent` deve corrispondere al valore Nome della stanza del servizio nel file qm.ini . Su Windows, il valore della chiave di registro del servizio di `ServiceComponent` deve essere uguale al valore della chiave di registro del nome ed è definito come: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\` dove *qmname* è il nome del gestore code.

Per sistemi AIX and Linux , i componenti del servizio vengono avviati nell'ordine in cui sono definiti nel file qm.ini . Su Windows, poiché viene utilizzato il registro Windows , IBM MQ emette una chiamata **RegEnumKey** che restituisce i valori in ordine alfabetico. Pertanto, su Windows i servizi vengono chiamati in ordine alfabetico, come sono definiti nel registro.

L'ordinamento delle definizioni `ServiceComponent` è significativo. Questo ordine indica l'ordine in cui i componenti vengono eseguiti per un dato servizio. Ad esempio, `AuthorizationService` su Windows è configurato con il componente predefinito OAM denominato `MQSeries.WindowsNT.auth.service`. È possibile definire ulteriori componenti per questo servizio per sovrascrivere l'OAM predefinito. A meno che non venga specificato `MQCACF_SERVICE_COMPONENT` , il primo componente rilevato in ordine alfabetico viene utilizzato per elaborare la richiesta e viene utilizzato il relativo nome.

Codici di ritorno

I componenti del servizio forniscono codici di ritorno al gestore code per creare report su varie condizioni. Riportano l'esito positivo o negativo dell'operazione e indicano se il gestore code deve procedere al successivo componente del servizio. Un parametro *Continuazione* separato contiene questa indicazione.

Dati componente

Un singolo componente del servizio potrebbe richiedere la condivisione dei dati tra le sue varie funzioni. I servizi installabili forniscono un'area dati facoltativa da trasmettere ad ogni richiamo di un componente del servizio. Questa area dati è destinata all'uso esclusivo del componente del servizio. È condiviso da tutte le chiamate di una particolare funzione, anche se sono effettuate da diversi spazi di indirizzo o processi. È garantito che sia indirizzabile dal componente del servizio ogni volta che viene richiamato. È necessario dichiarare la dimensione di questa area nella stanza *`ServiceComponent`* .

Inizializzazione e terminazione dei componenti
L'utilizzo delle opzioni di inizializzazione e di chiusura del componente.

Quando viene richiamata la routine di inizializzazione del componente, deve richiamare la funzione **MQZEP** del gestore code per ogni punto di ingresso supportato dal componente. **MQZEP** definisce un punto di ingresso per il servizio. Tutti i punti di uscita non definiti vengono considerati NULL.

Un componente viene sempre richiamato una sola volta con l'opzione di inizializzazione primaria, prima di essere richiamato in qualsiasi altro modo.

Un componente può essere richiamato con l'opzione di inizializzazione secondaria su alcune piattaforme. Ad esempio, può essere richiamato una volta per ogni processo, sottoprocesso o attività del sistema operativo da cui si accede al servizio.

Se viene utilizzata l'inizializzazione secondaria:

- Il componente può essere richiamato più di una volta per l'inizializzazione secondaria. Per ciascuna di queste chiamate, viene emessa una chiamata corrispondente per la terminazione secondaria quando il servizio non è più necessario.

  Per i servizi di denominazione, questa è la chiamata MQZ_TERM_NAME.

  Per i servizi di autorizzazione questa è la chiamata MQZ_TERM_AUTHORITY.
- I punti di ingresso devono essere nuovamente specificati (richiamando MQZEP) ogni volta che il componente viene richiamato per l'inizializzazione primaria e secondaria.
- Solo una copia dei dati del componente viene utilizzata per il componente; non esiste una copia diversa per ogni inizializzazione secondaria.

- Il componente non viene richiamato per altre chiamate al servizio (dal processo del sistema operativo, dal thread o dall'attività, come appropriato) prima che sia stata eseguita l'inizializzazione secondaria.
- Il componente deve impostare il parametro **Version** sullo stesso valore per l'inizializzazione primaria e secondaria.

Il componente viene sempre richiamato con l'opzione di terminazione primaria una volta, quando non è più richiesto. Non vengono effettuate ulteriori chiamate a questo componente.

Il componente viene richiamato con l'opzione di terminazione secondaria, se è stata richiamata per l'inizializzazione secondaria.

object authority manager (OAM)
Il componente del servizio di autorizzazione fornito con i prodotti IBM MQ è denominato OAM (Object Authority Manager).

Per impostazione predefinita, OAM è attivo e funziona con i comandi di controllo **dspmqaut** (autorizzazione di visualizzazione), **dmpmqaut** (autorizzazione di dump) e **setmqaut** (autorizzazione di impostazione o reimpostazione).

La sintassi di questi comandi e come utilizzarli sono descritti in Amministrazione IBM MQ for Multiplatforms mediante i comandi di controllo.

OAM funziona con l' *entità* di un principal o di un gruppo:

- Linux AIX Su sistemi AIX and Linux , un principal è un ID utente o un ID associato ad un programma applicativo in esecuzione per conto di un utente; un gruppo è una raccolta di principal definita dal sistema.
- Windows Su sistemi Windows , un principal è un ID utente Windows o un ID associato ad un programma applicativo in esecuzione per conto di un utente; un gruppo è un gruppo Windows .

Le autorizzazioni possono essere concesse o revocate a livello di principal o di gruppo.

Quando viene effettuata una richiesta MQI o viene emesso un comando, l'OAM controlla se l'entità associata all'operazione dispone dell'autorizzazione per eseguire l'operazione richiesta e per accedere alle risorse del gestore code specificate.

Il servizio di autorizzazione consente di convertire o sostituire il controllo dell'autorizzazione fornito per i gestori code scrivendo il componente del servizio di autorizzazione.

Servizio di denominazione
Il servizio dei nomi è un servizio installabile che fornisce supporto al gestore code per la ricerca del nome del gestore code che possiede una coda specificata. Nessun altro attributo della coda può essere richiamato da un servizio nomi.

Il servizio dei nomi consente a un'applicazione di aprire le code remote per l'emissione come se fossero code locali. Un servizio nomi non viene richiamato per oggetti diversi dalle code.

Nota: Le code remote devono avere il loro attributo **Scope** impostato su CELL.

Quando un'applicazione apre una coda, cerca il nome della coda nella directory del gestore code. Se non lo trova, cerca in tutti i servizi dei nomi che sono stati configurati, fino a quando non ne trova uno che riconosca il nome della coda. Se nessuno riconosce il nome, l'apertura ha esito negativo.

Il servizio dei nomi restituisce il gestore code proprietario per tale coda. Il gestore code continua quindi con la richiesta MQOPEN come se il comando avesse specificato il nome della coda e del gestore code nella richiesta originale.

L'NSI (name service interface) fa parte del framework IBM MQ .

Come funziona il servizio dei nomi

Se una definizione della coda specifica l'attributo **Scope** come gestore code, ossia SCOPE (QMGR) in MQSC, la definizione della coda (insieme a tutti gli attributi della coda) viene memorizzata solo nella directory del gestore code. Non può essere sostituito da un servizio installabile.

Se una definizione della coda specifica l'attributo **Scope** come cella, cioè SCOPE (CELL) in MQSC, la definizione della coda viene nuovamente memorizzata nella directory del gestore code, insieme a tutti gli attributi della coda. Tuttavia, anche il nome della coda e del gestore code vengono memorizzati in un servizio nomi. Se non è disponibile alcun servizio che possa memorizzare queste informazioni, non è possibile definire una coda con la cella *Scope* .

La directory in cui sono memorizzate le informazioni può essere gestita dal servizio oppure il servizio può utilizzare un servizio sottostante, ad esempio una directory LDAP, per questo scopo. In entrambi i casi, le definizioni memorizzate nella directory devono persistere, anche dopo che il componente e il gestore code sono stati terminati, fino a quando non vengono esplicitamente eliminati.

Nota:

1. Per inviare un messaggio alla definizione della coda locale di un host remoto (con un ambito CELL) su un gestore code differente all'interno di una cella di directory di denominazione, è necessario definire un canale.
2. Non è possibile ottenere i messaggi direttamente dalla coda remota, anche quando ha un ambito CELL.
3. Non è richiesta alcuna definizione di coda remota durante l'invio a una coda con ambito CELL.
4. Il servizio di denominazione definisce centralmente la coda di destinazione, anche se è ancora necessaria una coda di trasmissione per il gestore code di destinazione e una coppia di definizioni di canale. Inoltre, la coda di trasmissione sul sistema locale deve avere lo stesso nome del gestore code che possiede la coda di destinazione, con l'ambito della cella, sul sistema remoto.

   Ad esempio, se il gestore code remoto ha il nome QM01, la coda di trasmissione sul sistema locale deve avere anche il nome QM01.

Interfaccia servizio di autorizzazione
Il servizio di autorizzazione fornisce punti di ingresso per l'utilizzo da parte del gestore code.

I punti di ingresso sono i seguenti:

UTENTE_AUTENTICA_MQZ
Autentica un ID utente e una parola d'ordine e può impostare campi di contesto di identità.

MQZ_CHECK_AUTHORITY
Verifica se un'entità dispone dell'autorizzazione per eseguire una o più operazioni su un oggetto specificato.

MQZ_CHECK_PRIVILEGED
Verifica se un utente specificato è un utente privilegiato.

MQZ_COPY_ALL_AUTHORITY
Copia tutte le autorizzazioni correnti esistenti per un oggetto di riferimento in un altro oggetto.

MQZ_DELETE_AUTHORITY
Elimina tutte le autorizzazioni associate ad un oggetto specificato.

MQZ_ENUMERATE_AUTHORITY_DATA
Richiama tutti i dati di autorizzazione che corrispondono ai criteri di scelta specificati.

MQZ_FREE_UTENTE
Libera le risorse assegnate associate.

AUTORA_GET_MQZ
Ottiene l'autorità di cui dispone un'entità per accedere a un oggetto specificato.

MQZ_GET_EXPLICIT_AUTHORITY
Ottiene l'autorizzazione che un gruppo denominato ha per accedere a un oggetto specificato (ma senza l'autorizzazione aggiuntiva del gruppo **nobody**) o l'autorità che il gruppo primario del principal denominato ha per accedere a un oggetto specificato.

MQZ_INIT_AUTHORITY
Inizializza il componente del servizio di autorizzazione.

INQUIRE MQZ
Interroga la funzione supportata del servizio di autorizzazione.

MQZ_REFRESH_CACHE
Aggiornare tutte le autorizzazioni.

AUTORIZZAZIONE IMPOSTAZIONE MQZ
Imposta l'autorizzazione che un'entità ha su un oggetto specificato.

AUTORIZZAZIONE TERM MQZ
Termina il componente del servizio di autorizzazione.

Inoltre, su IBM MQ for Windows, il servizio di autorizzazione fornisce i seguenti punti di ingresso per l'utilizzo da parte del gestore code:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

Questi punti di ingresso supportano l'utilizzo del SID (Security Identifier) Windows .

Questi nomi sono definiti come **typedef** , nel file di intestazione `cmqzc.h`, che può essere utilizzato per creare il prototipo delle funzioni del componente.

La funzione di inizializzazione (**MQZ_INIT_AUTHORITY**) deve essere il punto di ingresso principale per il componente. Le altre funzioni vengono invocate tramite l'indirizzo del punto di entrata che la funzione di inizializzazione ha aggiunto al vettore del punto di ingresso del componente.

Interfaccia servizio nomi
Un servizio nomi fornisce i punti di ingresso per l'utilizzo da parte del gestore code.

Vengono forniti i seguenti punti di ingresso:

NOME_INIT_MQZ
Inizializza il componente servizio nomi.

NOME MQZ_TERM
Terminare il componente servizio nomi.

MQZ_NOME_RICERCA
Ricercare il nome del gestore code per la coda specificata.

NOME MQZ_INSERT
Inserire una voce contenente il nome del gestore code proprietario per la coda specificata nella directory utilizzata dal servizio.

NOME_ELIMINAZIONE_MQZ
Eliminare la voce per la coda specificata dalla directory utilizzata dal servizio.

Se è configurato più di un servizio nomi:

- Per la ricerca, la funzione MQZ_LOOKUP_NAME viene richiamata per ogni servizio nell'elenco fino a quando il nome della coda non viene risolto (a meno che un componente non indichi che la ricerca deve essere arrestata).
- Per inserimento, la funzione MQZ_INSERT_NAME viene richiamata per il primo servizio nell'elenco che supporta questa funzione.
- Per l'eliminazione, viene richiamata la funzione MQZ_DELETE_NAME per il primo servizio nell'elenco che supporta questa funzione.

Non dispone di più di un componente che supporta le funzioni di inserimento ed eliminazione. Tuttavia, un componente che supporta solo la ricerca è fattibile e può essere utilizzato, ad esempio, come ultimo

componente dell'elenco per risolvere qualsiasi nome che non sia noto a nessun altro componente del servizio nomi in un gestore code in cui è possibile definire il nome.

Nel linguaggio di programmazione C i nomi sono definiti come tipi di dati della funzione utilizzando l'istruzione typedef. Questi possono essere utilizzati per eseguire il prototipo delle funzioni di manutenzione, per garantire che i parametri siano corretti.

Il file di intestazione che contiene tutto il materiale specifico per i servizi installabili è `cmqzc.h` per il linguaggio C.

A parte la funzione di inizializzazione (MQZ_INIT_NAME), che deve essere il punto di ingresso principale del componente, le funzioni vengono richiamate dall'indirizzo del punto di ingresso aggiunto dalla funzione di inizializzazione, utilizzando la chiamata MQZEP.

Utilizzo di più componenti del servizio
È possibile installare più di un componente per un servizio. Ciò consente ai componenti di fornire solo implementazioni parziali del servizio e di fare affidamento su altri componenti per fornire le funzioni rimanenti.

Esempio di utilizzo di più componenti

Si supponga di creare due componenti dei servizi dei nomi denominati `ABC_name_serv` e `XYZ_name_serv`.

`ABC_name_serv`
Questo componente supporta l'inserimento o l'eliminazione di un nome dalla directory di servizio, ma non supporta la ricerca di un nome coda.

`XYZ_name_serv`
Questo componente supporta la ricerca di un nome coda, ma non supporta l'inserimento di un nome nella directory di servizio o l'eliminazione di un nome dalla directory di servizio.

Il componente `ABC_name_serv` contiene un database di nomi coda e utilizza due semplici algoritmi per inserire o eliminare un nome dalla directory di servizio.

Il componente `XYZ_name_serv` utilizza un algoritmo semplice che restituisce un nome gestore code fisso per qualsiasi nome coda con cui viene richiamato. Non contiene un database di nomi coda e quindi non supporta le funzioni di inserimento e di eliminazione.

I componenti vengono installati sullo stesso gestore code. Le stanze *`ServiceComponent`* vengono ordinate in modo che il componente `ABC_name_serv` venga richiamato per primo. Qualsiasi chiamata per inserire o eliminare una coda in una directory del componente viene gestita dal componente `ABC_name_serv` ; è l'unica che implementa queste funzioni. Tuttavia, una chiamata di ricerca che il componente `ABC_name_serv` non può risolvere viene passata al componente di sola ricerca, `XYZ_name_serv`. Questo componente fornisce un nome gestore code dal suo algoritmo semplice.

Omissione dei punti di ingresso quando si utilizzano più componenti

Se si decide di utilizzare più componenti per fornire un servizio, è possibile progettare un componente del servizio che non implementa determinate funzioni. Il framework dei servizi installabili non pone alcuna limitazione su cui è possibile omettere. Tuttavia, per servizi installabili specifici, l'omissione di una o più funzioni potrebbe essere logicamente incoerente con lo scopo del servizio.

Esempio di punti di ingresso utilizzati con più componenti

La Tabella 139 a pagina 949 mostra un esempio del servizio nomi installabile per cui sono stati installati i due componenti. Ognuno supporta una serie diversa di funzioni associate a questo particolare servizio installabile. Per la funzione di inserimento, viene richiamato prima il punto di ingresso del componente ABC. I punti di ingresso che non sono stati definiti per il servizio (utilizzando **MQZEP**) sono considerati NULL. Nella tabella viene fornito un punto di ingresso per l'inizializzazione, ma questo non è richiesto perché l'inizializzazione viene eseguita dal punto di ingresso principale del componente.

Quando il gestore code deve utilizzare un servizio installabile, utilizza i punti di ingresso definiti per tale servizio (le colonne in Tabella 139 a pagina 949). Prendendo ogni componente a turno, il gestore code determina l'indirizzo della routine che implementa la funzione richiesta. Richiama quindi la routine, se esiste. Se l'operazione ha esito positivo, i risultati e le informazioni sullo stato vengono utilizzati dal gestore code.

Tabella 139. Esempio di punti di ingresso per un servizio installabile

Numero funzione	**Componente servizio nomi ABC**	**Componente servizio nomi XYZ**
MQZID_INIT_NAME (Inizializza)	Inizializza ABC ()	XYZ_inizializza ()
MQZID_TERM_NAME (Termina)	Terminazione ABC ()	XYZ_termina ()
MQZID_INSERT_NAME (Inserimento)	Inseriment_ABC ()	NULL
MQZID_DELETE_NAME (Elimina)	Eliminazione ABC ()	NULL
MQZID_LOOKUP_NAME (Ricerca)	NULL	Ricerca XYZ ()

Se la routine non esiste, il gestore code ripete questo processo per il componente successivo nell'elenco. Inoltre, se la routine esiste ma restituisce un codice che indica che non è stato possibile eseguire l'operazione, il tentativo continua con il successivo componente disponibile. Le routine nei componenti del servizio potrebbero restituire un codice che indica che non devono essere effettuati ulteriori tentativi di esecuzione dell'operazione.

Configurazione di servizi e componenti

I componenti del servizio vengono configurati utilizzando i file di configurazione del gestore code, tranne sui sistemi Windows , in cui ciascun gestore code ha la propria stanza nel registro.

Procedura

1. Aggiungere le stanze al file di configurazione del gestore code `qm.ini`, per definire il servizio per il gestore code e specificare l'ubicazione del modulo:
   - Ogni servizio utilizzato deve avere una sezione `Service` , che definisce il servizio per il gestore code. Per ulteriori informazioni, consultare Stanza di servizio del file qm.ini.
   - Per ogni componente in un servizio, deve essere presente una stanza `ServiceComponent` . Questa stanza identifica nome e percorso del modulo contenente il codice per tale componente. Per ulteriori informazioni, consultare la stanza ServiceComponent del file qm.ini.

   Il componente servizio di autorizzazione, noto come OAM (Object Authority Manager), viene fornito con il prodotto. Quando si crea un gestore code, il file di configurazione del gestore code (o il registro sui sistemi Windows) viene aggiornato automaticamente in modo da includere le stanze appropriate per il servizio di autorizzazione e per il componente predefinito (OAM). Per gli altri componenti, è necessario configurare il file di configurazione del gestore code manualmente.

   Il codice per ciascun componente del servizio viene caricato nel gestore code quando il gestore code viene avviato, utilizzando il bind dinamico, dove è supportato sulla piattaforma.
2. Arrestare e riavviare il gestore code per attivare il componente.

Riferimenti correlati

Stanza di servizio del file qm.ini

Stanza ServiceComponent del file qm.ini

Aggiornamento di OAM dopo la modifica dell'autorizzazione di un utente

In IBM MQ, è possibile aggiornare le informazioni del gruppo di autorizzazioni di OAM immediatamente dopo aver modificato l'appartenenza del gruppo di autorizzazioni di un utente, riflettendo le modifiche apportate al livello del sistema operativo, senza dover arrestare e riavviare il gestore code. Per fare ciò, immettere il comando **REFRESH SECURITY** .

Nota: Quando si modificano le autorizzazioni con il comando `setmqaut` , OAM implementa tali modifiche immediatamente.

I gestori code memorizzano i dati di autorizzazione su una coda locale denominata SYSTEM.AUTH.DATA.QUEUE. Questi dati sono gestiti da **amqzfuma.exe**.

Riferimenti correlati

Aggiorna sicurezza

IBM i Servizi e componenti installabili su IBM i

Utilizzare queste informazioni per informazioni sui servizi installabili e sulle relative funzioni e componenti. L'interfaccia per queste funzioni è documentata in modo che l'utente, o i fornitori di software, possano fornire componenti.

I principali motivi per fornire i servizi installabili IBM MQ sono:

- Per fornire la flessibilità di scegliere se utilizzare i componenti forniti da IBM MQ for IBM io sostituirli o aumentarli con altri.
- Per consentire ai fornitori di partecipare, fornendo componenti che potrebbero utilizzare nuove tecnologie, senza apportare modifiche interne a IBM MQ for IBM i.
- Per consentire a IBM MQ di sfruttare le nuove tecnologie in modo più rapido ed economico, in modo da fornire i prodotti prima e a prezzi più bassi.

I *servizi installabili* e *componenti del servizio* fanno parte della struttura del prodotto IBM MQ . Al centro di questa struttura si trova la parte del gestore code che implementa la funzione e le regole associate alla MQI (Message Queue Interface). Questa parte centrale richiede una serie di funzioni di servizio, denominate *servizi installabili*, per eseguire il suo lavoro. Il servizio installabile disponibile in IBM MQ for IBM i è il servizio di autorizzazione.

Ogni servizio installabile è una serie correlata di funzioni implementate utilizzando uno o più *componenti del servizio*. Ogni componente viene richiamato utilizzando un'interfaccia correttamente strutturata e disponibile al pubblico. Ciò consente ai fornitori di software indipendenti e ad altre terze parti di fornire componenti installabili per aumentare o sostituire quelli forniti da IBM MQ for IBM i. Tabella 140 a pagina 950 riepiloga il supporto per il servizio di autorizzazione.

Tabella 140. Riepilogo componenti del servizio di autorizzazione

Componente fornito	Funzione	Requisiti
object authority manager (OAM)	Fornisce il controllo di autorizzazione sui comandi e sulle chiamate MQI. Gli utenti possono scrivere il proprio componente per aumentare o sostituire l'OAM.	(Sono presupponenti le funzioni di autorizzazione della piattaforma appropriate)
Componente servizio nomi DCE **Nota:** DCE è supportata solo su versioni di IBM MQ precedenti a V6.0.	• Consente ai gestori code di condividere le code oppure • Definito dall'utente **Nota:** Le code condivise devono avere l'attributo **Scope** impostato su CELL.	• DCE è richiesto per il componente fornito oppure • Un gestore di nomi di terze parti o scritti dall'utente

IBM i *Funzioni e componenti su IBM i*

Utilizzare queste informazioni per comprendere funzioni e componenti, punti di entrata, codici di ritorno e dati del componente che è possibile utilizzare in IBM MQ for IBM i.

Ogni servizio è costituito da una serie di funzioni correlate. Ad esempio, il servizio dei nomi contiene la funzione per:

- Ricerca di un nome coda e restituzione del nome del gestore code in cui è definita la coda

- Inserimento di un nome coda nella directory del servizio
- Eliminazione di un nome coda dalla directory del servizio

Contiene anche funzioni di inizializzazione e di terminazione.

Un servizio installabile è fornito da uno o più componenti del servizio. Ogni componente può eseguire alcune o tutte le funzioni definite per quel servizio. Il componente è anche responsabile della gestione di qualsiasi risorsa o software sottostante di cui ha bisogno per implementare il servizio. I file di configurazione forniscono un modo standard di caricare il componente e determinare gli indirizzi delle routine funzionali che esso fornisce.

I servizi e i componenti sono correlati come segue:

- Un servizio viene definito per un gestore code dalle stanze in un file di configurazione.
- Ogni servizio è supportato dal codice fornito nel gestore code. Gli utenti non possono modificare questo codice e quindi non possono creare i propri servizi.
- Ogni servizio è implementato da uno o più componenti, che possono essere forniti con il prodotto o scritti dall'utente. È possibile richiamare più componenti per un servizio, ognuno dei quali supporta diverse funzioni all'interno del servizio.
- I punti di ingresso collegano i componenti di servizio al codice supportato nel gestore code.

Punti di ingresso

Ciascun componente del servizio è rappresentato da un elenco di indirizzi del punto di ingresso delle routine che supportano un particolare servizio installabile. Il servizio installabile definisce la funzione che deve essere eseguita da ciascuna routine. L'ordine dei componenti del servizio quando vengono configurati definisce l'ordine in cui i punti di ingresso vengono richiamati nel tentativo di soddisfare una richiesta per il servizio. Nel file di intestazione fornito `cmqzc.h`, i punti di ingresso forniti per ogni servizio hanno un prefisso MQZID_.

Codici di ritorno

I componenti del servizio forniscono codici di ritorno al gestore code per creare report su una varietà di condizioni. Riportano l'esito positivo o negativo dell'operazione e indicano se il gestore code deve procedere al successivo componente del servizio. Un parametro *Continuazione* separato contiene questa indicazione.

Dati componente

Un singolo componente del servizio potrebbe richiedere la condivisione dei dati tra le sue varie funzioni. I servizi installabili forniscono un'area di dati facoltativa da trasmettere ad ogni richiamo di un particolare componente del servizio. Questa area dati è destinata all'uso esclusivo del componente del servizio. È condiviso da tutte le chiamate di una determinata funzione, anche se sono effettuate da diversi spazi di indirizzo o processi. È garantito che sia indirizzabile dal componente del servizio ogni volta che viene richiamato. È necessario dichiarare la dimensione di questa area nella stanza *`ServiceComponent`* .

IBM i *Inizializzazione su IBM i*

Quando viene richiamata la routine di inizializzazione del componente, è necessario richiamare la funzione MQZEP del gestore code per ciascun punto di ingresso supportato dal componente. MQZEP definisce un punto di ingresso per il servizio. Tutti i punti di uscita non definiti vengono considerati NULL.

Inizializzazione primaria
: Un componente viene sempre richiamato con questa opzione una volta, prima di essere richiamato in qualsiasi altro modo.

Inizializzazione secondaria
: Un componente può essere richiamato con questa opzione su determinate piattaforme. Ad esempio, può essere richiamato una volta per ogni processo, sottoprocesso o attività del sistema operativo da cui si accede al servizio.

Se viene utilizzata l'inizializzazione secondaria:

- Il componente può essere richiamato più di una volta per l'inizializzazione secondaria. Per ciascuna di queste chiamate, viene emessa una chiamata corrispondente per la terminazione secondaria quando il servizio non è più necessario.

  Per i servizi di autorizzazione questa è la chiamata MQZ_TERM_AUTHORITY.
- I punti di ingresso devono essere nuovamente specificati (richiamando MQZEP) ogni volta che il componente viene richiamato per l'inizializzazione primaria e secondaria.
- Solo una copia dei dati del componente viene utilizzata per il componente; non esiste una copia diversa per ogni inizializzazione secondaria.
- Il componente non viene richiamato per altre chiamate al servizio (dal processo del sistema operativo, dal thread o dall'attività, come appropriato) prima che sia stata eseguita l'inizializzazione secondaria.
- Il componente deve impostare il parametro **`Version`** sullo stesso valore per l'inizializzazione primaria e secondaria.

Terminazione primaria
Il componente viene sempre avviato con questa opzione una volta, quando non è più richiesto. Non vengono effettuate ulteriori chiamate a questo componente.

Terminazione secondaria
Il componente viene avviato con questa opzione, se è stato avviato per l'inizializzazione secondaria.

IBM i *Configurazione di servizi e componenti su IBM i*

Configurare i componenti del servizio utilizzando i file di configurazione del gestore code.

Procedura

1. Aggiungere le stanze al file di configurazione del gestore code `qm.ini`, per definire il servizio per il gestore code e specificare l'ubicazione del modulo:

   - Ogni servizio utilizzato deve avere una sezione `Service` , che definisce il servizio per il gestore code. Per ulteriori informazioni, consultare Stanza di servizio del file qm.ini.
   - Per ogni componente in un servizio, deve essere presente una stanza `ServiceComponent` . Questa stanza identifica nome e percorso del modulo contenente il codice per tale componente. Per ulteriori informazioni, consultare la stanza ServiceComponent del file qm.ini.

   Il componente servizio di autorizzazione, noto come OAM (Object Authority Manager), viene fornito con il prodotto. Quando si crea un gestore code, il file di configurazione del gestore code viene aggiornato automaticamente in modo da includere le stanze appropriate per il servizio di autorizzazione e per il componente predefinito (OAM). Per gli altri componenti, è necessario configurare il file di configurazione del gestore code manualmente.

   Il codice per ciascun componente del servizio viene caricato nel gestore code quando il gestore code viene avviato, utilizzando il bind dinamico, dove è supportato sulla piattaforma.
2.

IBM i *Creazione di un proprio componente del servizio su IBM i*

Utilizzare queste informazioni per informazioni su come creare un componente del servizio su IBM MQ for IBM i.

Per creare il proprio componente del servizio:

- Assicurarsi che il file di intestazione `cmqzc.h` sia incluso nel programma.
- Creare la libreria condivisa compilando il programma e collegandolo con le librerie condivise `libmqm*` e `libmqmzf*`.

  Nota: Poiché l'agente può essere eseguito in un ambiente con thread, è necessario creare l'OAM da eseguire in un ambiente con thread. Ciò include l'uso delle versioni con thread di `libmqm` e `libmqmzf`.

- Aggiungere le stanze al file di configurazione del gestore code per definire il servizio per il gestore code e specificare l'ubicazione del modulo.
- Arrestare e riavviare il gestore code per attivare il componente.

IBM i *Servizio di autorizzazione su IBM i*

Il servizio di autorizzazione è un servizio installabile che consente ai gestori code di richiamare le funzionalità di autorizzazione, ad esempio, controllando che l'ID utente disponga dell'autorità per aprire una coda.

Questo servizio è un componente di SEI (security abilita interface) IBM MQ , che fa parte del framework IBM MQ . Vengono discussi i seguenti argomenti:

- "object authority manager (OAM)" a pagina 953
- "Definizione del servizio per il sistema operativo" a pagina 953
- "Configurazione delle stanze del servizio di autorizzazione" a pagina 953
- "Interfaccia del servizio di autorizzazione su IBM i" a pagina 954

object authority manager (OAM)

Il componente del servizio di autorizzazione fornito con i prodotti IBM MQ è denominato OAM (object authority manager). Per impostazione predefinita, OAM è attivo e funziona con i comandi di controllo seguenti:

- **WRKMQMAUT** gestire l'autorità
- **WRKMQMAUTD** gestisce i dati di autorizzazione
- **DSPMQMAUT** visualizza autorizzazione oggetto
- **GRTMQMAUT** concedere l'autorizzazione all'oggetto
- **RVKMQMAUT** revoca autorizzazione oggetto
- **RFRMQMAUT** Aggiorna sicurezza

La sintassi di questi comandi e la relativa modalità di utilizzo sono descritti nella guida dei comandi CL. OAM funziona con l' *entità* di un principal o di un gruppo.

Quando viene effettuata una richiesta MQI o viene emesso un comando, OAM controlla l'autorizzazione dell'entità associata all'operazione per vedere se può eseguire le seguenti azioni:

- Eseguire l'operazione richiesta.
- Accedere alle risorse del gestore code specificate.

Il servizio di autorizzazione consente di convertire o sostituire il controllo dell'autorizzazione fornito per i gestori code scrivendo il componente del servizio di autorizzazione.

Definizione del servizio per il sistema operativo

Le stanze del servizio di autorizzazione nel file di configurazione del gestore code `qm.ini` definiscono il servizio di autorizzazione sul gestore code. Consultare "Configurazione di servizi e componenti su IBM i" a pagina 952 per informazioni sui tipi di stanza.

Configurazione delle stanze del servizio di autorizzazione

Su IBM MQ for IBM i:

Preside
È un profilo utente di sistema IBM i .

Gruppo
È un profilo gruppo di sistemi IBM i .

Le autorizzazioni possono essere concesse o revocate solo a livello di gruppo. Una richiesta di concessione o revoca dell'autorizzazione di un utente aggiorna il gruppo principale per tale utente.

Ogni gestore code ha il proprio file di configurazione del gestore code. Ad esempio, il percorso predefinito e il nome file del file di configurazione del gestore code per il gestore code QMNAME è `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

La stanza *`Service`* e la sezione *`ServiceComponent`* per il componente di autorizzazione predefinito vengono aggiunte automaticamente a `qm.ini` , ma possono essere sovrascritte da `WRKENVVAR`. Qualsiasi altra stanza *`ServiceComponent`* deve essere aggiunta manualmente.

Ad esempio, le seguenti stanze nel file di configurazione del gestore code definiscono due componenti del servizio di autorizzazione:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

Figura 100. Stanze del servizio di autorizzazione in qm.ini su IBM i

La prima stanza del componente di servizio `MQ.UNIX.authorization.service` definisce il componente del servizio di autorizzazione predefinito, OAM. Se si rimuove questa stanza e si riavvia il gestore code, l'OAM viene disabilitato e non viene eseguito alcun controllo di autorizzazione.

IBM i *Interfaccia del servizio di autorizzazione su IBM i*

L'interfaccia del servizio di autorizzazione fornisce diversi punti di ingresso che possono essere utilizzati dal gestore code.

UTENTE_AUTENTICA_MQZ
Autentica un ID utente e una parola d'ordine e può impostare campi di contesto di identità.

MQZ_CHECK_AUTHORITY
Verifica se un'entità dispone dell'autorizzazione per eseguire una o più operazioni su un oggetto specificato.

MQZ_COPY_ALL_AUTHORITY
Copia tutte le autorizzazioni correnti esistenti per un oggetto di riferimento in un altro oggetto.

MQZ_DELETE_AUTHORITY
Elimina tutte le autorizzazioni associate ad un oggetto specificato.

MQZ_ENUMERATE_AUTHORITY_DATA
Richiama tutti i dati di autorizzazione che corrispondono ai criteri di scelta specificati.

MQZ_FREE_UTENTE
Libera le risorse assegnate associate.

AUTORA_GET_MQZ
Ottiene l'autorità di cui dispone un'entità per accedere a un oggetto specificato.

MQZ_GET_EXPLICIT_AUTHORITY
Ottiene l'autorizzazione che un gruppo denominato ha per accedere a un oggetto specificato (ma senza l'autorizzazione aggiuntiva del gruppo **nobody**) o l'autorità che il gruppo primario del principal denominato ha per accedere a un oggetto specificato.

MQZ_INIT_AUTHORITY
Inizializza il componente del servizio di autorizzazione.

INQUIRE MQZ
Interroga la funzione supportata del servizio di autorizzazione.

MQZ_REFRESH_CACHE
Aggiornare tutte le autorizzazioni.

AUTORIZZAZIONE IMPOSTAZIONE MQZ
Imposta l'autorizzazione che un'entità ha su un oggetto specificato.

AUTORIZZAZIONE TERM MQZ
Termina il componente del servizio di autorizzazione.

Questi punti di ingresso supportano l'utilizzo del SID (Security Identifier) Windows .

Questi nomi sono definiti come **typedef** , nel file di intestazione `cmqzc.h`, che può essere utilizzato per creare il prototipo delle funzioni del componente.

La funzione di inizializzazione (**MQZ_INIT_AUTHORITY**) deve essere il punto di ingresso principale per il componente. Le altre funzioni vengono invocate tramite l'indirizzo del punto di entrata che la funzione di inizializzazione ha aggiunto al vettore del punto di ingresso del componente.

Per ulteriori informazioni, fare riferimento a "Creazione di un proprio componente del servizio su IBM i" a pagina 952.

Multi Scrittura e compilazione di uscite API su Multiplatforms

Le uscite API consentono di scrivere il codice che modifica il funzionamento delle chiamate API IBM MQ , come MQPUT e MQGET, e quindi inserire tale codice immediatamente prima o subito dopo tali chiamate.

Nota: z/OS Non supportato su IBM MQ for z/OS.

Perché utilizzare le uscite API?

Ciascuna delle applicazioni ha un lavoro specifico da eseguire e il relativo codice dovrebbe eseguire tale attività nel modo più efficiente possibile. Ad un livello superiore, è possibile applicare gli standard o i processi di business ad un particolare gestore code per tutte le applicazioni che utilizzano quel gestore code. È più efficiente eseguire questa operazione al di sopra del livello delle singole applicazioni, e quindi senza dover modificare il codice di ciascuna applicazione interessata.

Di seguito sono riportati alcuni suggerimenti di aree in cui le uscite API potrebbero essere utili:

Sicurezza
Per motivi di sicurezza, è possibile fornire l'autenticazione, controllando che le applicazioni siano autorizzate ad accedere a una coda o a un gestore code. È anche possibile controllare l'utilizzo dell'API da parte delle applicazioni, autenticando le singole chiamate API o anche i relativi parametri.

di business
Per maggiore flessibilità, è possibile rispondere alle rapide modifiche dell'ambiente di business senza modificare le applicazioni che si basano sui dati in tale ambiente. È possibile, ad esempio, disporre di uscite API che rispondono alle variazioni dei tassi di interesse, dei tassi di cambio o del prezzo dei componenti in un ambiente di produzione.

Monitoraggio dell'utilizzo di una coda o di un gestore code
Per monitorare l'utilizzo di una coda o di un gestore code, è possibile tracciare il flusso di applicazioni e messaggi, registrare gli errori nelle chiamate API, configurare le tracce di controllo per scopi di account o raccogliere le statistiche di utilizzo per scopi di pianificazione.

Cosa accade quando viene eseguita un'uscita API?

Dopo aver scritto un programma di uscita e averlo identificato in IBM MQ, il gestore code richiama automaticamente il codice di uscita nei punti registrati.

Le routine di uscita API da eseguire sono identificate nelle stanze su Multiplatforms. Questo argomento descrive le stanze nei file di configurazione `mqs.ini` e `qm.ini`.

La definizione delle routine può verificarsi in tre posizioni:

1. ApiExitCommon, nel file mqs.ini , identifica le routine, per l'intero IBM MQ, applicate all'avvio dei gestori code. Questi possono essere sovrascritti dalle routine definite per i singoli gestori code (vedere l'elemento “3” a pagina 956 in questo elenco).
2. Il modello ApiExit, nel file mqs.ini , identifica le routine, per l'intero IBM MQ, copiate nel set locale ApiExit(vedere la voce “3” a pagina 956 in questo elenco) quando viene creato un nuovo gestore code.
3. ApiExitlocale, nel file qm.ini , identifica le routine che si applicano a uno specifico gestore code.

Quando viene creato un nuovo gestore code, le definizioni di modello ApiExitin mqs.ini vengono copiate in ApiExitDefinizioni locali in qm.ini per il nuovo gestore code. Quando un gestore code viene avviato, vengono utilizzate sia le definizioni ApiExitCommon che ApiExitLocal. Le definizioni locali ApiExitsostituiscono le definizioni comuni ApiExitse entrambe identificano una routine con lo stesso nome. L'attributo Sequence , descritto in “Configurazione uscite API” a pagina 960 determina l'ordine di esecuzione delle routine definite nelle stanze.

Utilizzo delle uscite API su più installazioni di IBM MQ

Assicurarsi che le uscite API scritte per la versione precedente di IBM MQ siano utilizzate per gestire tutte le versioni, poiché le modifiche apportate alle uscite in IBM WebSphere MQ 7.1 potrebbero non funzionare con una versione precedente. Per ulteriori informazioni sulle modifiche apportate alle uscite, consultare “Scrittura di uscite e servizi installabili su AIX, Linux, and Windows” a pagina 938.

Gli esempi forniti per le uscite API amqsaem e amqsaxe riflettono le modifiche richieste durante la scrittura delle uscite. L'applicazione client deve garantire che le librerie IBM MQ corrette che corrispondono all'installazione del gestore code a cui è associata l'applicazione siano collegate ad essa prima dell'avvio dell'applicazione.

Multi *Scrittura delle uscite API*

È possibile scrivere uscite per ogni chiamata API utilizzando il linguaggio di programmazione C.

Uscite disponibili

Le uscite sono disponibili per ogni chiamata API, come segue:

- MQCB, per registrare nuovamente un callback per l'handle dell'oggetto specificato e controllare l'attivazione e le modifiche al callback
- MQCTL, per eseguire azioni di controllo sugli handle di oggetto aperti per una connessione
- MQCONN/MQCONNX, per fornire un handle di connessione del gestore code da utilizzare nelle chiamate API successive
- MQDISC, per disconnettersi da un gestore code
- MQBEGIN, per iniziare un'unità di lavoro globale (UOW)
- MQBACK, per eseguire il backout di una UOW
- MQCMIT, per eseguire il commit di una UOW
- MQOPEN, per aprire una risorsa IBM MQ per un accesso successivo
- MQCLOSE, per chiudere una risorsa IBM MQ che era stata precedentemente aperta per l'accesso
- MQGET, per richiamare un messaggio da una coda precedentemente aperta per l'accesso
- MQPUT1, per inserire un messaggio in una coda
- MQPUT, per inserire un messaggio in una coda precedentemente aperta per l'accesso
- MQINQ, per analizzare gli attributi di una risorsa IBM MQ precedentemente aperta per l'accesso
- MQSET, per impostare gli attributi di una coda che è stata precedentemente aperta per l'accesso
- MQSTAT, per recuperare le informazioni sullo stato
- MQSUB, per registrare la sottoscrizione delle applicazioni a un particolare argomento
- MQSUBRQ, per effettuare una richiesta di sottoscrizione

MQ_CALLBACK_EXIT fornisce una funzione di uscita da eseguire prima e dopo l'elaborazione del callback. Per ulteriori informazioni, consultare Callback - MQ_CALLBACK_EXIT.

Scrittura delle uscite API

All'interno delle uscite API, le chiamate assumono il seguente formato generale:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

dove *call* è il nome della chiamata MQI senza il prefisso MQ ; ad esempio, PUT, GET. Il *parameters* controlla la funzione dell'uscita, fornendo principalmente la comunicazione tra l'uscita e i blocchi di controllo esterni MQAXP (la struttura del parametro dell'uscita API) e MQAXC (la struttura del contesto dell'uscita API). *context* descrive il contesto in cui è stata richiamata l'uscita API e *ApiCallParameters* rappresenta i parametri per la chiamata MQI.

Per aiutarti a scrivere la tua uscita API, viene fornita un'uscita di esempio, amqsaxe0.c; questa uscita genera voci di traccia in un file che specifichi. È possibile utilizzare questo esempio come punto di partenza quando si scrivono le uscite. Per ulteriori informazioni sull'utilizzo dell'uscita di esempio, consultare "Il programma di esempio di uscita API" a pagina 1078.

Per ulteriori informazioni sulle chiamate API exit, sui blocchi di controllo esterni e sugli argomenti associati, vedi Guida di riferimento API exit.

Per informazioni generali su come scrivere, compilare e configurare un'uscita, vedere "Scrittura di uscite e servizi installabili su AIX, Linux, and Windows" a pagina 938.

Utilizzo degli handle dei messaggi nelle uscite API

È possibile controllare le proprietà del messaggio a cui ha accesso un'uscita API. Le proprietà sono associate a un handle ExitMsg. Le proprietà impostate in un'uscita di inserimento sono impostati sul messaggio da inserire, ma le proprietà richiamate in un'uscita di acquisizione non vengono restituite all'applicazione.

Quando si registra una funzione di uscita MQ_INIT_EXIT utilizzando la chiamata MQXEP MQI con **Function** impostato su MQXF_INIT e **ExitReason** impostato su MQXR_CONNECTION, si passa una struttura MQXEPO come parametro **ExitOpts** . La struttura MQXEPO contiene il campo ExitProperties , che specifica la serie di proprietà da rendere disponibili all'uscita. Viene specificato come una stringa di caratteri che rappresenta il prefisso delle proprietà, che corrisponde al nome di una cartella MQRFH2 .

Ogni uscita API riceve una struttura MQAXP, contenente un campo Handle ExitMsg. Questo campo è impostato su un valore generato da IBM MQ ed è specifico di un collegamento. L'handle non viene quindi modificato tra le uscite API dello stesso tipo o di tipi differenti sulla stessa connessione.

In MQ_PUT_EXIT o MQ_PUT1_EXIT con un **ExitReason** di MQXR_BEFORE, ossia un'uscita API eseguita prima di inserire un messaggio, tutte le proprietà (diverse dalle proprietà del descrittore del messaggio) associate all'handle ExitMsgquando l'uscita viene completata vengono impostate sul messaggio da inserire. Per evitare ciò, impostare ExitMsgHandle su MQHM_NONE. È anche possibile fornire un handle del messaggio differente.

In MQ_GET_EXIT e MQ_CALLBACK_EXIT, l'handle ExitMsgviene cancellato dalle proprietà e popolato con le proprietà specificate nel campo ExitProperties quando MQ_INIT_EXIT è stato registrato, diverse dalle proprietà del descrittore del messaggio. Queste proprietà non sono rese disponibili per l'applicazione di acquisizione. Se l'applicazione di richiamo ha specificato una gestione del messaggio nel campo MQGMO (Get message options), tutte le proprietà associate a tale gestione, incluse le proprietà del descrittore del messaggio, sono disponibili per l'uscita API. Per impedire che l'handle ExitMsgvenga popolato con le proprietà, impostarlo su MQHM_NONE.

Nota: Per le proprietà del messaggio di uscita da elaborare in:

- Una funzione MQ_GET_EXIT successiva, è necessario definire una funzione MQ_GET_EXIT precedente per l'exit.

- Prima della funzione MQ_CALLBACK_EXIT, è necessario definire una funzione MQ_CB_EXIT precedente per l'exit.

V 9.3.2 Da IBM MQ 9.3.2, le istruzioni precedenti relative a MQ-GET - EXIT e MQ_CALLBACK_EXIT non vengono più applicate.

Un programma di esempio, amqsaem0.c, viene fornito per illustrare l'utilizzo degli handle dei messaggi nelle uscite API.

Riferimenti correlati

Riferimenti alle uscite utente, alle uscite API e ai servizi installabili

Multi

Compilazione delle uscite API

Dopo aver scritto un'uscita, compilarla e collegarla nel modo seguente.

I seguenti esempi mostrano i comandi utilizzati per il programma di esempio descritto in “Il programma di esempio di uscita API” a pagina 1078. Per piattaforme diverse da sistemi Windows , è possibile trovare il codice di uscita API di esempio in *MQ_INSTALLATION_PATH*/samp e la libreria condivisa compilata e collegata in *MQ_INSTALLATION_PATH*/samp/bin.

Windows Per sistemi Windows , è possibile trovare il codice di uscita API di esempio in *MQ_INSTALLATION_PATH* \Tools\c\Samples. *MQ_INSTALLATION_PATH* rappresenta la directory in cui è stato installato IBM MQ .

Nota: La guida sulla programmazione di applicazioni a 64 bit è elencata in Standard di codifica su piattaforme a 64 bit.

Per i client Multicast, le uscite API e le uscite di conversione dati devono essere in grado di essere eseguite sul lato client perché alcuni messaggi potrebbero non passare attraverso il gestore code. Le seguenti librerie fanno parte dei pacchetti client e dei pacchetti server:

Tabella 141. Librerie che si trovano nei package client e server

Sistema operativo	**Librerie**
AIX AIX	32 bit & 64 bit: libmqm.a & libmqm_r.a
IBM i IBM i	LIBMQM & LIBMQM_R
Linux Linux	32 bit & 64 bit: libmqm.so & libmqm_r.so
Windows Windows	32 bit & 64 bit: mqm.dll & mqm.pdb

Linux AIX *Compilazione di uscite API su sistemi AIX and Linux*

Esempi di come compilare le uscite API sui sistemi AIX and Linux .

Su tutte le piattaforme, il punto di accesso al modulo è MQStart.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

AttivoAIX

AIX

Compilare il codice origine dell'uscita API immettendo uno dei seguenti comandi:

Applicazioni a 32 bit

Senza thread

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Threaded

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Applicazioni a 64 bit

Senza thread

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Threaded

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

AttivoLinux

Linux

Compilare il codice origine dell'uscita API immettendo uno dei seguenti comandi:

Applicazioni a 31 bit

Senza thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Threaded

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Applicazioni a 32 bit

Senza thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Threaded

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Applicazioni a 64 bit

Senza thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Threaded

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *Compilazione di uscite API su sistemi Windows*

Compilare e collegare il programma di uscita API di esempio, `amqsaxe0.c`, su Windows

Un file manifest è un documento XML facoltativo contenente la versione o qualsiasi altra informazione che può essere incorporata in un'applicazione compilata o DLL.

Se non hai tale documento, ometti il parametro -manifest *manifest.file* nel comando **mt** .

Adattare i comandi negli esempi in Figura 101 a pagina 960 o Figura 102 a pagina 960 per compilare e collegare `amqsaxe0.c` su Windows. I comandi funzionano con Microsoft Visual Studio 2008, 2010 o 2012. Gli esempi presuppongono che la directory `C:\Program Files\IBM\MQ\tools\c\samples` sia la directory corrente.

32 bit

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

Figura 101. Compila e collega `amqsaxe0.c` a 32 bit Windows

64 bit

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

Figura 102. Compila e collega `amqsaxe0.c` su 64 bit Windows

Concetti correlati

“Il programma di esempio di uscita API” a pagina 1078
L'uscita API di esempio genera una traccia MQI per un file specificato dall'utente con un prefisso definito nella variabile d'ambiente **MQAPI_TRACE_LOGFILE** .

IBM i *Completamento delle uscite API su IBM i*
Compilazione delle uscite API su IBM i.

Un'uscita viene creata come segue (per un esempio di linguaggio C):

1. Creare un modulo utilizzando CRTCMOD. Compilarlo per utilizzare teraspace includendo il parametro `TERASPACE(*YES *TSIFC)`.
2. Creare un programma di servizio dal modulo utilizzando CRTSRVPGM. È necessario collegarlo al programma di servizio QMQM/LIBMQMZF_R per le uscite API a più thread.

Configurazione uscite API

Configurare IBM MQ per abilitare le uscite API modificando le informazioni di configurazione.

Per modificare le informazioni di configurazione, è necessario modificare le stanze che definiscono le routine di uscita e la sequenza in cui vengono eseguite. Queste informazioni possono essere modificate nei seguenti modi:

- Linux Windows Utilizzo di IBM MQ Explorer su Windows e Linux (piattaformex86 e x86-64).
- Windows Utilizzando il comando **amqmdain** su Windows.
- Multi Utilizzando i file `mqs.ini` e `qm.ini` direttamente su Multiplatforms.

  Il file `mqs.ini` contiene informazioni relative a tutti i gestori code su un determinato nodo. È possibile trovarlo nelle seguenti ubicazioni:

  – Linux AIX Nella directory `/var/mqm` su AIX and Linux.

  – Windows Nel WorkPath specificato nella chiave `HKLM\SOFTWARE\IBM\WebSphere MQ` sui sistemi Windows .

  – IBM i Nella directory `/QIBM/UserData/mqm` su IBM i.

  Il file `qm.ini` contiene informazioni relative a uno specifico gestore code. È presente un file di configurazione del gestore code per ciascun gestore code, contenuto nella root della struttura di directory occupata dal gestore code. Ad esempio, il percorso e il nome di un file di configurazione per un gestore code denominato QMNAME è:

  IBM i Su sistemi IBM i :

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  Linux AIX Su sistemi AIX and Linux :

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows Su sistemi Windows :

  ```
  C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
  ```

  Prima di modificare un file di configurazione, eseguirne il backup in modo da disporre di una copia a cui è possibile tornare se necessario.

  È possibile modificare i file di configurazione in uno dei seguenti modi:

  – Automaticamente, utilizzando i comandi che modificano la configurazione dei gestori code sul nodo.

  – Manualmente, utilizzando un editor di testo standard.

  Se si imposta un valore non corretto su un attributo del file di configurazione, il valore viene ignorato e viene emesso un messaggio operatore per indicare il problema. L'effetto è lo stesso di perdere completamente l'attributo.

Stanze da configurare

Le stanze che devono essere modificate sono le seguenti:

ApiExitCommon

Definito in `mqs.ini` e in IBM MQ Explorer nella pagina delle proprietà IBM MQ , in Uscite.

Quando viene avviato un gestore code, gli attributi in questa stanza vengono letti e quindi sovrascritti dalle uscite API definite in `qm.ini`.

ApiExitTemplate

Definito in `mqs.ini` e in IBM MQ Explorer nella pagina delle proprietà IBM MQ , in Uscite.

Quando viene creato un qualsiasi gestore code, gli attributi in questa stanza vengono copiati nel file qm.ini appena creato nella stanza ApiExitlocale.

ApiExitLocal

Definito in `qm.ini` e in IBM MQ Explorer nella pagina delle proprietà del gestore code, in Uscite.

All'avvio del gestore code, le uscite API definite in questo punto sovrascrivono i valori predefiniti definiti in `mqs.ini`.

Attributi per le stanze

- Denominare l'uscita API utilizzando il seguente attributo:

  Nome=ApiExit_name
  Il nome descrittivo dell'uscita API inoltrato nel campo Nome ExitInfodella struttura MQAXP.

  Questo nome deve essere univoco, non deve superare i 48 caratteri e contenere solo caratteri validi per i nomi degli oggetti IBM MQ (ad esempio, nomi coda).

- Identificare il modulo e il punto di ingresso del codice di uscita API da eseguire utilizzando i seguenti attributi:

  Funzione=nome_funzione
  Il nome del punto di ingresso della funzione nel modulo contenente il codice di uscita API. Questo punto di ingresso è la funzione MQ_INIT_EXIT.

  La lunghezza di questo campo è limitata a MQ_EXIT_NAME_LENGTH.

  Modulo=nome_modulo
  Il modulo contenente il codice di uscita API.

  Se questo campo contiene il percorso completo del modulo, questo verrà visualizzato così come è.

  Se questo campo contiene solo il nome del modulo, il modulo si trova utilizzando l'attributo `ExitsDefaultPath` in `ExitPath` in qm.ini.

  Su piattaforme che supportano librerie separate, è necessario fornire sia una versione non con thread che una versione con thread del modulo di uscita API. La versione con thread deve avere un suffisso `_r` . La versione con thread dello stub dell'applicazione IBM MQ accoda implicitamente `_r` al nome modulo fornito prima che venga caricato.

  La lunghezza di questo campo è limitata alla lunghezza massima del percorso supportata dalla piattaforma.

- Facoltativamente, passare i dati con l'exit utilizzando il seguente attributo:

  Dati=nome_dati
  I dati da passare all'uscita API nel campo ExitData della struttura MQAXP.

  Se si include questo attributo, vengono rimossi gli spazi iniziali e finali, la stringa rimanente viene troncata a 32 caratteri e il risultato viene passato all'uscita. Se si omette questo attributo, il valore predefinito di 32 spazi viene passato all'uscita.

  La lunghezza massima di questo campo è 32 caratteri.

- Identificare la sequenza di questa uscita in relazione ad altre uscite utilizzando il seguente attributo:

  Sequenza=numero_sequenza
  La sequenza in cui questa uscita API viene richiamata rispetto ad altre uscite API. Un'uscita con un numero di sequenza basso viene richiamata prima di un'uscita con un numero di sequenza più alto. Non è necessario che la numerazione di sequenza delle uscite sia contigua. Una sequenza di 1, 2, 3 ha lo stesso risultato di una sequenza di 7, 42, 1096. Se due uscite hanno lo stesso numero di sequenza, il gestore code decide quale richiamare per primo. È possibile determinare quale

elemento è stato richiamato dopo l'evento inserendo l'ora o un indicatore nell'area ExitChainindicata da ExitChainAreaPtr in MQAXP oppure scrivendo il proprio file di log.

Questo attributo è un valore numerico senza segno.

Stanze di esempio

Il file mqs.ini di esempio contiene le seguenti stanze:

ApiExitTemplate
Questa stanza definisce un'uscita con il nome descrittivo `OurPayrollQueueAuditor`, il nome modulo `auditore` il numero di sequenza 2. Un valore dati di 123 viene passato all'uscita.

ApiExitCommon
Questa stanza definisce un'uscita con il nome descrittivo `MQPoliceman`, il nome modulo `tmqpe` il numero di sequenza 1. I dati passati sono un'istruzione (`CheckEverything`).

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

Il seguente file qm.ini di esempio contiene una definizione locale ApiExitdi un'uscita con nome descrittivo `ClientApplicationAPIchecker`, nome modulo `ClientAppCheckere` numero di sequenza 3.

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

Programmi di uscita canale per canali di messaggistica

Questa raccolta di argomenti contiene informazioni relative ai programmi di uscita canale IBM MQ per i canali di messaggistica.

Gli MCA (Message Channel Agent) possono anche richiamare le uscite di conversione dati. Per ulteriori informazioni sulla scrittura delle uscite di conversione dati, consultare “Scrittura delle uscite di conversione dati” a pagina 985.

Alcune di queste informazioni si applicano alle uscite sui canali MQI, che connettono IBM MQ MQI clients ai gestori code. Per ulteriori informazioni, consultare Channel - exit programs for MQI channels.

I programmi di uscita canale vengono richiamati in punti definiti nell'elaborazione eseguita dai programmi MCA.

Alcuni di questi programmi di uscita utente funzionano in coppie complementari. Ad esempio, se un programma di uscita utente viene richiamato dall'MCA mittente per codificare i messaggi per la trasmissione, il processo complementare deve funzionare all'estremità ricevente per invertire il processo.

La Tabella 142 a pagina 964 mostra i tipi di uscita canale disponibili per ogni tipo di canale.

Tabella 142. Uscite canale disponibili per ciascun tipo di canale

Tipo canale	Uscita messaggi	Uscita nuovo tentativo messaggio	Uscita ricezione	Uscita di sicurezza	Uscita invio	Uscita definizione automatica
Canale di trasmissione	Sì		Sì	Sì	Sì	
Canale server	Sì		Sì	Sì	Sì	
Canale mittente cluster	Sì		Sì	Sì	Sì	Sì
Canale di ricezione	Sì	Sì	Sì	Sì	Sì	Sì
Canale richiedente	Sì	Sì	Sì	Sì	Sì	
Canale ricevente cluster	Sì	Sì	Sì	Sì	Sì	Sì
Canale di connessione client			Sì	Sì	Sì	
Canale di connessione server			Sì	Sì	Sì	Sì

Note: z/OS

1. Su z/OS, l'uscita di definizione automatica si applica solo a canali mittente cluster e ricevente cluster.

Se si stanno per eseguire uscite di canale su un client, non è possibile utilizzare la variabile di ambiente MQSERVER. Invece, creare e fare riferimento a una tabella di definizione del canale client (CCDT) come descritto in Tabella di definizione del canale client.

Panoramica sull'elaborazione

Una panoramica di come gli MCA utilizzano i programmi channel - exit.

All'avvio, gli MCA scambiano una finestra di dialogo di avvio per sincronizzare l'elaborazione. Quindi passano a uno scambio di dati che include le uscite di sicurezza. Queste uscite devono terminare correttamente per completare la fase di avvio e consentire il trasferimento dei messaggi.

La fase di verifica di sicurezza è un ciclo, come mostrato in Figura 103 a pagina 965.

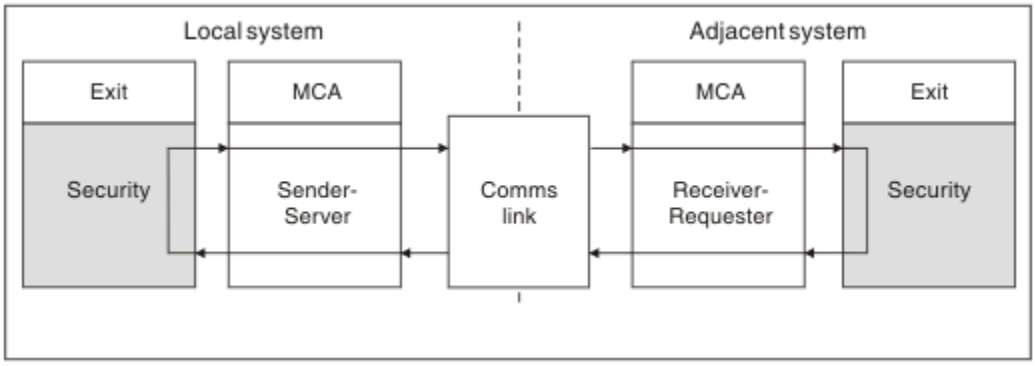


Figura 103. Loop uscita di sicurezza

Durante la fase di trasferimento del messaggio, l'MCA mittente riceve i messaggi da una coda di trasmissione, richiama l'uscita del messaggio, richiama l'uscita di invio e quindi invia il messaggio all'MCA ricevente, come mostrato in Figura 104 a pagina 965.

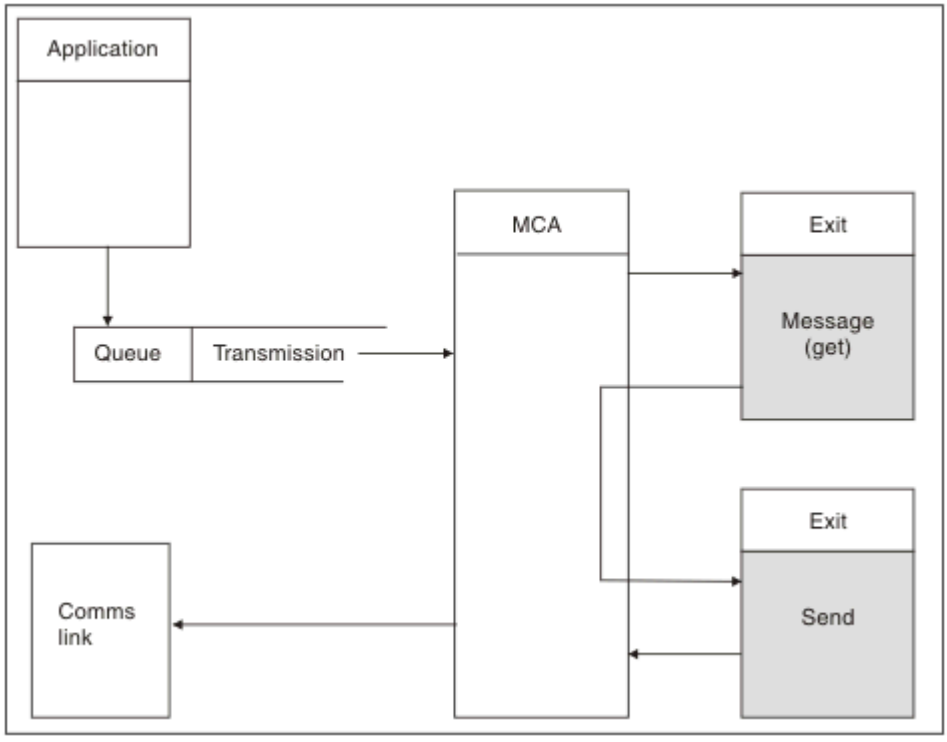


Figura 104. Esempio di uscita di invio all'estremità mittente del canale di messaggi

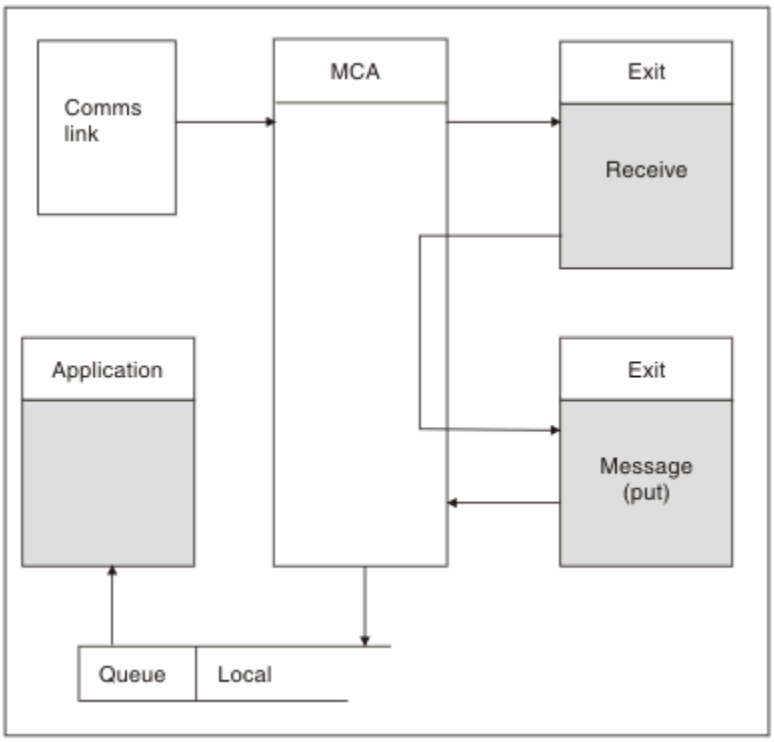


Figura 105. Esempio di un'uscita di ricezione all'estremità del destinatario del canale di messaggi

L'MCA ricevente riceve un messaggio dal collegamento di comunicazione, richiama l'uscita di ricezione, richiama l'uscita di messaggio e inserisce il messaggio sulla coda locale, come mostrato in Figura 105 a pagina 966. (L'uscita di ricezione può essere richiamata più di una volta prima che venga richiamata l'uscita del messaggio.)

Scrittura di programmi di uscita canale

È possibile utilizzare le seguenti informazioni per scrivere programmi di uscita canale.

Le uscite utente e i programmi di uscita canale possono utilizzare tutte le chiamate MQI, tranne come indicato nelle sezioni che seguono. Per MQ V7 e versioni successive, la struttura MQCXP versione 7 e successive contiene l'handle di connessione hConn, che può essere utilizzato invece di emettere MQCONN. Per le versioni precedenti, per ottenere l'handle di connessione, è necessario emettere un MQCONN, anche se viene restituita un'avvertenza MQRC_ALREADY_CONNECTED perché il canale stesso è connesso al gestore code.

Notare che l'uscita del canale deve essere thread - safe.

Per le uscite sui canali di connessione client, il gestore code a cui l'exit tenta di connettersi dipende dal modo in cui l'exit è stata collegata. Se l'uscita è stata collegata a MQM.LIB (o QMQM/LIBMQM su IBM i) e non si specifica un nome gestore code nella chiamata MQCONN, l'exit tenta di connettersi al gestore code predefinito sul sistema. Se l'uscita è stata collegata a MQM.LIB (o QMQM/LIBMQM su IBM i) e si specifica il nome del gestore code passato all'uscita tramite il campo QMgrName di MQCD, l'uscita tenta di connettersi a tale gestore code. Se l'uscita è stata collegata a MQIC.LIB o qualsiasi altra libreria, la chiamata MQCONN ha esito negativo se si specifica o meno un nome gestore code.

Si consiglia di evitare di modificare lo stato della transazione associata al hConn inoltrato in un'uscita del canale; non è necessario utilizzare i verbi MQCMIT, MQBACK o MQDISC con il canale hConne non è possibile utilizzare il verbo MQBEGIN specificando il canale hConn.

Se si utilizza MQCONNX specificando MQCNO_HANDLE_SHARE_BLOCK o MQCNO_HANDLE_SHARE_NO_BLOCK per creare una nuova connessione IBM MQ , è responsabilità dell'utente assicurarsi che la connessione sia gestita correttamente e che si disconnetta correttamente dal gestore code. Ad esempio, un'uscita del canale che crea una nuova connessione al gestore code ad

ogni chiamata senza disconnettersi, determina la creazione di handle di connessione e un aumento del numero di thread dell'agent.

Un'uscita viene eseguita nello stesso thread dell'MCA stesso e utilizza lo stesso handle di connessione. Quindi, viene eseguito all'interno della stessa UOW dell'MCA e tutte le chiamate effettuate nel punto di sincronizzazione vengono sottoposte a commit o a backout dal canale alla fine del batch.

Pertanto, un'uscita messaggio del canale potrebbe inviare messaggi di notifica di cui è stato eseguito il commit solo su quella coda quando è stato eseguito il commit del batch contenente il messaggio originale. Quindi, è possibile emettere le chiamate MQI del punto di sincronizzazione da un'uscita del messaggio del canale.

Un'uscita canale può cambiare i campi in MQCD. Tuttavia, tali modifiche non vengono applicate, ad eccezione delle circostanze elencate. Se un programma di uscita del canale modifica un campo nella struttura dati MQCD, il nuovo valore viene ignorato dal processo del canale IBM MQ . Tuttavia, il nuovo valore rimane nel MQCD e viene passato a tutte le uscite rimanenti in una catena di uscita e a tutte le conversazioni che condividono l'istanza del canale. Per ulteriori informazioni, consultare Modifica dei campi MQCD in un'uscita canale

Inoltre, per i programmi scritti in C, la funzione libreria C non rientrante non deve essere utilizzata in un programma di uscita canale.

Linux AIX Se si utilizzano più librerie di uscita del canale contemporaneamente, possono verificarsi problemi su alcune piattaforme UNIX and Linux se il codice per due diverse uscite contiene funzioni con lo stesso nome. Quando viene caricata un'uscita canale, il programma di caricamento dinamico risolve i nomi delle funzioni nella libreria di uscita negli indirizzi in cui viene caricata la libreria. Se due librerie di uscita definiscono funzioni separate che hanno nomi identici, questo processo di risoluzione potrebbe risolvere in modo non corretto i nomi funzione di una libreria per utilizzare le funzioni di un'altra. Se si verifica questo problema, specificare al linker che deve solo esportare le funzioni di uscita e MQStart richieste, poiché queste funzioni non sono interessate. Altre funzioni devono avere una visibilità locale in modo che non vengano utilizzate da funzioni esterne alla propria libreria di uscita. Consultare la documentazione per il linker per ulteriori informazioni.

Tutte le uscite sono richiamate con una struttura di parametri di uscita del canale (MQCXP), una struttura di definizione del canale (MQCD), un buffer di dati preparato, un parametro di lunghezza dati e un parametro di lunghezza del buffer. La lunghezza del buffer non deve essere superata:

- Per le uscite dei messaggi, è necessario consentire l'invio del messaggio più grande richiesto attraverso il canale, più la lunghezza della struttura MQXQH.
- Per le uscite di invio e ricezione, il buffer più grande che è necessario consentire è il seguente:

  LU 6.2
  32 KB

  TCP:

  IBM i IBM i 16 KB

  IBM i Altri 32 KB

  Nota: La lunghezza massima utilizzabile potrebbe essere di 2 byte inferiore a questa lunghezza. Verificare il valore restituito in MaxSegmentLength per i dettagli. Per ulteriori informazioni sulla lunghezza MaxSegment, consultare MaxSegmentLength.

  NetBIOS:
  64 KB

  SPX:
  64 KB

  Nota: Le uscite di ricezione sui canali mittenti e le uscite mittente sui canali riceventi utilizzano buffer di 2 KB per TCP.
- Per le uscite di sicurezza, la funzione di accodamento distribuito assegna un buffer di 4000 byte.

È consentito che l'uscita restituisca un buffer alternativo, insieme ai parametri pertinenti. Consultare “Programmi di uscita canale per canali di messaggistica” a pagina 963 per i dettagli della chiamata.

z/OS *Scrittura dei programmi di uscita del canale su z/OS*

È possibile utilizzare le seguenti informazioni per scrivere e compilare programmi di uscita canale per z/OS.

Le uscite vengono avviate come da un LINK z/OS , in:

- Stato programma problema non autorizzato
- Modalità di controllo spazio di indirizzo principale
- Modalità non cross - memory
- Modalità registro di non accesso
- modalità di indirizzamento a 31 bit

I moduli modificati dal link devono essere inseriti nel dataset specificato dall'istruzione CSQXLIB DD della procedura dello spazio di indirizzi dell'iniziatore di canali; i nomi dei moduli di caricamento sono specificati come nomi di uscita nella definizione del canale.

Quando si scrivono uscite canale per z/OS, si applicano le seguenti regole:

- Le uscite devono essere scritte in assembler o in C; se viene utilizzato C, deve essere conforme all'ambiente di programmazione dei sistemi C per le uscite di sistema, descritto nel manuale z/OS C/C++ Programming Guide.
- Le uscite vengono caricate dalle librerie non autorizzate definite da un'istruzione DD CSQXLIB. Se CSQXLIB ha DISP=SHR, le uscite possono essere aggiornate mentre l'iniziatore del canale è in esecuzione. La nuova versione viene utilizzata quando il canale viene riavviato.
- Le uscite devono essere rientranti e in grado di essere eseguite ovunque nella memoria virtuale.
- Le uscite devono reimpostare l'ambiente, al ritorno, su quello alla voce.
- Le uscite devono liberare la memoria ottenuta o assicurarsi che venga liberata da un successivo richiamo di uscita.

  Per l'archiviazione che deve essere conservata tra i richiami, utilizzare il servizio z/OS STORAGE o la funzione della libreria 4kmalc per System Programming C.

  Per ulteriori informazioni su questa funzione, consultare 4kmalc() -- Allocate Page - Aligned Storage.
- È possibile utilizzare tutte le chiamate MQI IBM MQ tranne MQCMIT o CSQBCMT e MQBACK o CSQBBAK. Devono essere contenuti dopo MQCONN (con un nome gestore code vuoto). Se queste chiamate vengono utilizzate, l'uscita deve essere modificata tramite link con lo stub CSQXSTUB.

  L'eccezione a questa regola è che le uscite del canale di protezione possono emettere chiamate MQI di commit e backout. Per emettere tali chiamate, codificare i verbi CSQXCMT e CSQXBAK al posto di MQCMIT o CSQBCMT e MQBACK o CSQBBAK.
- Tutte le uscite che utilizzano stub CSQXSTUB da IBM WebSphere MQ 7.0 o versioni successive devono essere modificate tramite link in una libreria di caricamento CSQXLIB con formato PDS - E.
- Le uscite non devono utilizzare i servizi di sistema che causano un'attesa, poiché l'utilizzo dei servizi di sistema influirebbe in modo grave sulla gestione di alcuni o di tutti gli altri canali. Molti canali vengono eseguiti in un singolo TCB in genere. Se si esegue un'operazione in un'uscita che causa un'attesa e non si utilizza MQXWAIT, si verifica un'attesa di tutti questi canali. L'attesa dei canali non dà alcun problema funzionale, ma potrebbe avere un effetto negativo sulle prestazioni. La maggior parte degli SVC coinvolgono le attese, quindi è necessario evitarle, ad eccezione dei seguenti SVC:
  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

  In generale, quindi, evitare SVC, PC e I/O. Utilizzare invece la chiamata MQXWAIT.
- Le uscite non emettono ESTAEs o SPIEs, ad eccezione delle attività secondarie che collegano, poiché la loro gestione degli errori potrebbe interferire con la gestione degli errori eseguita da IBM MQ. Ciò

significa che IBM MQ potrebbe non essere in grado di eseguire il ripristino da un errore o che il programma di uscita potrebbe non ricevere tutte le informazioni sull'errore.

- La chiamata MQXWAIT (consultare MQXWAIT). fornisce un servizio di attesa che attende I/O e altri eventi; se questo servizio viene utilizzato, le uscite non devono utilizzare lo stack di collegamento.

  Per I/O e altre funzioni che non forniscono funzioni non bloccanti o una BCE su cui attendere, è necessario ATTACHed una attività secondaria separata e il relativo completamento atteso da MQXWAIT; a causa dell'elaborazione che questa tecnica comporta, questa funzione deve essere utilizzata solo dall'uscita di sicurezza.

- La chiamata MQDISC MQI non causa un commit implicito all'interno del programma di uscita. Un commit del processo del canale viene eseguito solo quando il protocollo del canale lo richiede.

I seguenti esempi di uscita vengono forniti con IBM MQ for z/OS:

CSQ4BAX0
Questo esempio è scritto in assembler e illustra l'utilizzo di MQXWAIT.

CSQ4BCX1 e CSQ4BCX2 CSQ4BCX2
Questi esempi sono scritti in C e illustrano come accedere ai parametri.

CSQ4BCX3 e CSQ4BAX3
Questi campioni sono scritti rispettivamente in C e in assembler.

L'esempio CSQ4BCX3 (che è precompilato in SCSQAUTH LOADLIB, dovrebbe funzionare senza alcuna modifica necessaria sull'uscita stessa. È possibile creare una LOADLIB (ad esempio, denominata MY.TEST.LOADLIB) e copiare il membro SCSQAUTH (CSQ4BCX3).

Per impostare un'uscita di sicurezza su una connessione client, attenersi alla seguente procedura:

1. Stabilire un segmento OMVS valido per l'ID utente utilizzato dall'iniziatore di canali.

   Ciò consente all'iniziatore di canali IBM MQ for z/OS di utilizzare TCP/IP con l'interfaccia del socket z/OS UNIX System Services (z/OS UNIX), per facilitare l'elaborazione dell'uscita. Si noti che non è necessario definire un segmento OMVS per l'ID utente di qualsiasi client di connessione.
2. Assicurarsi che il codice di uscita stesso venga eseguito solo in un ambiente controllato dal programma.

   Ciò significa che tutto ciò che viene caricato nello spazio di indirizzi CHINIT deve essere caricato da una libreria controllata dal programma (ovvero tutte le librerie in STEPLIB) e tutte le librerie denominate in CSQXLIB e

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   Per impostare una libreria di caricamento come controllata dal programma, utilizzare un comando simile a questo esempio:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   Quindi, è possibile attivare o aggiornare l'ambiente controllato dal programma immettendo il seguente comando:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```
3. Aggiungere l'uscita LOADLIB alla DD CSQXLIB (nella procedura avviata CHINIT), immettendo il seguente comando:

   ```
   ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
   ```

   Questo attiva l'uscita per il canale specificato.

4. L'ESM (External Security Manager) elenca tutte le altre librerie da controllare dal programma, ma si noti che nessuna delle librerie ESM o C deve essere sotto il controllo del programma.

Consultare IBM MQ for z/OS canale di connessione server per ulteriori informazioni sull'impostazione di un'uscita di sicurezza utilizzando l'esempio CSQ4BCX3.

CSQ4BCX4
Questo esempio è scritto in C e illustra l'utilizzo dei campi di **RemoteProduct** e **RemoteVersion** in MQCXP.

Concetti correlati

"Scrittura dei programmi di uscita del canale su IBM i" a pagina 970
È possibile utilizzare le seguenti informazioni per scrivere e compilare programmi di uscita canale per IBM i.

"Scrittura di programmi di uscita canale su AIX, Linux, and Windows" a pagina 971
È possibile utilizzare le seguenti informazioni per scrivere programmi di uscita del canale per sistemi AIX, Linux, and Windows .

Riferimenti correlati

IBM MQ for z/OS Canale di connessione server

Scrittura dei programmi di uscita del canale su IBM i

È possibile utilizzare le seguenti informazioni per scrivere e compilare programmi di uscita canale per IBM i.

L'uscita è un oggetto programma scritto nel linguaggio ILE C, ILE RPG o ILE COBOL. I nomi del programma di uscita e le relative librerie vengono denominati nella definizione di canale.

Osservare le seguenti condizioni durante la creazione e la compilazione di un programma di uscita:

- Il programma deve essere reso protetto dal sottoprocesso e creato con il compilatore ILE C, ILE RPG o ILE COBOL. Per ILE RPG è necessario specificare la specifica di controllo THREAD (*SERIALIZE) e per ILE COBOL è necessario specificare SERIALIZE per l'opzione THREAD dell'istruzione PROCESS. I programmi devono anche essere collegati alle librerie IBM MQ con thread: QMQM/LIBMQM_R nel caso di ILE C e ILE RPG e AMQ0STUB_R nel caso di ILE COBOL. Per ulteriori informazioni sulla sicurezza del sottoprocesso delle applicazioni RPG o COBOL, fare riferimento al manuale Programmer's Guide appropriato per il linguaggio.
- IBM MQ for IBM i richiede che i programmi di uscita siano abilitati per il supporto teraspace. (Teraspace è una forma di memoria condivisa introdotta in OS/400 V4R4.) Per i compilatori ILE RPG e COBOL, tutti i programmi compilati su OS/400 V4R4 o versioni successive sono abilitati. Per C, i programmi devono essere compilati con le opzioni TERASPACE (*YES *TSIFC) specificate sui comandi CRTCMOD o CRTBNDC.
- Un'uscita che restituisce un puntatore al proprio spazio di buffer deve garantire che l'oggetto puntato esista oltre il periodo di tempo del programma di uscita del canale. Il puntatore non può essere l'indirizzo di una variabile nello stack di programmi né di una variabile nello heap di programmi. Invece, il puntatore deve essere ottenuto dal sistema. Un esempio è uno spazio utente creato nell'uscita utente. Per assicurarsi che qualsiasi area dati assegnata dal programma di uscita del canale sia ancora disponibile per l'MCA al termine del programma, l'uscita del canale deve essere eseguita nel gruppo di attivazione del chiamante o in un gruppo di attivazione denominato. Eseguire questa operazione impostando il parametro ACTGRP su CRTPGM su un valore definito dall'utente o *CALLER. Se il programma viene creato in questo modo, il programma di uscita del canale può allocare memoria dinamica e passare un puntatore a questa memoria all'MCA.

Concetti correlati

"Scrittura di programmi di uscita canale su AIX, Linux, and Windows" a pagina 971
È possibile utilizzare le seguenti informazioni per scrivere programmi di uscita del canale per sistemi AIX, Linux, and Windows .

"Scrittura dei programmi di uscita del canale su z/OS" a pagina 968

È possibile utilizzare le seguenti informazioni per scrivere e compilare programmi di uscita canale per z/OS.

ALW *Scrittura di programmi di uscita canale su AIX, Linux, and Windows*
È possibile utilizzare le seguenti informazioni per scrivere programmi di uscita del canale per sistemi AIX, Linux, and Windows .

Seguire le istruzioni descritte in “Scrittura di uscite e servizi installabili su AIX, Linux, and Windows” a pagina 938. Utilizzare le seguenti informazioni specifiche sull'uscita del canale, dove appropriato:

L'uscita deve essere scritta in C ed è una DLL su Windows.

Definire una routine MQStart () fittizia nell'uscita e specificare `MQStart` come punto di ingresso nella libreria. Figura 106 a pagina 971 mostra come impostare una voce per il programma:

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

Figura 106. Codice sorgente di esempio per un'uscita canale

Quando si scrivono uscite di canale per Windows utilizzando Visual C + +, è necessario scrivere il proprio file DEF . Un esempio di come viene mostrato in Figura 107 a pagina 971. Per ulteriori informazioni sulla scrittura dei programmi di uscita del canale, consultare “Scrittura di programmi di uscita canale” a pagina 966.

```
EXPORTS
ChannelExit
```

Figura 107. File DEF di esempio per Windows

Concetti correlati

“Scrittura dei programmi di uscita del canale su IBM i” a pagina 970
È possibile utilizzare le seguenti informazioni per scrivere e compilare programmi di uscita canale per IBM i.

“Scrittura dei programmi di uscita del canale su z/OS” a pagina 968
È possibile utilizzare le seguenti informazioni per scrivere e compilare programmi di uscita canale per z/OS.

Programmi di uscita di sicurezza del canale
È possibile utilizzare i programmi di uscita di sicurezza per verificare che il partner all'altra estremità di un canale sia autentico. Questa operazione è nota come autenticazione.

Per indicare che un canale deve utilizzare un'uscita di sicurezza, specificare il nome dell'uscita nel campo **SCYEXIT** della definizione del canale.

Nota: L'autenticazione può essere ottenuta anche con i record di autenticazione di canale. I record di autenticazione di canale forniscono una grande flessibilità nella prevenzione dell'accesso ai gestori code da parte di determinati utenti e canali e nell'associazione di utenti remoti agli identificatori utente IBM MQ . Il supporto TLS viene fornito anche da IBM MQ per autenticare i tuoi utenti e per fornire la crittografia e i controlli di integrità dei dati per i tuoi dati. Per ulteriori informazioni su TLS, vedi Protocolli di sicurezza TLS in IBM MQ. Tuttavia, se si richiedono ancora forme più sofisticate (o diverse) di

elaborazione della sicurezza e altri tipi di verifiche e di creazione di un contesto di sicurezza, considerare la scrittura di uscite di sicurezza.

Gli attributi Oggetto e DN emittente vengono visualizzati nei seguenti attributi di stato del canale:

- SSLPEER (selettore PCF MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (selettore PCF MQCACH_SSL_CERT_ISSUER_NAME)

Questi valori vengono restituiti dai comandi di stato del canale e dai dati passati alle uscite di sicurezza del canale elencati, come mostrato:

- Ptr MQCD SSLPeerName
- Ptr MQCXP SSLRemCertIssName

Un'uscita di sicurezza può essere scritta in C o Java.

I programmi di uscita di sicurezza del canale vengono richiamati nelle seguenti posizioni nel ciclo di elaborazione di un MCA:

- All'inizio e alla fine di MCA.
- Immediatamente dopo che la negoziazione dei dati iniziale è terminata all'avvio del canale. Il destinatario o l'estremità del server del canale può avviare uno scambio di messaggi di sicurezza con l'estremità remota fornendo un messaggio da consegnare all'uscita di sicurezza all'estremità remota. Potrebbe anche rifiutarsi di farlo. Il programma di uscita viene riavviato per elaborare qualsiasi messaggio di sicurezza ricevuto dall'estremità remota.
- Immediatamente dopo che la negoziazione dei dati iniziale è terminata all'avvio del canale. L'estremità mittente o richiedente del canale elabora un messaggio di sicurezza ricevuto dall'estremità remota oppure avvia uno scambio di sicurezza quando l'estremità remota non può. Il programma di uscita viene avviato di nuovo per elaborare tutti i messaggi di sicurezza successivi che potrebbero essere ricevuti.

Un canale richiedente non viene mai richiamato con MQXR_INIT_SEC. Il canale notifica al server di avere un programma di uscita di sicurezza e il server ha quindi la possibilità di avviare un'uscita di sicurezza. Se non ne ha uno, informa il richiedente e viene restituito un flusso di lunghezza zero al programma di uscita.

Nota: Evitare di inviare messaggi di sicurezza a lunghezza zero.

Esempi di dati scambiati dai programmi di uscita di sicurezza sono illustrati nelle figure da Figura 108 a pagina 973 a Figura 111 a pagina 975. Questi esempi mostrano la sequenza di eventi che si verificano che coinvolgono l'uscita di sicurezza del ricevente e l'uscita di sicurezza del mittente. Le righe successive nelle figure rappresentano il passare del tempo. In alcuni casi, gli eventi al destinatario e al mittente non sono correlati e quindi possono verificarsi contemporaneamente o in momenti diversi. In altri casi, un evento in un programma di uscita risulta in un evento complementare che si verifica successivamente nell'altro programma di uscita. Ad esempio, in Figura 108 a pagina 973:

1. Il destinatario e il mittente vengono richiamati ciascuno con MQXR_INIT, ma questi richiami non sono correlati e possono quindi verificarsi contemporaneamente o in momenti diversi.
2. Il destinatario viene successivamente richiamato con MQXR_INIT_SEC, ma restituisce MQXCC_OK che non richiede alcun evento complementare all'uscita del mittente.
3. Il mittente viene quindi richiamato con MQXR_INIT_SEC. Ciò non è correlato al richiamo del destinatario con MQXR_INIT_SEC. Il mittente restituisce MQXCC_SEND_SEC_MSG, che causa un evento complementare all'uscita del destinatario.
4. Il destinatario viene quindi richiamato con MQXR_SEC_MSG e restituisce MQXCC_SEND_SEC_MSG, che causa un evento complementare all'uscita del mittente.
5. Il mittente viene quindi richiamato con MQXR_SEC_MSG e restituisce MQXCC_OK che non richiede alcun evento complementare all'uscita del destinatario.

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Message transfer begins	

Figura 108. Scambio avviato dal mittente con accordo

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figura 109. Scambio avviato dal mittente senza accordo

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figura 110. Scambio avviato dal destinatario con accordo

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

Figura 111. Scambio avviato dal destinatario senza accordo

Al programma di uscita di sicurezza del canale viene passato un buffer dell'agent contenente i dati di sicurezza, escluse le intestazioni di trasmissione, generati dall'uscita di sicurezza. Questi dati possono essere dati adatti in modo che entrambe le estremità del canale siano in grado di eseguire la convalida di sicurezza.

Il programma di uscita di sicurezza, sia all'estremità di invio che a quella di ricezione del canale messaggi, può restituire uno dei due codici di risposta a qualsiasi chiamata:

- Scambio di sicurezza terminato senza errori
- Sopprimere il canale e chiudere

Nota:

1. Le uscite di sicurezza del canale generalmente funzionano a coppie. Quando si definiscono i canali appropriati, verificare che i programmi di uscita compatibili siano denominati per entrambe le estremità del canale.
2. IBM i In IBM i , i programmi di uscita di sicurezza che sono stati compilati con `Use adopted authority` (USEADPAUT = *YES) possono adottare l'autorizzazione QMQM o QMQMADM. Fare attenzione che l'uscita non utilizzi questa funzione per rappresentare un rischio per la sicurezza del sistema.
3. Su un canale TLS su cui l'altra estremità del canale fornisce un certificato, l'uscita di sicurezza riceve il DN (Distinguished Name) dell'oggetto di questo certificato nel campo MQCD a cui accede SSLPeerNamePtr e il DN (Distinguished Name) dell'emittente nel campo MQCXP a cui accede SSLRemCertIssNamePtr. Gli utilizzi in cui è possibile inserire questo nome sono:
   - Per limitare l'accesso sul canale TLS.
   - Per modificare MQCD.MCAUserIdentifier basato sul nome.

Concetti correlati

Concetti di TLS (Transport Layer Security)

Riferimenti correlati

Record di autenticazione di canale

Scrittura di un'uscita di sicurezza

È possibile scrivere un'uscita di sicurezza utilizzando il codice di base dell'uscita di sicurezza.

Figura 112 a pagina 976 illustra come scrivere un'uscita di sicurezza.

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

Figura 112. Codice di base uscita di sicurezza

MQStart del punto di ingresso IBM MQ standard deve esistere, ma non è richiesto per eseguire alcuna funzione. Il nome della funzione (EntryPoint in questo esempio) può essere modificato, ma la funzione deve essere esportata quando la libreria viene compilata e collegata. Come nell'esempio precedente, i puntatori pChannelExitParms devono essere passati a PMQCXP e la definizione pChannela PMQCD. Per informazioni generali sulla chiamata delle uscite del canale e sull'utilizzo dei parametri, vedi MQ_CHANNEL_EXIT. Questi parametri vengono utilizzati in un'uscita di sicurezza come segue:

PMQVOID pChannelExitParms
I/O

Puntatore alla struttura MQCXP - cast a PMQCXP per accedere ai campi. Questa struttura viene utilizzata per comunicare tra Exit e MCA. I seguenti campi in MQCXP sono particolarmente interessanti per le uscite di sicurezza:

ExitReason
Indica all'uscita di sicurezza lo stato corrente nello scambio di sicurezza e viene utilizzato quando si decide quale azione intraprendere.

ExitResponse
La risposta all'MCA che determina la fase successiva nello scambio di sicurezza.

ExitResponse2
Indicatori di controllo supplementari per gestire il modo in cui l'MCA interpreta la risposta dell'uscita di sicurezza.

Area ExitUser
16 byte (massimo) di memoria che possono essere utilizzati dall'uscita di sicurezza per mantenere lo stato tra le chiamate.

ExitData
Contiene i dati specificati nel campo SCYDATA della definizione del canale (32 byte riempiti a destra con spazi).

Definizione pChannelPMQVOID
I/O

Puntatore alla struttura MQCD - cast a PMQCD per accedere ai campi. Questo parametro contiene la definizione del canale. I seguenti campi in MQCD sono di particolare interesse per le uscite di sicurezza:

ChannelName
Il nome del canale (20 byte riempiti a destra con spazi).

ChannelType
Un codice che definisce il tipo di canale.

Identificativo utente MCA
Questo gruppo di tre campi viene inizializzato sul valore del campo MCAUSER specificato nella definizione del canale. Qualsiasi identificativo utente specificato dall'uscita di sicurezza in questi campi viene utilizzato per il controllo accessi (non applicabile ai canali SDR, SVR, CLNTCONN o CLUSSDR).

MCAUserIdentifier
I primi 12 byte dell'identificativo sono stati riempiti a destra con spazi.

LongMCAUserIdPtr
Il puntatore a un buffer che contiene l'identificativo di lunghezza completa (con terminazione null non garantita) ha la priorità su MCAUserIdentifier.

LongMCAUserIdLength
Lunghezza della stringa indicata da LongMCAUserIdPtr - deve essere impostata se è impostato LongMCAUserIdPtr .

Identificativo utente remoto
Si applica solo alle coppie di canali CLNTCONN/SVRCONN. Se non è definita alcuna uscita di sicurezza CLNTCONN, questi tre campi vengono inizializzati dall'MCA client, quindi potrebbero contenere un identificativo utente dall'ambiente del client che può essere utilizzato da un'uscita di sicurezza SVRCONN per l'autenticazione e quando si specifica l'identificativo utente MCA. Se è definita un'uscita di sicurezza CLNTCONN, questi campi non vengono inizializzati e possono essere impostati dall'uscita di protezione CLNTCONN oppure è possibile utilizzare i messaggi di sicurezza per passare un identificativo utente dal client al server.

Identificativo RemoteUser
I primi 12 byte dell'identificativo sono stati riempiti a destra con spazi.

LongRemoteUserIdPtr
Il puntatore a un buffer contenente l'identificativo di lunghezza completa (con terminazione null non garantita) ha la priorità sull'identificativo RemoteUser.

LongRemoteUserIdLunghezza
Lunghezza della stringa indicata da LongRemoteUserIdPtr - deve essere impostata se è impostato LongRemoteUserIdPtr.

Lunghezza pDataPMQLONG
I/O

Puntatore a MQLONG. Contiene la lunghezza di qualsiasi uscita di sicurezza contenuta in AgentBuffer al richiamo dell'uscita di sicurezza. Deve essere impostato da un'uscita di sicurezza sulla lunghezza di qualsiasi messaggio inviato in AgentBuffer o ExitBuffer.

PMQLONG pAgentBufferLength
input

Puntatore a MQLONG. La lunghezza dei dati contenuti nel AgentBuffer al richiamo dell'uscita di sicurezza.

Buffer pAgentPMQVOID
I/O

Al richiamo dell'uscita di sicurezza, questo fa riferimento a qualsiasi messaggio inviato dall'uscita partner. Se ExitResponse2 nella struttura di MQCXP ha l'indicatore MQXR2_USE_AGENT_BUFFER impostato (predefinito), un'uscita di sicurezza deve impostare questo parametro per puntare a tutti i dati del messaggio inviati.

PMQLONG pExitBufferLength
I/O

Puntatore a MQLONG. Questo parametro viene inizializzato a 0 al primo richiamo di un'uscita di sicurezza e il valore restituito viene mantenuto tra le chiamate all'uscita di sicurezza durante uno scambio di sicurezza.

PMQPTR pExitBufferAddr
I/O

Questo parametro viene inizializzato su un puntatore null al primo richiamo di un'uscita di sicurezza e il valore restituito viene mantenuto tra le chiamate all'uscita di sicurezza durante uno scambio di sicurezza. Se l'indicatore MQXR2_USE_EXIT_BUFFER è impostato in ExitResponse2 nella struttura MQCXP, un'uscita di sicurezza deve impostare questo parametro per puntare a qualsiasi dato del messaggio inviato.

Differenze nel comportamento tra le uscite di sicurezza definite sulle coppie di canali CLNTCONN/SVRCONN e altre coppie di canali
Le uscite di sicurezza possono essere definite su tutti i tipi di canale. Tuttavia, il comportamento delle uscite di sicurezza definite sulle coppie di canali CLNTCONN/SVRCONN è leggermente differente dalle uscite di sicurezza definite su altre coppie di canali.

Un'uscita di sicurezza su un canale CLNTCONN può impostare l'identificativo utente remoto nella definizione del canale per l'elaborazione da parte di un'uscita SVRCONN partner o per l'autorizzazione OAM se non è definita alcuna uscita di sicurezza SVRCONN e il campo MCAUSER di SVRCONN non è impostato.

Se non viene definita alcuna uscita di sicurezza CLNTCONN, l'identificativo utente remoto nella definizione del canale viene impostato su un identificativo utente dall'ambiente client (che può essere vuoto) dall'MCA del client.

Uno scambio di sicurezza tra le uscite di sicurezza definite su una coppia di canali CLNTCONN e SVRCONN viene completato correttamente quando l'uscita di sicurezza SVRCONN restituisce una ExitResponse di MQXCC_OK. Uno scambio di sicurezza tra altre coppie di canali viene completato correttamente quando l'uscita di protezione che ha avviato lo scambio restituisce una ExitResponse di MQXCC_OK.

Tuttavia, il codice MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse può essere utilizzato per forzare il proseguimento dello scambio di sicurezza: se una ExitResponse di MQXCC_SEND_AND_REQUEST_SEC_MSG viene restituita da un'uscita di sicurezza CLNTCONN o SVRCONN, l'uscita partner deve rispondere inviando un messaggio di sicurezza (non MQXCC_OK o una risposta null) o il canale viene terminato. Per le uscite di sicurezza definite su altri tipi di canale, una ExitResponse di MQXCC_OK restituita in risposta a MQXCC_SEND_AND_REQUEST_SEC_MSG dall'uscita di sicurezza del partner risulta nella continuazione dello scambio di sicurezza come se fosse stata restituita una risposta null e non in chiusura del canale.

Uscita di sicurezza SSPI
IBM MQ for Windows fornisce un'uscita di sicurezza che fornisce l'autenticazione per canali IBM MQ utilizzando SSPI (Security Services Programming Interface). SSPI fornisce le funzionalità di sicurezza integrate di Windows.

Questa uscita di sicurezza è sia per il client IBM MQ che per il server IBM MQ .

I pacchetti di sicurezza vengono caricati da security.dll o secur32.dll. Queste DLL vengono fornite con il sistema operativo.

L'autenticazione unidirezionale viene fornita su Windows, utilizzando i servizi di autenticazione NTLM. L'autenticazione bidirezionale viene fornita su Windows 2000, utilizzando servizi di autenticazione Kerberos .

Il programma di uscita di sicurezza viene fornito in formato origine e oggetto. È possibile utilizzare il codice oggetto così com' è oppure è possibile utilizzare il codice origine come punto di partenza per creare i propri programmi di uscita utente. Per ulteriori informazioni sull'utilizzo dell'oggetto o del codice sorgente dell'uscita di sicurezza SSPI, consultare “Utilizzo dell'uscita di sicurezza SSPI su Windows” a pagina 1137

Programmi di uscita di invio e ricezione del canale
È possibile utilizzare le uscite di invio e ricezione per eseguire attività quali la compressione e la decompressione dei dati. È possibile specificare un elenco di programmi di uscita di invio e ricezione da eseguire in successione.

I programmi di uscita di invio e ricezione del canale vengono richiamate nelle seguenti posizioni nel ciclo di elaborazione di un MCA:

- I programmi di uscita di invio e ricezione vengono richiamati per l'inizializzazione all'avvio MCA e per la terminazione alla terminazione MCA.
- Il programma di uscita di invio viene richiamato all'una o all'altra estremità del canale, a seconda della fine in cui viene inviata una trasmissione per un trasferimento di messaggi, immediatamente prima che venga inviata una trasmissione sul collegamento. La nota 4 spiega perché le uscite sono disponibili in entrambe le direzioni anche se i canali di messaggi inviano messaggi in una sola direzione.
- Il programma di uscita di ricezione viene richiamato all'una o all'altra estremità del canale, a seconda della fine in cui viene ricevuta una trasmissione per un trasferimento di messaggi, immediatamente dopo che una trasmissione è stata presa dal collegamento. La nota 4 spiega perché le uscite sono disponibili in entrambe le direzioni anche se i canali di messaggi inviano messaggi in una sola direzione.

È possibile che vi siano molte trasmissioni per un trasferimento di messaggi e che vi siano molte iterazioni dei programmi di uscita di invio e ricezione prima che un messaggio raggiunga l'uscita di messaggio all'estremità di ricezione.

Ai programmi di uscita di invio e ricezione del canale viene passato un buffer dell'agent che contiene i dati di trasmissione come inviati o ricevuti dal collegamento delle comunicazioni. Per i programmi di uscita di invio, i primi 8 byte del buffer sono riservati all'utilizzo da parte di MCA e non devono essere modificati. Se il programma restituisce un buffer differente, questi primi 8 byte devono esistere nel nuovo buffer. Il formato dei dati presentati ai programmi di uscita non è definito.

Un buon codice di risposta deve essere restituito dai programmi di uscita di invio e ricezione. Qualsiasi altra risposta causa una fine anomala MCA (fine anomala).

Nota: Non emettere una chiamata MQGET, MQPUT o MQPUT1 all'interno di un punto di sincronizzazione da un'uscita di invio o ricezione.

Nota:

1. Le uscite di invio e ricezione generalmente funzionano in coppie. Ad esempio, un'uscita di invio potrebbe comprimere i dati e un'uscita di ricezione potrebbe decomprimerli oppure un'uscita di invio potrebbe codificare i dati e un'uscita di ricezione potrebbe decodificarli. Quando si definiscono i canali appropriati, verificare che i programmi di uscita compatibili siano denominati per entrambe le estremità del canale.
2. Se la compressione è attivata per il canale, alle uscite vengono trasmessi dati compressi.
3. Le uscite di invio e ricezione del canale potrebbero essere richiamate per i segmenti di messaggio diversi dai dati dell'applicazione, ad esempio i messaggi di stato. Non vengono richiamati durante la finestra di avvio, né durante la fase di verifica della sicurezza.
4. Anche se i canali dei messaggi inviano i messaggi solo in una direzione, i dati di controllo del canale, come i battiti cardiaci e la fine dell'elaborazione batch, fluiscono in entrambe le direzioni e queste uscite sono disponibili anche in entrambe le direzioni. Tuttavia, alcuni dei flussi di dati di avvio del canale iniziale sono esenti dall'elaborazione da parte di una qualsiasi delle uscite.
5. Ci sono circostanze in cui le uscite di invio e ricezione possono essere richiamate fuori sequenza; ad esempio, se si sta eseguendo una serie di programmi di uscita o se si stanno eseguendo anche uscite di sicurezza. Quindi, quando l'uscita di ricezione viene richiamata per la prima volta per elaborare i dati, potrebbe ricevere i dati che non sono passati attraverso l'uscita di invio corrispondente. Se l'uscita di ricezione ha appena eseguito l'operazione, ad esempio la decompressione, senza prima verificare che sia stata richiesta, i risultati sarebbero imprevisti.

   È necessario codificare le uscite di invio e ricezione in modo che l'uscita di ricezione possa controllare che i dati ricevuti siano stati elaborati dalla corrispondente uscita di invio. Il modo consigliato per farlo è quello di codificare i programmi di uscita in modo che:

   - L'uscita di invio imposta il valore del nono byte di dati su 0 e sposta tutti i dati su 1 byte, prima di eseguire l'operazione. (I primi 8 byte sono riservati per l'utilizzo da parte di MCA.)
   - Se l'uscita di ricezione riceve dati che hanno uno 0 in byte 9, sa che i dati provengono dall'uscita di invio. Rimuove lo 0, esegue l'operazione complementare e sposta i dati risultanti indietro di 1 byte.
   - Se l'uscita di ricezione riceve dati che hanno un valore diverso da 0 nel byte 9, assume che l'uscita di invio non sia stata eseguita e restituisce i dati al chiamante senza modificarli.

   Quando si utilizzano le uscite di sicurezza, se il canale viene terminato dall'uscita di sicurezza, è possibile che un'uscita di invio venga richiamata senza la corrispondente uscita di ricezione. Un modo per evitare questo problema consiste nel codificare l'uscita di sicurezza per impostare un indicatore, in MQCD.SecurityUserData o MQCD.SendUserData, ad esempio, quando l'uscita decide di terminare il canale. Quindi, l'uscita di invio deve controllare questo campo ed elaborare i dati solo se l'indicatore non è impostato. Questo controllo impedisce all'uscita di invio di modificare inutilmente i dati e quindi evita eventuali errori di conversione che potrebbero verificarsi se l'uscita di sicurezza ha ricevuto dati modificati.

Programmi di uscita di invio del canale - prenotazione di spazio
È possibile utilizzare uscite di invio e ricezione per trasformare i dati prima della trasmissione. I programmi di uscita invio canale possono aggiungere i propri dati sulla trasformazione riservando spazio nel buffer di trasmissione.

Questi dati vengono elaborati dal programma di uscita di ricezione e quindi rimossi dal buffer. Ad esempio, potresti voler crittografare i dati e aggiungere una chiave di sicurezza per la decrittografia.

Come riservare spazio e utilizzarlo

Quando il programma di uscita di invio viene richiamato per l'inizializzazione, impostare il campo *ExitSpace* di MQXCP sul numero di byte da riservare. Consultare MQCXP per i dettagli. *ExitSpace* può essere impostato solo durante l'inizializzazione, ovvero quando *ExitReason* ha valore MQXR_INIT.

Quando l'uscita di invio viene richiamata immediatamente prima della trasmissione, con *ExitReason* impostato su MQXR_XMIT, *ExitSpace* byte sono riservati nel buffer di trasmissione. *ExitSpace* non è supportato su z/OS.

L'uscita di invio non deve necessariamente utilizzare tutto lo spazio riservato. Può utilizzare meno di *ExitSpace* byte o, se il buffer di trasmissione non è pieno, l'uscita può utilizzare più della quantità riservata. Quando si imposta il valore *ExitSpace*, è necessario lasciare almeno 1 KB per i dati del messaggio nel buffer di trasmissione. Le prestazioni del canale possono essere influenzate se viene utilizzato spazio riservato per grandi quantità di dati.

Il buffer di trasmissione è generalmente lungo 32KB . Tuttavia, se il canale utilizza TLS, la dimensione del buffer di trasmissione viene ridotta a 15.352 byte in modo da adattarsi alla lunghezza massima del record definita da RFC 6101 e alla relativa famiglia di standard TLS. Ulteriori 1024 byte sono riservati per l'utilizzo da parte di IBM MQ, quindi lo spazio di buffer di trasmissione massimo utilizzabile dalle uscite di invio è di 14.328 byte.

Cosa succede all'estremità ricevente del canale

I programmi di uscita di ricezione del canale devono essere impostati per essere compatibili con le corrispondenti uscite di invio. Le uscite di ricezione devono conoscere il numero di byte nello spazio riservato e devono rimuovere i dati in tale spazio.

Uscite di invio multiple

È possibile specificare un elenco di programmi di uscita di invio e ricezione da eseguire in successione. IBM MQ conserva uno spazio totale riservato da tutte le uscite di invio. Questo spazio totale deve lasciare almeno 1 KB per i dati del messaggio nel buffer di trasmissione.

Il seguente esempio mostra come viene assegnato lo spazio per tre uscite di invio, chiamate in successione:

1. Quando viene richiamata l'inizializzazione:
   - L'uscita di invio A riserva 1 KB.
   - L'uscita di invio B riserva 2 KB.
   - Uscita di invio C riserva 3 KB.
2. La dimensione massima di trasmissione è 32 KB e i dati utente sono lunghi 5 KB.
3. L'uscita A viene richiamata con 5 KB di dati; sono disponibili fino a 27 KB, poiché 5 KB sono riservati alle uscite B e C. L'uscita A aggiunge 1 KB, la quantità riservata.
4. L'uscita B viene richiamata con 6 KB di dati; sono disponibili fino a 29 KB, poiché 3 KB sono riservati all'uscita C. L'uscita B aggiunge 1 KB, meno dei 2 KB riservati.
5. L'uscita C viene richiamata con 7 KB di dati; sono disponibili fino a 32 KB. L'uscita C aggiunge 10K, più dei 3 KB che ha riservato. Questa quantità è valida, poiché la quantità totale di dati, 17 KB, è inferiore al massimo di 32 KB.

La dimensione massima del buffer di trasmissione per un canale che utilizza TLS è 15.352 byte, non 32KB. Questo perché i segmenti di trasmissione socket sicuri sottostanti sono limitati a 16KB e parte dello spazio è richiesto per i sovraccarichi del record TLS. Ulteriori 1024 byte sono riservati per l'utilizzo da parte di IBM MQ, quindi lo spazio di buffer di trasmissione massimo utilizzabile dalle uscite di invio è di 14.328 byte.

Programmi di uscita messaggi canale
È possibile utilizzare l'uscita del messaggio del canale per eseguire attività quali la crittografia sul collegamento, la convalida o la sostituzione di ID utente in entrata, la conversione dei dati del messaggio, la registrazione su giornale e la gestione dei messaggi di riferimento. È possibile specificare un elenco di programmi di uscita messaggi da eseguire in successione.

I programmi di uscita dei messaggi del canale vengono richiamati nelle seguenti posizioni nel ciclo di elaborazione dell'MCA:

- All'inizio e alla fine di MCA
- Immediatamente dopo che un MCA di invio ha emesso una chiamata MQGET
- Prima che l'MCA ricevente emani una chiamata MQPUT

All'uscita del messaggio viene passato un buffer dell'agent contenente l'intestazione della coda di trasmissione MQXQH e il testo del messaggio dell'applicazione richiamato dalla coda. Il formato di MQXQH viene fornito in MQXQH - Transmission - queue header.

Se si utilizzano messaggi di riferimento (ovvero, messaggi che contengono solo un'intestazione che punta a qualche altro oggetto che deve essere inviato), l'uscita del messaggio riconosce l'intestazione, MQRMH. Identifica l'oggetto, lo richiama in qualsiasi modo appropriato lo aggiunge all'intestazione e lo passa all'MCA per la trasmissione all'MCA ricevente. All'MCA di ricezione, un'altra uscita del messaggio riconosce che questo messaggio è un messaggio di riferimento, estrae l'oggetto e trasmette l'intestazione alla coda di destinazione. Consultare “Messaggi di riferimento” a pagina 796 e “Esecuzione degli esempi del messaggio di riferimento” a pagina 1110 per ulteriori informazioni sui messaggi di riferimento e su alcune uscite di messaggi di esempio che li gestiscono.

Le uscite messaggio possono restituire le seguenti risposte:

- Inviare il messaggio (GET exit). Il messaggio potrebbe essere stato modificato dall'uscita. (Questo restituisce MQXCC_OK.)
- Inserire il messaggio nella coda (uscita PUT). Il messaggio potrebbe essere stato modificato dall'uscita. (Questo restituisce MQXCC_OK.)
- Non elaborare il messaggio. Il messaggio viene inserito nella coda di messaggi non recapitabili (coda di messaggi non recapitati) dall'MCA.
- Chiudere il canale.
- Codice di ritorno errato, che causa la fine anomala dell'MCA.

Nota:

1. Le uscite messaggio vengono richiamate una volta per ogni messaggio completo trasferito, anche quando il messaggio è suddiviso in parti.
2. Linux AIX Se si fornisce un'uscita del messaggio su AIX o Linux, la conversione automatica degli ID utente in caratteri minuscoli (descritta qui) non funziona.
3. Un'uscita viene eseguita nello stesso thread dell'MCA stesso. Viene eseguito anche all'interno della stessa unità di lavoro (UOW) dell'MCA poiché utilizza lo stesso handle di connessione. Quindi, tutte le chiamate effettuate nel punto di sincronizzazione vengono sottoposte a commit o a backout dal canale alla fine del batch. Ad esempio, un programma di uscita del messaggio del canale può inviare messaggi di notifica a un altro e questi messaggi vengono sottoposti a commit solo sulla coda quando viene eseguito il commit del batch contenente il messaggio originale.

   Pertanto, è possibile emettere chiamate MQI del punto di sincronizzazione da un programma di uscita messaggi del canale.

Conversione del messaggio al di fuori dell'uscita del messaggio
Prima di richiamare l'uscita messaggio, l'MCA ricevente esegue alcune conversioni sul messaggio. Questo argomento descrive gli algoritmi utilizzati per eseguire la conversione.

Quali intestazioni vengono elaborate

Una routine di conversione viene eseguita nell'MCA del destinatario prima che venga richiamata l'uscita del messaggio. La routine di conversione inizia con l'intestazione MQXQH all'inizio del messaggio. La routine di conversione viene quindi eseguita tramite le intestazioni concatenate che seguono MQXQH, eseguendo la conversione dove necessario. Le intestazioni concatenate possono estendersi oltre l'offset contenuto nel parametro HeaderLength dei dati MQCXP passati all'uscita del messaggio del destinatario. Le seguenti intestazioni vengono convertite in loco:

- MQXQH (nome formato " MQXMIT ")
- MQMD (questa intestazione fa parte di MQXQH e non ha un nome formato)
- MQMDE (nome formato " MQHMDE ")
- MQDH (nome formato " MQHDIST ")
- MQWIH (nome formato " MQHWIH ")

Le seguenti intestazioni non vengono convertite, ma vengono superate mentre l'MCA continua a elaborare le intestazioni concatenate:

- MQDLH (nome formato " MQDEAD ")
- tutte le intestazioni con nomi formato che iniziano con i tre caratteri 'MQH' (ad esempio " MQHRF ") che non sono altrimenti menzionati

Modalità di elaborazione delle intestazioni

Il parametro Format di ogni intestazione IBM MQ viene letto da MCA. Il parametro Formato è di 8 byte all'interno dell'intestazione, che sono 8 caratteri a byte singolo contenenti un nome.

L'MCA interpreta quindi i dati che seguono ciascuna intestazione come se fossero del tipo denominato. Se il formato è il nome di un tipo di intestazione idoneo per la conversione dati IBM MQ , viene convertito. Se è un altro nome che indica dati nonMQ (ad esempio MQFMT_NONE o MQFMT_STRING), l'MCA arresta l'elaborazione delle intestazioni.

Cos' è MQCXP HeaderLength?

Il parametro HeaderLength nei dati MQCXP forniti a un'uscita messaggio è la lunghezza totale delle intestazioni MQXQH (che include MQMD), MQMDE e MQDH all'inizio del messaggio. Queste intestazioni sono concatenate utilizzando i nomi e le lunghezze 'Formato'.

MQWIH

Le intestazioni concatenate possono estendersi oltre la HeaderLength nell'area dati utente. L'intestazione MQWIH, se presente, è una di quelle intestazioni che vengono visualizzate oltre HeaderLength.

Se è presente un'intestazione MQWIH nelle intestazioni concatenate, viene convertita in posizione prima che venga richiamata l'uscita del messaggio del destinatario.

Programma di uscita nuovo tentativo messaggio canale
L'uscita di nuovo tentativo del messaggio del canale viene richiamata quando un tentativo di aprire la coda di destinazione non riesce. È possibile utilizzare l'uscita per determinare in quali circostanze ritentare, quante volte ritentare e con quale frequenza.

Questa uscita viene richiamata anche all'estremità ricevente del canale all'avvio e alla chiusura MCA.

L'uscita del nuovo tentativo del messaggio del canale viene passata a un buffer dell'agent contenente l'intestazione della coda di trasmissione, MQXQH, e il testo del messaggio dell'applicazione come richiamato dalla coda. Il formato di MQXQH viene fornito in Panoramica per MQXQH.

L'uscita viene richiamata per tutti i codici di errore; l'uscita determina per quali codici di errore desidera che l'MCA esegua un nuovo tentativo, per quante volte e a quali intervalli. (Il valore del conteggio dei tentativi dei messaggi impostato quando il canale è stato definito viene passato all'exit in MQCD, ma l'exit può ignorare questo valore.)

Il campo MsgRetryCount in MQCXP viene incrementato dall'MCA ogni volta che viene richiamata l'uscita e l'uscita restituisce MQXCC_OK con il tempo di attesa contenuto nel campo Intervallo MsgRetrydi MQCXP o MQXCC_SUPPRESS_FUNCTION. I tentativi continuano indefinitamente fino a quando l'uscita non restituisce MQXCC_SUPPRESS_FUNCTION nel campo ExitResponse di MQCXP. Consultare MQCXP per informazioni sull'azione eseguita da MCA per questi codici di completamento.

Se tutti i tentativi hanno esito negativo, il messaggio viene scritto nella coda di messaggi non recapitabili. Se non è disponibile una coda di messaggi non instradabili, il canale si arresta.

Se non si definisce un'uscita di nuovo tentativo del messaggio per un canale e si verifica un errore che è probabile sia temporaneo, ad esempio MQRC_Q_FULL, l'MCA utilizza il conteggio di nuovi tentativi del messaggio e gli intervalli di nuovi tentativi del messaggio impostati quando il canale è stato definito. Se l'errore è di natura più permanente e non è stato definito un programma di uscita per gestirlo, il messaggio viene scritto nella coda di messaggi non recapitabili.

Programma di uscita di definizione automatica del canale
L'uscita di definizione automatica del canale può essere utilizzata quando viene ricevuta una richiesta per avviare un canale ricevente o di connessione server, ma non esiste alcuna definizione per tale canale (non per IBM MQ for z/OS). Può anche essere richiamato su tutte le piattaforme per i canali mittente e ricevente del cluster per consentire la modifica della definizione per un'istanza del canale.

L'uscita di definizione automatica del canale può essere richiamata su tutte le piattaforme tranne z/OS quando viene ricevuta una richiesta di avviare un canale ricevente o di connessione server, ma non esiste alcuna definizione di canale. È possibile utilizzarla per modificare la definizione predefinita fornita per un canale ricevente o di connessione server definito automaticamente, SYSTEM.AUTO.RECEIVERo SYSTEM.AUTO.SVRCON. Consultare Preparazione dei canali per una descrizione del modo in cui le definizioni dei canali possono essere create automaticamente.

L'uscita di definizione automatica del canale può essere richiamata anche quando viene ricevuta una richiesta di avvio di un canale mittente del cluster. Può essere richiamato per i canali mittente cluster e ricevente cluster per consentire la modifica della definizione per questa istanza del canale. In questo caso, l'uscita si applica anche a IBM MQ for z/OS. Un uso comune dell'uscita di definizione automatica del canale consiste nel modificare i nomi delle uscite dei messaggi (MSGEXIT, RCVEXIT, SCYEXIT e SENDEXIT) perché i nomi delle uscite hanno formati differenti su piattaforme differenti. Se non viene specificata alcuna uscita di definizione automatica del canale, il comportamento predefinito su z/OS consiste nell'esaminare un nome di uscita distribuita nel formato *[path]/libraryname(function)* e utilizzare fino a otto caratteri di funzione, se presenti, o nome libreria. Su z/OS, un programma di uscita di definizione automatica del canale deve modificare i campi indicati da MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr e ReceiveUserDataPtr, invece di MsgExit, MsgUserData, SendExit, SendUserData, ReceiveExit e campi di dati ReceiveUser.

Per ulteriori informazioni, consultare Utilizzo dei canali definiti automaticamente.

Come per altre uscite di canale, l'elenco dei parametri è:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

`ChannelExitParms` sono descritti in MQCXP. `ChannelDefinition` è descritto in MQCD.

MQCD contiene i valori utilizzati nella definizione di canale predefinita se non vengono modificati dall'exit. L'uscita può modificare solo un sottoinsieme di campi; consultare MQ_CHANNEL_AUTO_DEF_EXIT. Tuttavia, il tentativo di modificare altri campi non causa un errore.

L'uscita di definizione automatica del canale restituisce una risposta MQXCC_OK o MQXCC_SUPPRESS_FUNCTION. Se nessuna di queste risposte viene restituita, l'MCA continua l'elaborazione come se fosse stata restituita MQXCC_SUPPRESS_FUNCTION. In altre parole, la definizione automatica viene abbandonata, non viene creata alcuna nuova definizione di canale e il canale non può essere avviato.

ALW *Compilazione dei programmi di uscita del canale su sistemi AIX, Linux, and Windows*

Utilizzare i seguenti esempi per compilare i programmi di uscita del canale per i sistemi AIX, Linux, and Windows .

Windows

Windows

Il comando compilatore e linker per i programmi channel - exit su Windows:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

Sistemi AIX and Linux

Linux AIX

In questi esempi, `exit` è il nome della libreria e `ChannelExit` è il nome della funzione. Su AIX il file di esportazione viene denominato `exit.exp`. Questi nomi vengono utilizzati dalla definizione di canale per fare riferimento al programma di uscita utilizzando il formato descritto in Definizione canale MQCD. Consultare anche il parametro MSGEXIT del comando DEFINE CHANNEL .

AIX Comandi del compilatore di esempio e del linker per le uscite del canale su AIX:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Esempi di comandi del compilatore e del linker per channel - exit su Linux dove il gestore code è a 32 bit:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Esempi di comandi del compilatore e del linker per le uscite del canale su Linux dove il gestore code è a 64 bit:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Sul client, è possibile utilizzare un'uscita a 32 bit o a 64 bit. Questa uscita deve essere collegata a mqic_r.

AIX in AIX, tutte le funzioni richiamate da IBM MQ devono essere esportate. Un file di esportazione di esempio per questo file make:

```
#
!channelExit
MQStart
```

Configurazione delle uscite canale

Per richiamare l'uscita del canale, è necessario denominarlo nella definizione del canale.

Le uscite canale devono essere denominate nella definizione del canale. È possibile eseguire questa denominazione quando si definiscono per la prima volta i canali oppure è possibile aggiungere le informazioni in un secondo momento utilizzando, ad esempio, il comando MQSC ALTER CHANNEL. È anche possibile fornire i nomi di uscita del canale nella struttura dati del canale MQCD. Il formato del nome di uscita dipende dalla piattaforma IBM MQ ; per informazioni, consultare Comandi MQCD o MQSC .

Se la definizione del canale non contiene un nome di programma di uscita utente, l'uscita utente non viene richiamata.

L'uscita di definizione automatica del canale è la proprietà del gestore code, non il singolo canale. Affinché questa uscita possa essere richiamata, deve essere denominata nella definizione del gestore code. Per modificare una definizione del gestore code, utilizzare il comando MQSC ALTER QMGR.

Scrittura delle uscite di conversione dati

Questa raccolta di argomenti contiene informazioni su come scrivere le uscite di conversione dati.

Nota: Non supportato in MQSeries per VSE/ESA.

Quando si esegue un MQPUT, l'applicazione crea il descrittore del messaggio (MQMD) del messaggio. Poiché IBM MQ deve essere in grado di comprendere il contenuto di MQMD indipendentemente dalla piattaforma su cui viene creato, viene convertito automaticamente dal sistema.

I dati dell'applicazione, tuttavia, non vengono convertiti automaticamente. Se i dati carattere vengono scambiati tra le piattaforme in cui i campi `CodedCharSetId` e `Encoding` differiscono, ad esempio, tra ASCII e EBCDIC, l'applicazione deve organizzare la conversione del messaggio. La conversione dei dati dell'applicazione può essere eseguita dal gestore code stesso o da un programma di uscita utente, denominato *uscita di conversione dati*. Il gestore code può eseguire la conversione dei dati autonomamente, utilizzando una delle routine di conversione integrate, se i dati dell'applicazione si trovano in uno dei formati integrati (come MQFMT_STRING). Questo argomento contiene informazioni sulla funzione di uscita di conversione dati fornita da IBM MQ quando i dati dell'applicazione non sono in formato integrato.

Il controllo può essere passato all'uscita di conversione dati durante una chiamata MQGET. Ciò evita la conversione su diverse piattaforme prima di raggiungere la destinazione finale. Tuttavia, se la destinazione finale è una piattaforma che non supporta la conversione dei dati su MQGET, è necessario specificare CONVERT (YES) sul canale mittente che invia i dati alla destinazione finale. Ciò garantisce che IBM MQ converta i dati durante la trasmissione. In questo caso, l'uscita di conversione dati deve risiedere sul sistema in cui è definito il canale mittente.

La chiamata MQGET viene emessa direttamente dall'applicazione. Impostare i campi `CodedCharSetId` e `Encoding` in MQMD sulla serie di caratteri e sulla codifica richieste. Se l'applicazione utilizza la stessa serie di caratteri e la stessa codifica del gestore code, impostare `CodedCharSetId` su MQCCSI_Q_MGR e `Encoding` su MQENC_NATIVE. Una volta completata la chiamata MQGET, questi campi hanno i valori appropriati per i dati del messaggio restituiti. Potrebbero essere diversi dai valori richiesti se la conversione non ha avuto esito positivo. L'applicazione deve reimpostare questi campi sui valori richiesti prima di ogni chiamata MQGET.

Le condizioni richieste per l'uscita di conversione dati da chiamare sono definite per la chiamata MQGET in MQGET.

Per una descrizione dei parametri passati all'uscita di conversione dati e per le note di utilizzo dettagliate, consultare Conversione dati per la chiamata MQ_DATA_CONV_EXIT e la struttura MQDXP.

I programmi che convertono i dati dell'applicazione tra diverse codifiche macchina e CCSID devono essere conformi alla DCI (Data Conversion Interface) IBM MQ .

Per i client Multicast, le uscite API e le uscite di conversione dati devono essere in grado di essere eseguite sul lato client perché alcuni messaggi potrebbero non passare attraverso il gestore code. Le seguenti librerie fanno parte dei pacchetti client e dei pacchetti server:

Tabella 143. Librerie che si trovano nei package client e server

Sistema operativo	**Librerie**
AIX	32 bit & 64 bit: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 bit & 64 bit: libmqm.so & libmqm_r.so
Windows	32 bit & 64 bit: mqm.dll & mqm.pdb

Richiamo dell'uscita di conversione dati

Un'uscita di conversione dati è un'uscita scritta dall'utente che riceve il controllo durante l'elaborazione di una chiamata MQGET.

L'uscita viene richiamata se le seguenti istruzioni sono vere:

- L'opzione MQGMO_CONVERT è specificata nella chiamata MQGET.
- Alcuni o tutti i dati del messaggio non si trovano nella serie di caratteri o nella codifica richiesta.
- Il campo *Format* nella struttura MQMD associata al messaggio non è MQFMT_NONE.
- Il valore *BufferLength* specificato nella chiamata MQGET non è zero.
- La lunghezza dei dati del messaggio è diversa da zero.
- Il messaggio contiene dati con un formato definito dall'utente. Il formato definito dall'utente può occupare l'intero messaggio o essere preceduto da uno o più formati incorporati. Ad esempio, il formato definito dall'utente potrebbe essere preceduto da un formato MQFMT_DEAD_LETTER_HEADER. L'uscita viene richiamata per convertire solo il formato definito dall'utente; il gestore code converte tutti i formati integrati che precedono il formato definito dall'utente.

  Un'uscita scritta dall'utente può essere richiamata anche per convertire un formato integrato, ma ciò si verifica solo se le routine di conversione integrate non possono convertire il formato incorporato con esito positivo.

Ci sono altre condizioni, descritte completamente nelle note di utilizzo della chiamata MQ_DATA_CONV_EXIT in MQ_DATA_CONV_EXIT.

Consultare MQGET per i dettagli della chiamata MQGET. Le uscite di conversione dati non possono utilizzare chiamate MQI diverse da MQXCNVC.

Una nuova copia dell'uscita viene caricata quando un'applicazione tenta di recuperare il primo messaggio che utilizza *Format* da quando l'applicazione si è connessa al gestore code. Una nuova copia potrebbe essere caricata anche in altre occasioni se il gestore code ha eliminato una copia precedentemente caricata.

L'uscita conversione dati viene eseguita in un ambiente simile a quello del programma che ha emesso la chiamata MQGET. Oltre alle applicazioni utente, il programma può essere un MCA (message channel agent) che invia messaggi a un gestore code di destinazione che non supporta la conversione dei messaggi. L'ambiente include lo spazio di indirizzo e il profilo utente, dove applicabile. L'uscita non può compromettere l'integrità del gestore code, perché non viene eseguita nell'ambiente del gestore code.

Conversione dati su z/OS

z/OS

Su z/OS, tenere presente quanto segue:

- I programmi di uscita possono essere scritti solo in linguaggio assembly.
- I programmi di uscita devono essere rientranti e in grado di essere eseguiti ovunque nello storage.
- I programmi di uscita devono ripristinare l'ambiente all'uscita su quello all'entrata e devono liberare la memoria ottenuta.
- I programmi di uscita non devono essere WAIT o emettere ESTAEs o SPIEs.
- I programmi di uscita vengono generalmente richiamati come se fossero richiamati da z/OS LINK in:
  - Stato programma problema non autorizzato
  - Modalità di controllo spazio di indirizzo principale
  - Modalità non cross - memory
  - Modalità di registrazione non di accesso
  - modalità di indirizzamento a 31 bit
  - modalità TCB - PRB
- Quando viene utilizzata da un'applicazione CICS , l'exit viene richiamato da EXEC CICS LINK e deve essere conforme alle convenzioni di programmazione CICS . I parametri vengono passati da puntatori (indirizzi) nell'area di comunicazione CICS (COMMAREA).

  Anche se non è consigliato, i programmi di uscita utente possono utilizzare anche le chiamate API CICS , con la seguente attenzione:

– Non emettere punti di sincronizzazione, poiché i risultati potrebbero influenzare le unità di lavoro dichiarate dall'MCA.
– Non aggiornare le risorse controllate da un gestore risorse diverso da IBM MQ for z/OS, incluse quelle controllate da CICS Transaction Server.

Per canali con CONVERT = YES, l'uscita viene caricata dal dataset a cui fa riferimento l'istruzione CSQXLIB DD. Le uscite fornite da MQCSQCBDCI e CSQCBDCO per IBM MQ CICS Bridge si trovano in SCSQAUTH.

Scrittura di un programma di uscita conversione dati per IBM i

Informazioni sulla procedura da considerare durante la scrittura dei programmi di uscita di conversione dati MQ per IBM i.

Eseguire queste operazioni:

1. Denominare il formato del messaggio. Il nome deve rientrare nel campo *Format* di MQMD. Il nome *Format* non deve contenere spazi iniziali e gli spazi finali vengono ignorati. Il nome dell'oggetto non deve contenere più di otto caratteri non vuoti, poiché *Format* è lungo solo otto caratteri. Ricordarsi di utilizzare questo nome ogni volta che si invia un messaggio (il nostro esempio utilizza il nome Formato).
2. Creare una struttura per rappresentare il proprio messaggio. Vedere Sintassi valida per un esempio.
3. Eseguire questa struttura tramite il comando CVTMQMDTA per creare un frammento di codice per l'uscita di conversione dati.

   Le funzioni generate dal comando CVTMQMDTA utilizzano le macro fornite nel file QMQM/H (AMQSVMHA). Queste macro sono scritte supponendo che tutte le strutture siano impacchettate; modificarle in caso contrario.
4. Prendere una copia del file di origine della struttura fornito, QMQMSAMP/QCSRC (AMQSVFC4) e rinominarlo. (Il nostro esempio utilizza il nome EXIT_MOD.)
5. Trovare le seguenti caselle di commento nel file di origine e inserire il codice come descritto:
   a. Verso la fine del file di origine, una casella di commento inizia con:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      Qui, inserire il frammento di codice generato nel passo “3” a pagina 988.
   b. Vicino al centro del file di origine, una casella di commento inizia con:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      Questo è seguito da una chiamata di commento alla funzione `ConverttagSTRUCT`.

      Modificare il nome della funzione nel nome della funzione aggiunta nel passaggio “5.a” a pagina 988. Eliminare i caratteri di commento per attivare la funzione. Se ci sono diverse funzioni, creare chiamate per ognuna di esse.
   c. All'inizio del file di origine, una casella di commento inizia con:

      ```
      /* Insert the function prototypes for the functions produced by */
      ```

      Qui, inserire le istruzioni del prototipo di funzione per le funzioni aggiunte nel passo “5.a” a pagina 988.

   Se il messaggio contiene dati carattere, il codice generato richiama MQXCNVC; ciò può essere risolto collegando il programma di servizio QMQM/LIBMQM.
6. Compilare il modulo di origine, EXIT_MOD, come segue:

   ```
   CRTCMOD MODULE(library/EXIT_MOD) +
   ```

```
   SRCFILE(QCSRC) +
   TERASPACE(*YES *TSIFC)
```

7. Creare / collegare il programma.

   Per le applicazioni senza thread, utilizzare quanto segue:

```
   CRTPGM PGM(library/Format)  +
      MODULE(library/EXIT_MOD)  +
      BNDSRVPGM(QMQM/LIBMQM)  +
      ACTGRP(QMQM)  +
      USRPRF(*USER)
```

   Oltre a creare l'uscita di conversione dati per l'ambiente di base, ne è richiesta un'altra nell'ambiente con thread. Questo oggetto caricabile deve essere seguito da _R. Utilizzare la libreria LIBMQM_R per risolvere le chiamate a MQXCNVC. Entrambi gli oggetti caricabili sono richiesti per un ambiente con thread.

```
   CRTPGM PGM(library/Format_R)  +
      MODULE(library/EXIT_MOD)  +
      BNDSRVPGM(QMQM/LIBMQM_R)  +
      ACTGRP(QMQM)  +
      USRPRF(*USER)
```

8. Inserire l'output nell'elenco librerie per il lavoro IBM MQ . Si consiglia, per la produzione, di memorizzare i programmi di uscita conversione dati in QSYS.

Nota:

1. Se CVTMQMDTA utilizza strutture compresse, tutte le applicazioni IBM MQ devono utilizzare il qualificatore _Packed.
2. I programmi di uscita conversione dati devono essere rientranti.
3. MQXCNVC è l'unica chiamata MQI che può essere emessa da un'uscita di conversione dati.
4. Compilare il programma di uscita con l'opzione compilatore profilo utente impostata su *USER, in modo che l'uscita venga eseguita con l'autorizzazione dell'utente.
5. L'abilitazione della memoria Teraspace è richiesta per tutte le uscite utente con IBM MQ for IBM i ; specificare TERASPACE (*YES *TSIFC) nei comandi CRTCMOD e CRTBNDC.

Scrittura di un programma di uscita conversione dati per IBM MQ for z/OS

Informazioni sui passi da considerare durante la scrittura dei programmi di uscita di conversione dati per IBM MQ for z/OS.

Eseguire queste operazioni:

1. Prendere la struttura di origine fornita CSQ4BAX9 (per ambienti nonCICS) o CSQ4CAX9 (per CICS) come punto di partenza.
2. Eseguire il programma di utilità CSQUCVX.
3. Seguire le istruzioni nel prologo di CSQ4BAX9 o CSQ4CAX9 per incorporare le routine generate dal programma di utilità CSQUCVX, nell'ordine in cui le strutture si verificano nel messaggio che si desidera convertire.
4. Il programma di utilità presuppone che le strutture dati non siano impacchettate, che l'allineamento implicito dei dati sia rispettato e che le strutture inizino su un limite fullword, con i byte ignorati come richiesto (come tra ID e VERSION nell'esempio in Sintassi valida). Se le strutture sono compresse, omettere le macro CMQXCALA generate. Pertanto, considerare la possibilità di dichiarare le proprie strutture in modo tale che tutti i campi vengano denominati e nessun byte venga ignorato; nell'esempio in Sintassi valida, aggiungere un campo "MQBYTE DUMMY;" tra ID e VERSION.
5. L'uscita fornita restituisce un errore se il buffer di input è più breve del formato del messaggio da convertire. Anche se l'uscita converte il maggior numero possibile di campi completi, l'errore causa la restituzione di un messaggio non convertito all'applicazione. Se si desidera consentire la conversione dei buffer di input brevi per quanto possibile, inclusi i campi parziali, modificare il valore TRUNC=

nella macro CSQXCDFA su YES: non viene restituito alcun errore, in modo che l'applicazione riceva un messaggio convertito. L'applicazione deve gestire il troncamento.

6. Aggiungere qualsiasi altro codice di elaborazione speciale necessario.
7. Ridenominare il programma con il proprio nome formato dati.
8. Compilare e collegare - modificare il programma come un programma di applicazione batch (a meno che non sia per l'utilizzo con applicazioni CICS). Le macro nel codice generato dal programma di utilità si trovano nella libreria **thlqual**.SCSQMACS.

   Se il messaggio contiene dati carattere, il codice generato richiama MQXCNVC. Se l'uscita utilizza questa chiamata, modificarla con il programma stub di uscita CSQASTUB. Lo stub è indipendente dalla lingua e dall'ambiente. In alternativa, è possibile caricare lo stub dinamicamente utilizzando il nome della chiamata dinamica CSQXCNVC. Per ulteriori informazioni, fare riferimento a “Richiamo dinamico dello stub IBM MQ” a pagina 1031.

   Inserire il modulo modificato dal collegamento nella propria libreria di caricamento dell'applicazione e in un dataset a cui fa riferimento l'istruzione CSQXLIB DD della procedura di attività avviata dall'iniziatore del canale.
9. Se l'uscita è destinata all'utilizzo da parte delle applicazioni CICS , compilarla e modificarla tramite link come un programma applicativo CICS , incluso CSQASTUB, se richiesto. Inserirlo nella libreria del programma applicativo CICS . Definire il programma su CICS nel modo tipico, specificando EXECKEY (CICS) nella definizione.

Nota: Anche se le librerie di runtime LE/370 sono necessarie per l'esecuzione del programma di utilità CSQUCVX (consultare il passo “2” a pagina 989), non sono necessarie per la modifica del collegamento o per l'esecuzione dell'uscita di conversione dati (consultare i passi “8” a pagina 990 e “9” a pagina 990).

Consultare “Scrittura delle applicazioni bridge IMS” a pagina 76 per informazioni sulla conversione dei dati all'interno del bridge IBM MQ - IMS .

Linux AIX *Scrittura di un'uscita di conversione dati per i sistemi IBM MQ for AIX or Linux*

Informazioni sulle fasi da considerare durante la scrittura di programmi di uscita conversione dati per sistemi IBM MQ for AIX or Linux .

Eseguire queste operazioni:

1. Denominare il formato del messaggio. Il nome deve rientrare nel campo *Format* di MQMD e deve essere in maiuscolo, ad esempio MYFORMAT. Il nome *Format* non deve contenere spazi iniziali. Gli spazi finali vengono ignorati. Il nome dell'oggetto non deve contenere più di otto caratteri non vuoti, poiché *Format* è lungo solo otto caratteri. Ricordarsi di utilizzare questo nome ogni volta che si invia un messaggio.

   Se l'uscita di conversione dati viene utilizzata in un ambiente con thread, l'oggetto caricabile deve essere seguito da _r per indicare che si tratta di una versione con thread.
2. Creare una struttura per rappresentare il proprio messaggio. Vedere Sintassi valida per un esempio.
3. Eseguire questa struttura tramite il comando `crtmqcvx` per creare un frammento di codice per l'uscita di conversione dati.

   Le funzioni generate dal comando `crtmqcvx` utilizzano macro che presuppongono che tutte le strutture siano impacchettate; in caso contrario, modificarle.
4. Copiare il file di origine della struttura fornito, ridenominandolo nel nome del formato del messaggio impostato nel passo “1” a pagina 990. Il file di origine della struttura e la copia sono di sola lettura.

   Il file di origine della struttura è denominato amqsvfc0.c.
5. Su IBM MQ for AIX, viene fornito anche un file di esportazione skeleton denominato amqsvfc.exp . Copiare questo file, ridenominandolo MYFORMAT.EXP.
6. La struttura include un file di intestazione di esempio, amqsvmha.h, nella directory *MQ_INSTALLATION_PATH*/inc, dove *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello

in cui è installato IBM MQ . Verificare che il percorso di inclusione punti a questa directory per selezionare questo file.

Il file amqsvmha.h contiene le macro utilizzate dal codice generato dal comando `crtmqcvx` . Se la struttura da convertire contiene dati carattere, queste macro richiamano MQXCNVC.

7. Trovare le seguenti caselle di commento nel file di origine e inserire il codice come descritto:
   a. Verso la fine del file di origine, una casella di commento inizia con:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      Qui, inserire il frammento di codice generato nel passo “3” a pagina 990.
   b. Vicino al centro del file di origine, una casella di commento inizia con:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      Questo è seguito da una chiamata di commento alla funzione `ConverttagSTRUCT`.

      Modificare il nome della funzione nel nome della funzione aggiunta nel passaggio “7.a” a pagina 991. Eliminare i caratteri di commento per attivare la funzione. Se ci sono diverse funzioni, creare chiamate per ognuna di esse.
   c. All'inizio del file di origine, una casella di commento inizia con:

      ```
      /* Insert the function prototypes for the functions produced by */
      ```

      Qui, inserire le istruzioni del prototipo di funzione per le funzioni aggiunte nel passo “3” a pagina 990.
8. Compilare l'uscita come una libreria condivisa, utilizzando MQStart come punto di ingresso. Per fare ciò, consultare “Compilazione di uscite di conversione dati su sistemi AIX and Linux” a pagina 991.
9. Inserire l'emissione nella directory di uscita. La directory di uscita predefinita è `/var/mqm/exits` per i sistemi a 32 bit e `/var/mqm/exits64`per i sistemi a 64 bit. È possibile modificare tali directory nel file qm.ini o mqclient.ini . Questo percorso può essere impostato per ogni gestore code e l'uscita viene ricercata solo in tale percorso o percorsi.

Nota:

1. Se `crtmqcvx` utilizza strutture compresse, tutte le applicazioni IBM MQ devono essere compilate in questo modo.
2. I programmi di uscita conversione dati devono essere rientranti.
3. MQXCNVC è l'unica chiamata MQI che può essere emessa da un'uscita di conversione dati.

Linux AIX *Compilazione di uscite di conversione dati su sistemi AIX and Linux*

Esempi di come compilare un'uscita di conversione dati su sistemi AIX and Linux .

Su tutte le piattaforme, il punto di accesso al modulo è MQStart.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

AIX

AIX

Compilare il codice di origine di uscita immettendo uno dei comandi seguenti:

Applicazioni a 32 bit

Senza thread

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Threaded

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Applicazioni a 64 bit

Senza thread

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Threaded

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

Compilare il codice di origine di uscita immettendo uno dei comandi seguenti:

Applicazioni a 31 bit

Senza thread

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

Threaded

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Applicazioni a 32 bit

Senza thread

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

Threaded

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Applicazioni a 64 bit

Senza thread

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
```

```
      -I  MQ_INSTALLATION_PATH/inc
```

Threaded

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Windows *Scrittura di un'uscita di conversione dati per IBM MQ for Windows*

Informazioni sui passi da considerare durante la scrittura dei programmi di uscita di conversione dati per IBM MQ for Windows.

Eseguire queste operazioni:

1. Denominare il formato del messaggio. Il nome deve rientrare nel campo *Format* di MQMD. Il nome *Format* non deve contenere spazi iniziali. Gli spazi finali vengono ignorati. Il nome dell'oggetto non deve contenere più di otto caratteri non vuoti, poiché *Format* è lungo solo otto caratteri.

   Un file .DEF denominato amqsvfcn.def viene fornito anche nella directory degli esempi, *MQ_INSTALLATION_PATH*\Tools\C\Samples. *MQ_INSTALLATION_PATH* è la directory in cui è installato IBM MQ . Prendere una copia di questo file e ridenominarlo, ad esempio, in MYFORMAT.DEF. Assicurarsi che il nome della DLL da creare e il nome specificato in MYFORMAT.DEF sono le stesse. Sovrascrivere il nome FORMAT1 in MYFORMAT.DEF con il nome del nuovo formato.

   Ricordarsi di utilizzare questo nome ogni volta che si invia un messaggio.
2. Creare una struttura per rappresentare il proprio messaggio. Vedere Sintassi valida per un esempio.
3. Eseguire questa struttura tramite il comando `crtmqcvx` per creare un frammento di codice per l'uscita di conversione dati.

   Le funzioni generate dal comando CRTMQCVX utilizzano macro scritte presupponendo che tutte le strutture siano impacchettate; in caso contrario, modificarle.
4. Copiare il file di origine della struttura fornita, amqsvfc0.c, ridenominandolo con il nome del formato del messaggio impostato nel passo “1” a pagina 993.

   amqsvfc0.c si trova in *MQ_INSTALLATION_PATH*\Tools\C\Samples dove *MQ_INSTALLATION_PATH* è la directory in cui è installato IBM MQ . La directory di installazione predefinita è C:\Program Files\IBM\MQ.

   La struttura include un file di intestazione di esempio amqsvmha.h nella directory *MQ_INSTALLATION_PATH*\Tools\C\include . Verificare che il percorso di inclusione punti a questa directory per selezionare questo file.

   Il file amqsvmha.h contiene le macro utilizzate dal codice generato dal comando CRTMQCVX. Se la struttura da convertire contiene dati carattere, queste macro richiamano MQXCNVC.
5. Trovare le seguenti caselle di commento nel file di origine e inserire il codice come descritto:
   a. Verso la fine del file di origine, una casella di commento inizia con:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      Qui, inserire il frammento di codice generato nel passo “3” a pagina 993.
   b. Vicino al centro del file di origine, una casella di commento inizia con:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      Questo è seguito da una chiamata di commento alla funzione ConverttagSTRUCT.

Modificare il nome della funzione nel nome della funzione aggiunta nel passaggio “5.a” a pagina 993. Eliminare i caratteri di commento per attivare la funzione. Se ci sono diverse funzioni, creare chiamate per ognuna di esse.

c. All'inizio del file di origine, una casella di commento inizia con:

```
/* Insert the function prototypes for the functions produced by */
```

Qui, inserire le istruzioni del prototipo di funzione per le funzioni aggiunte nel passo “3” a pagina 993.

6. Creare il seguente file di comandi:

```
cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
MYFORMAT.C


MYFORMAT.DEF
```

dove *MQ_INSTALLATION_PATH* è la directory in cui è installato IBM MQ .

7. Immettere il file dei comandi per compilare l'uscita come un file DLL.
8. Inserire l'output nella sottodirectory di uscita sotto la directory di dati IBM MQ . La directory predefinita per l'installazione delle uscite su sistemi a 32 bit è *MQ_DATA_PATH*\Exits e per sistemi a 64 bit è *MQ_DATA_PATH*\Exits64

   Il percorso utilizzato per ricercare le uscite di conversione dati viene fornito nel registro. La cartella del registro è:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

   e la chiave di registro è ExitsDefaultPath. Questo percorso può essere impostato per ogni gestore code e l'uscita viene ricercata solo in tale percorso o percorsi.

Nota:

1. Se CRTMQCVX utilizza strutture compresse, tutte le applicazioni IBM MQ devono essere compilate in questo modo.
2. I programmi di uscita conversione dati devono essere rientranti.
3. MQXCNVC è l'unica chiamata MQI che può essere emessa da un'uscita di conversione dati.

Windows *Uscire e passare ai file di caricamento sui sistemi operativi Windows*

I processi del gestore code IBM WebSphere MQ for Windows 7.5 sono a 32 bit. Di conseguenza, quando si utilizzano applicazioni a 64 bit, alcuni tipi di file di uscita e di file di caricamento degli switch XA devono avere anche una versione a 32 bit disponibile per l'utilizzo da parte del gestore code. Se la versione a 32 bit del file di caricamento dell'uscita o dello switch XA è richiesta e non è disponibile, la chiamata o il comando API pertinente non riesce.

Sono supportati due attributi in qm.ini file per *ExitPath*. Questi sono ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits e ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64. *MQ_INSTALLATION_PATH* rappresenta la directory di livello superiore in cui è installato IBM MQ . L'utilizzo di queste informazioni garantisce che sia possibile trovare la libreria appropriata. Se un'uscita viene utilizzata in un cluster di IBM MQ , ciò garantisce anche che sia possibile trovare la libreria appropriata su un sistema remoto.

La seguente tabella elenca i diversi tipi di file di caricamento Exit e Switch e indica se sono richieste versioni a 32 bit o a 64 bit o entrambe, in base all'utilizzo di applicazioni a 32 bit o a 64 bit:

Tipi di file	Applicazioni a 32 bit	Applicazioni a 64 bit
uscita API	32 bit e 64 bit	64 bit
Uscita conversione dati	32 bit	64 bit
Uscite canale server (tutti i tipi)	64 bit	64 bit
Uscite canale client (tutti i tipi)	32 bit	64 bit
Uscita di servizio installabile	64 bit	64 bit
Uscita WLM cluster	64 bit	64 bit
Uscita instradamento pubblicazione / sottoscrizione	64 bit	64 bit
File di caricamento switch database	32 bit e 64 bit	64 bit
Librerie AX di External Transaction Manager	32 bit	64 bit
Uscita pre - connessione	32 bit	64 bit

Riferimento alle definizioni di connessione mediante un'uscita di pre - connessione da un repository

IBM MQ MQI clients può essere configurato per ricercare un repository per ottenere le definizioni di connessione utilizzando una libreria di uscita pre - connessione.

Introduzione

Un'applicazione client può connettersi a un gestore code utilizzando CCDT (client channel definition tables). In genere, il file CCDT si trova su un server di file di rete centrale e dispone di client che vi fanno riferimento. Poiché è difficile gestire e gestire varie applicazioni client che fanno riferimento al file CCDT, un approccio flessibile consiste nel memorizzare le definizioni client in un repository globale come una directory LDAP, un repository WebSphere Registry and Repository o qualsiasi altro repository. La memorizzazione delle definizioni di connessione client in un repository rende più semplice la gestione delle definizioni di connessione client e le applicazioni possono accedere alle definizioni di connessione client corrette e più aggiornate.

Durante l'esecuzione della chiamata MQCONN/X, IBM MQ MQI client carica una libreria di uscita di preconnessione specificata dall'applicazione e richiama una funzione di uscita per richiamare le definizioni di connessione. Le definizioni di connessione richiamate vengono quindi utilizzate per stabilire una connessione a un gestore code. I dettagli della libreria di uscita e della funzione da richiamare sono specificati nel file di configurazione mqclient.ini .

Sintassi

`void MQ_PRECONNECT_EXIT` (*pExitParms*, *pQMgrName*, *ppConnectOpts*, *pCompCode*, *pReason*);

Parametri

Parametri pExit

Tipo: input / output PMQNXP

La struttura del parametro di uscita **`PreConnection`** .

La struttura è assegnata e gestita dal chiamante dell'uscita.

Nome pQMgr

Tipo: input/output PMQCHAR

Il nome del gestore code.

In fase di input, questo parametro è la stringa di filtro fornita alla chiamata API MQCONN tramite il parametro **QMgrName** . Questo campo potrebbe essere vuoto, esplicito o contenere determinati caratteri jolly. Il campo viene modificato dall'uscita. Il parametro è NULL quando l'uscita viene chiamata con MQXR_TERM.

Opzioni ppConnect
Tipo: ppConnectOpts input/output

Opzioni che controllano l'azione di MQCONNX.

Questo è un puntatore a una struttura di opzioni di connessione MQCNO che controlla l'azione della chiamata API MQCONN. Il parametro è NULL quando l'uscita viene chiamata con MQXR_TERM. Il client MQI fornisce sempre una struttura MQCNO all'uscita, anche se non è stata originariamente fornita dall'applicazione. Se un'applicazione fornisce una struttura MQCNO, il client effettua un duplicato per inoltrarlo all'uscita in cui viene modificato. Il client conserva la proprietà di MQCNO.

Un MQCD a cui si fa riferimento tramite MQCNO ha la precedenza su qualsiasi definizione di connessione fornita tramite l'array. Il client utilizza la struttura MQCNO per connettersi al gestore code e gli altri vengono ignorati.

Codice pComp
Tipo: input / output PMQLONG

Codice di completamento.

Puntatore a un MQLONG che riceve il codice di completamento delle uscite. Deve essere uno dei seguenti valori:

- MQCC_OK - Completamento riuscito
- MQCC_WARNING - Avviso (completamento parziale)
- MQCC_FAILED - Chiamata non riuscita

pReason
Tipo: input / output PMQLONG

Codice di qualificazione motivo pComp.

Puntatore ad un MQLONG che riceve il codice motivo di uscita. Se il codice di completamento è MQCC_OK, l'unico valore valido è:

- MQRC_NONE - (0, x '000') Nessun motivo per la notifica.

Se il codice di completamento è MQCC_FAILED o MQCC_WARNING, la funzione di uscita può impostare il campo del codice motivo su qualsiasi valore MQRC_ * valido.

Richiamo C

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

Scrittura e compilazione di uscite di pubblicazione

È possibile configurare un'uscita di pubblicazione sul gestore code per modificare i contenuti di un messaggio pubblicato prima che venga ricevuto dai sottoscrittori. È anche possibile modificare l'intestazione del messaggio o non consegnare il messaggio a una sottoscrizione.

Nota: Le uscite di pubblicazione non sono supportate su z/OS.

È possibile utilizzare l'uscita di pubblicazione per esaminare e modificare i messaggi consegnati ai sottoscrittori:

- Esaminare il contenuto di un messaggio pubblicato per ciascun sottoscrittore
- Modificare il contenuto di un messaggio pubblicato per ciascun sottoscrittore
- Modificare la coda in cui viene inserito un messaggio
- Arrestare la consegna di un messaggio a un sottoscrittore

Scrittura di un'uscita di pubblicazione

Utilizzare i passi riportati in “Scrittura di uscite e servizi installabili su AIX, Linux, and Windows” a pagina 938per scrivere e compilare l'uscita.

Il provider dell'uscita di pubblicazione definisce le operazioni dell'uscita. L'uscita, tuttavia, deve essere conforme alle regole definite in MQPSXP.

IBM MQ non fornisce un'implementazione del punto di ingresso MQ_PUBLISH_EXIT. Fornisce una dichiarazione typedef di linguaggio C. Utilizzare typedef per dichiarare correttamente i parametri in un'uscita scritta dall'utente. Il seguente esempio illustra come utilizzare la dichiarazione typedef:

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

L'uscita di pubblicazione viene eseguita nel processo del gestore code, come risultato delle seguenti operazioni:

- Un'operazione di pubblicazione in cui un messaggio viene consegnato a uno o più sottoscrittori
- Un'operazione di sottoscrizione in cui vengono consegnati uno o più messaggi conservati
- Un'operazione di richiesta di sottoscrizione in cui vengono consegnati uno o più messaggi conservati

Se l'uscita di pubblicazione viene richiamata per una connessione, la prima volta che viene richiamata viene impostato un codice *ExitReason* di MQXR_INIT . Prima che la connessione si disconnetta dopo aver utilizzato un'uscita di pubblicazione, l'uscita viene richiamata con un codice *ExitReason* di MQXR_TERM.

Se l'uscita di pubblicazione è configurata, ma non può essere caricata quando il gestore code viene avviato, le operazioni di pubblicazione / sottoscrizione dei messaggi sono inibite per il gestore code. È necessario risolvere il problema o riavviare il gestore code prima di riabilitare la messaggistica di pubblicazione / sottoscrizione.

Ogni connessione IBM MQ che richiede l'uscita di pubblicazione potrebbe non riuscire a caricare o inizializzare l'uscita. Se l'uscita non viene caricata o inizializzata, le operazioni di pubblicazione / sottoscrizione che richiedono l'uscita di pubblicazione vengono disabilitate per tale connessione. Le operazioni hanno esito negativo con il IBM MQ codice di errore MQRC_PUBLISH_EXIT_ERROR.

Il contesto in cui viene richiamata l'uscita di pubblicazione è la connessione di un'applicazione al gestore code. Un'area dati utente viene gestita dal gestore code per ogni connessione che sta eseguendo operazioni di pubblicazione. L'uscita può conservare le informazioni nell'area dati utente per ogni connessione.

Un'uscita di pubblicazione può utilizzare alcune chiamate MQI. Può utilizzare solo quelle chiamate MQI che gestiscono le proprietà del messaggio. Le chiamate sono:

- MQBUFMH

- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Se l'uscita di pubblicazione modifica il gestore code di destinazione o il nome della coda, non viene eseguito alcun nuovo controllo di autorizzazione.

Compilazione di un'uscita di pubblicazione

L'uscita di pubblicazione è una libreria caricata dinamicamente; può essere considerata come un'uscita canale. Per informazioni sulla compilazione delle uscite, consultare "Scrittura di uscite e servizi installabili su AIX, Linux, and Windows" a pagina 938.

Uscita di pubblicazione di esempio

Il programma di uscita di esempio è denominato `amqspse0.c`. Scrive un messaggio diverso in un file di log a seconda che l'exit sia stata richiamata per le operazioni di inizializzazione, pubblicazione o terminazione. Dimostra inoltre l'utilizzo del campo dell'area utente di uscita per allocare e liberare la memoria in modo appropriato.

Configurazione delle uscite di pubblicazione

È necessario definire alcuni attributi per configurare un'uscita di pubblicazione.

Su Windows e Linux è possibile utilizzare IBM MQ explorer per definire gli attributi. Gli attributi sono definiti nella pagina delle proprietà del gestore code, in Pubblicazione / Sottoscrizione.

Per configurare l'uscita di pubblicazione nel file `qm.ini` sui sistemi AIX and Linux , creare una sezione denominata `PublishSubscribe`. La stanza `PublishSubscribe` ha i seguenti attributi:

PublishExitPath = [path] |*nome_modulo*
: Il nome e il percorso del modulo contenente il codice di uscita di pubblicazione. La lunghezza massima di questo campo è `MQ_EXIT_NAME_LENGTH`. L'impostazione predefinita è nessuna uscita di pubblicazione.

PublishExitFunzione = *nome_funzione*
: Il nome del punto di ingresso della funzione nel modulo che contiene il codice di uscita pubblicazione. La lunghezza massima di questo campo è `MQ_EXIT_NAME_LENGTH`.

  IBM i Su IBM i, se viene utilizzato un programma, omettere `PublishExitFunction`.

PublishExitData = *stringa*
: Se il gestore code sta richiamando un'uscita di pubblicazione, passa una struttura `MQPSXP` come input. I dati specificati utilizzando l'attributo **PublishExitData** vengono forniti nel campo *ExitData* della struttura. La stringa può avere una lunghezza massima di `MQ_EXIT_DATA_LENGTH` caratteri. Il valore predefinito è 32 caratteri vuoti.

Scrittura e compilazione delle uscite del carico di lavoro del cluster

Scrivere un programma di uscita del carico di lavoro del cluster per personalizzare la gestione del carico di lavoro dei cluster. È possibile prendere in considerazione il costo dell'utilizzo di un canale in diverse ore del giorno o il contenuto del messaggio quando si instradano i messaggi. Questi sono fattori che non vengono considerati dall'algoritmo di gestione del workload standard.

Nella maggior parte dei casi, l'algoritmo di gestione del carico di lavoro è sufficiente per le proprie esigenze. Tuttavia, in modo da poter fornire il proprio programma di uscita utente per adattare la gestione del carico di lavoro, IBM MQ include un'uscita utente, l'uscita del carico di lavoro cluster.

Potresti avere alcune informazioni particolari sulla tua rete o sui messaggi che potresti utilizzare per influenzare il bilanciamento del carico di lavoro. Potresti sapere quali sono i canali ad alta capacità o gli instradamenti di rete economici oppure potresti voler instradare i messaggi in base al loro contenuto. È possibile decidere di scrivere un programma di uscita del carico di lavoro del cluster o utilizzarne uno fornito da una terza parte.

L'uscita del carico di lavoro del cluster viene richiamata quando si accede a una coda cluster. Viene richiamato da `MQOPEN`, `MQPUT1` e `MQPUT`.

Il gestore code di destinazione selezionato all'ora `MQOPEN` è fisso se è specificato `MQOO_BIND_ON_OPEN` . In questo caso l'uscita viene eseguita solo una volta.

Se il gestore code di destinazione non è corretto all'ora `MQOPEN` , il gestore code di destinazione viene scelto al momento della chiamata `MQPUT` . Se il gestore code di destinazione non è disponibile o ha esito negativo mentre il messaggio è ancora sulla coda di trasmissione, l'uscita viene richiamata di nuovo. Viene selezionato un nuovo gestore code di destinazione. Se il canale dei messaggi ha esito negativo durante il trasferimento del messaggio e viene eseguito il backout del messaggio, viene selezionato un nuovo gestore code di destinazione.

Multi Su Multiplatforms, il gestore code carica la nuova uscita del carico di lavoro del cluster al successivo avvio del gestore code.

Se la definizione del gestore code non contiene un nome del programma di uscita del carico di lavoro del cluster, l'uscita del carico di lavoro del cluster non viene richiamata.

Vari dati vengono passati a un'exit del carico di lavoro del cluster nella struttura del parametro di uscita, `MQWXP`:

- La struttura di definizione dei messaggi, `MQMD`.
- Il parametro di lunghezza del messaggio.
- Una copia del messaggio o parte di esso.

Su piattaforme nonz/OS , se si utilizza `CLWLMode=FAST`, ogni processo del sistema operativo carica la propria copia dell'uscita. Connessioni differenti al gestore code possono causare il richiamo di copie differenti dell'exit. Se l'uscita viene eseguita in modalità sicura predefinita, `CLWLMode=SAFE`, una singola copia dell'uscita viene eseguita nel proprio processo separato.

Scrittura delle uscite del carico di lavoro del cluster

z/OS Per informazioni sulla scrittura delle uscite del carico di lavoro del cluster per z/OS, consultare “Programmazione dell'uscita del carico di lavoro del cluster per IBM MQ for z/OS” a pagina 1001.

Da IBM MQ 9.1.0, le uscite del carico di lavoro del cluster vengono eseguite nello spazio di indirizzo dell'iniziatore del canale, invece che nello spazio di indirizzo del gestore code. Se si dispone di un'uscita del carico di lavoro del cluster, è necessario rimuovere l'istruzione CSQXLIB DD dalla procedura dell'attività avviata del gestore code e aggiungere il dataset che contiene l'uscita del carico di lavoro del cluster alla concatenazione CSQXLIB sulla procedura dell'attività avviata dall'iniziatore del canale.

Multi Per Multiplatforms, le uscite del carico di lavoro del cluster non devono utilizzare chiamate MQI. Per altri aspetti, le norme per la scrittura e la compilazione dei programmi di uscita del workload del cluster sono simili alle regole che si applicano ai programmi di uscita del canale. Seguire le istruzioni riportate in “Scrittura di uscite e servizi installabili su AIX, Linux, and Windows” a pagina 938e utilizzare il programma di esempio “Uscita del carico di lavoro del cluster di esempio” a pagina 1000 per scrivere e compilare l'exit.

Per ulteriori informazioni sulle uscite del canale, consultare “Scrittura di programmi di uscita canale” a pagina 966.

Configurazione delle uscite del carico di lavoro del cluster

È possibile denominare le uscite del carico di lavoro del cluster nella definizione del gestore code specificando l'attributo di uscita del carico di lavoro cluster nel comando ALTER QMGR . Ad esempio:

```
ALTER QMGR CLWLEXIT(myexit)
```

Riferimenti correlati

Chiamate di uscita del carico di lavoro cluster e strutture dati

Uscita del carico di lavoro del cluster di esempio

IBM MQ include un programma di uscita del carico di lavoro del cluster di esempio. È possibile copiare l'esempio e utilizzarlo come base per i propri programmi.

z/OS **IBM MQ for z/OS**

Il programma di uscita del workload del cluster di esempio viene fornito in Assembler e in C. La versione Assembler è denominata CSQ4BAF1 ed è disponibile nella libreria `thlqual.SCSQASMS`. La versione C è denominata CSQ4BCF1 ed è disponibile nella libreria `thlqual.SCSQC37S`. `thlqual` è il qualificatore di alto livello della libreria di destinazione per i dataset IBM MQ nell'installazione.

Multi **IBM MQ for Multiplatforms**

Il programma di uscita del carico di lavoro del cluster di esempio viene fornito in C e viene denominato `amqswlm0.c`. Può essere trovato in:

Tabella 144. Percorso del programma di uscita del workload del cluster di esempio per Multiplatforms

Piattaforma	**Percorso file**
AIX AIX	*MQ_INSTALLATION_PATH*`/samp`
Windows Windows	*MQ_INSTALLATION_PATH* `\Tools\c\Samples`
IBM i IBM i	La libreria `qmqm`

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Questa uscita di esempio instrada tutti i messaggi a un particolare gestore code, a meno che tale gestore code non diventi non disponibile. Reagisce all'errore del gestore code instradando i messaggi a un altro gestore code.

Indicare a quale gestore code si desidera inviare i messaggi. Specificare il nome del canale ricevente del cluster nell'attributo CLWLDATA nella definizione del gestore code. Ad esempio:

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Per abilitare l'exit, fornire il nome e il percorso completo nell'attributo CLWLEXIT :

Linux AIX Su AIX and Linux:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows Su Windows:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS Su z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

dove x è `'A'` o `'C'`, in base al linguaggio di programmazione della versione che si sta utilizzando.

IBM i Su IBM i, utilizzare uno dei comandi seguenti:

- Utilizzare il comando MQSC:

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  Sia il nome del programma che il nome della libreria occupano 10 caratteri e, se necessario, vengono riempiti di spazi vuoti a destra.
- Utilizzare il comando CL:

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

Ora, invece di utilizzare l'algoritmo di gestione del carico di lavoro fornito, IBM MQ richiama questa uscita per instradare tutti i messaggi al gestore code scelto.

z/OS *Programmazione dell'uscita del carico di lavoro del cluster per IBM MQ for z/OS*

Le uscite del carico di lavoro del cluster vengono richiamate come se fossero da un comando z/OS **LINK** . Le uscite sono soggette a una serie di norme di programmazione rigorose. Evitare di utilizzare la maggior parte dei comandi SVC che implicano attese o l'utilizzo di STAE o ESTAE in un'uscita del carico di lavoro.

Le uscite del carico di lavoro del cluster vengono richiamate come se fossero da un z/OS **LINK** in:

- Stato programma problema non autorizzato
- Modalità di controllo spazio di indirizzo principale
- Modalità non cross - memory
- Modalità registro di non accesso
- modalità di indirizzamento a 31 bit
- Chiave di memoria 8
- Maschera chiave programma 8
- Chiave TCB 8

Inserire i moduli modificati dal link nel dataset specificato dall'istruzione `CSQXLIB DD` della procedura dell'attività avviata dell'iniziatore di canali. I nomi dei moduli di caricamento vengono specificati come nomi di uscita del workload nella definizione del gestore code.

Quando si scrivono uscite del workload per IBM MQ for z/OS, si applicano le seguenti regole:

- È necessario scrivere le uscite in assembler o C. Se si utilizza C, deve essere conforme all'ambiente di programmazione dei sistemi C per le uscite di sistema, descritto nel manuale *z/OS C/C++ Programming Guide*, SC09-4765.
- Se si utilizza la chiamata `MQXCLWLN` , modificare il link con `CSQMFCLW`, fornito in *`thlqual`*`.SCSQLOAD`.
- Le uscite vengono caricate dalle librerie non autorizzate definite da una istruzione `CSQXLIB DD` . Se `CSQXLIB` dispone di `DISP=SHR`, le uscite possono essere aggiornate mentre il gestore code è in esecuzione, con la nuova versione utilizzata nel thread `MQCONN` successivo che il gestore code avvia.
- Le uscite devono essere rientranti e in grado di essere eseguite ovunque nella memoria virtuale.
- Le uscite devono resettare l'ambiente al ritorno a quello alla voce.
- Le uscite devono liberare qualsiasi memoria ottenuta oppure assicurarsi che la memoria sia liberata da un successivo richiamo di uscita.

- Non sono consentite chiamate MQI.
- Le uscite non devono utilizzare alcun servizio di sistema che potrebbe causare un'attesa, poiché un'attesa compromette gravemente le prestazioni del gestore code. In generale, quindi, evitare un SVC, un PC o un I/O.
- Le uscite non devono emettere un ESTAE o SPIE, ad eccezione delle attività secondarie che allegano.

Nota: Non ci sono restrizioni assolute su ciò che è possibile fare in una uscita. Tuttavia, la maggior parte degli SVC coinvolgono le attese, quindi evitarle, ad eccezione dei seguenti comandi:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Non utilizzare ESTAE ed ESPIE perché la loro gestione degli errori potrebbe interferire con la gestione degli errori eseguita da IBM MQ. IBM MQ potrebbe non essere in grado di eseguire il ripristino da un errore oppure il programma di uscita potrebbe non ricevere tutte le informazioni sull'errore.

Il parametro di sistema EXITLIM limita la quantità di tempo per cui un'uscita può essere eseguita. Il valore predefinito per EXITLIM è 30 secondi. Se vedi il codice di ritorno MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' la tua uscita potrebbe essere in loop. Se si ritiene che l'uscita abbia bisogno di più di 30 secondi per essere completata, aumentare il valore di EXITLIM.

Creazione di un'applicazione procedurale

È possibile scrivere un'applicazione IBM MQ in uno dei diversi linguaggi procedurali ed eseguire l'applicazione su diverse piattaforme.

AIX Creazione dell'applicazione procedurale su AIX

Le pubblicazioni AIX descrivono come creare applicazioni eseguibili dai programmi scritti.

Questo argomento descrive le attività aggiuntive e le modifiche alle attività standard che è necessario eseguire quando si creano le applicazioni IBM MQ for AIX da eseguire in AIX. Sono supportati C, C + + e COBOL. Per informazioni sulla preparazione dei programmi C + +, consultare Utilizzo di C++.

Le attività che devi eseguire per creare un'applicazione eseguibile utilizzando IBM MQ for AIX variano con il linguaggio di programmazione in cui è scritto il codice di origine. Oltre a codificare le chiamate MQI nel codice di origine, è necessario aggiungere le istruzioni di lingua appropriate per includere i file di inclusione IBM MQ for AIX per la lingua che si sta utilizzando. Acquisire familiarità con il contenuto di questi file. Consultare “File di definizione dati IBM MQ” a pagina 718 per una descrizione completa.

Quando si eseguono applicazioni server con thread o client con thread, impostare la variabile di ambiente AIXTHREAD_SCOPE = S.

AIX *Preparazione dei programmi C in AIX*

Questo argomento contiene informazioni sul collegamento delle librerie necessarie per preparare i programmi C su AIX.

I programmi C precompilati vengono forniti nella directory *MQ_INSTALLATION_PATH*/samp/bin . Utilizzare il compilatore ANSI ed eseguire i seguenti comandi. Per ulteriori informazioni sulla programmazione di applicazioni a 64 bit, consultare Coding standards on 64 - bit platforms.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Per applicazioni a 32 bit:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

dove amqsput0 è un programma di esempio.

Per le applicazioni a 64 bit:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

dove `amqsput0` è un programma di esempio.

V 9.3.5 Per le applicazioni a 32 bit che utilizzano il compilatore XLC 17:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

dove `amqsput0` è un programma di esempio.

V 9.3.5 Per applicazioni a 64 bit che utilizzano il compilatore XLC 17:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

dove `amqsput0` è un programma di esempio.

Se si utilizza il compilatore VisualAge C/C++ per i programmi C++, è necessario includere l'opzione `-q namemangling=v5` per risolvere tutti i simboli IBM MQ quando si collegano le librerie.

Se si desidera utilizzare i programmi su una macchina su cui è installato solo IBM MQ MQI client for AIX , ricompilare invece i programmi per collegarli alla libreria client (`-lmqic`).

Collegamento di librerie

Sono necessarie le seguenti librerie:

- Collegare i programmi con la libreria appropriata fornita da IBM MQ.

  In un ambiente senza thread, collegarsi a una delle seguenti librerie:

File di libreria	Tipo di programma / uscita
libmqm.a	Server per C
libmqic.a & libmqm.a	Client per C

  In un ambiente con thread, collegarsi a una delle seguenti librerie:

File di libreria	Tipo di programma / uscita
libmqm_r.a	Server per C
libmqic_r.a & libmqm_r.a	Client per C

  Ad esempio, per creare un'applicazione IBM MQ con thread semplice da una singola unità di compilazione, eseguire questi comandi.

  Per applicazioni a 32 bit:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

  dove `amqsput0` è un programma di esempio.

  Per le applicazioni a 64 bit:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

  dove `amqsput0` è un programma di esempio.

  V 9.3.5 Per le applicazioni a 32 bit che utilizzano il compilatore XLC 17:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

dove `amqsput0` è un programma di esempio.

V 9.3.5 Per applicazioni a 64 bit che utilizzano il compilatore XLC 17:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

dove `amqsput0` è un programma di esempio.

Se si desidera utilizzare i programmi su una macchina su cui è installato solo IBM MQ MQI client for AIX , ricompilare invece i programmi per collegarli alla libreria client (`-lmqic`).

Nota:

1. Non è possibile collegarsi a più di una libreria. In altre parole, non è possibile collegarsi contemporaneamente a una libreria con thread e non con thread.
2. Se si sta scrivendo un servizio installabile (consultare Amministrazione IBM MQ per ulteriori informazioni), è necessario collegarsi alla libreria `libmqmzf.a` in un'applicazione non con thread e alla libreria `libmqmzf_r.a` in un'applicazione con thread.
3. Se si sta producendo un'applicazione per il coordinamento esterno da un gestore transazioni compatibile con XA come IBM TXSeries, Encina o BEA Tuxedo, è necessario collegarsi a `libmqmxa.a` (o `libmqmxa64.a` se il gestore transazioni considera il tipo 'long' come 64 bit) e alle librerie `libmqz.a` in un'applicazione non con thread e a `libmqmxa_r.a` (o `libmqmxa64_r.a`) e `libmqz_r.a` in un'applicazione con thread.
4. È necessario collegare le applicazioni attendibili alle librerie IBM MQ con thread. Tuttavia, è possibile collegare solo un thread alla volta in un'applicazione attendibile su sistemi IBM MQ for AIX or Linux .
5. È necessario collegare le librerie IBM MQ prima di qualsiasi altra libreria del prodotto.

AIX *Preparazione dei programmi COBOL in AIX*

Utilizzare queste informazioni durante la preparazione dei programmi COBOL in AIX utilizzando IBM COBOL Set e Micro Focus COBOL.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

- I copy book COBOL a 32 bit sono installati nella seguente directory:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy32
  ```

  e i collegamenti simbolici vengono creati in:

  ```
  MQ_INSTALLATION_PATH/inc
  ```

- I copy book COBOL a 64 bit sono installati nella seguente directory:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

Nei seguenti esempi impostare la variabile di ambiente **COBCPY** su:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

per applicazioni a 32 bit e:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

per applicazioni a 64 bit.

È necessario collegare il programma a uno dei seguenti file di libreria:

File di libreria	Tipo di programma / uscita
libmqmcb.a	Server per COBOL (applicazione senza thread)
libmqmcb_r.a	Server per COBOL (applicazione con thread)
libmqicb.a	Client per COBOL (applicazione senza thread)
libmqicb_r.a	Client per COBOL (applicazione con thread)

È possibile utilizzare il compilatore IBM COBOL Set o Micro Focus COBOL a seconda del programma:

- I programmi che iniziano con `amqm` sono adatti per il compilatore Micro Focus COBOL e
- I programmi che iniziano con `amq0` sono adatti per il compilatore.

Preparazione dei programmi COBOL utilizzando IBM COBOL Set for AIX

I programmi COBOL di esempio vengono forniti con IBM MQ. Per compilare tale programma, immettere il comando appropriato dal seguente elenco:

Applicazione server senza thread a 32 bit

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

Applicazione client senza thread a 32 bit

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

Applicazione server con thread a 32 bit

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Applicazione client con thread a 32 bit

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Applicazione server senza thread a 64 bit

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

Applicazione client senza thread a 64 bit

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

Applicazione server con thread a 64 bit

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Applicazione client con thread a 64 bit

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Preparazione dei programmi COBOL utilizzando Micro Focus COBOL

Impostare le variabili di ambiente prima di compilare il proprio programma come segue:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Per compilare un programma COBOL a 32 bit utilizzando Micro Focus COBOL, immettere:

- Server per COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- Client per COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- Server con thread per COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Client con thread per COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Per compilare un programma COBOL a 64 bit utilizzando Micro Focus COBOL, immettere:

- Server per COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- Client per COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- Server con thread per COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Client con thread per COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

dove `amqminqx` è un programma di esempio

Consultare la documentazione Micro Focus COBOL per una descrizione delle variabili di ambiente che è necessario impostare.

AIX *Preparazione dei programmi di applicazione CICS in AIX*

Utilizzare queste informazioni durante la preparazione dei programmi CICS in AIX.

Utilizzare i moduli *switch XA* per collegare CICS con IBM MQ. Per ulteriori informazioni sulla struttura switch XA, consultare Strutture switch XA.

Il file di codice sorgente di esempio viene fornito per consentire lo sviluppo di switch XA per altri messaggi di transazione. Il nome del modulo di caricamento switch fornito è elencato inTabella 145 a pagina 1007.

Tabella 145. Codice essenziale per programmi di applicazione CICS su AIX: routine di inizializzazione XA

Descrizione	**C (origine)**	**C (exec) - aggiungi al tuo XAD.Stanza**
Routine di inizializzazione XA	amqzscix.c	*amqzsc* - CICS per AIX

Utilizzare la versione preintegrata del IBM MQ file di caricamento switch *amqzsc*, fornito con il prodotto.

Collegare sempre le transazioni C con la libreria IBM MQ threadsafe *libmqm_r.a.*, e le transazioni COBOL con la libreria COBOL *libmqmcb_r.a*.

È possibile trovare ulteriori informazioni sul supporto delle transazioni CICS nel manuale Amministrazione IBM MQ IBM MQ System Administration Guide.

AIX *Supporto TXSeries CICS*

IBM MQ su AIX supporta TXSeries CICS utilizzando l'interfaccia XA. Assicurarsi che le applicazioni CICS siano collegate alla versione con thread delle librerie IBM MQ .

È possibile eseguire programmi CICS utilizzando IBM COBOL Set for AIX o Micro Focus COBOL. Le sezioni seguenti descrivono la differenza tra i programmi CICS in esecuzione su IBM COBOL Set for AIX e Micro Focus COBOL.

Scrivere i programmi IBM MQ caricati nella stessa regione CICS in C o COBOL. Non è possibile effettuare una combinazione di chiamate C e COBOL MQI nella stessa regione CICS . La maggior parte delle chiamate MQI nella seconda lingua utilizzata ha esito negativo con un codice motivo `MQRC_HOBJ_ERROR`.

Preparazione dei programmi CICS COBOL utilizzando IBM COBOL Set for AIX

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Per utilizzare IBM COBOL, attenersi alla seguente procedura:

1. Esportare la seguente variabile di ambiente:

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                -e _iwz_cobol_main  \
```

   dove `LIB` è una direttiva del compilatore.
2. Convertire, compilare e collegare il programma immettendo:

```
cicstcl -l IBMCOB yourprog.ccp
```

Preparazione dei programmi CICS COBOL utilizzando Micro Focus COBOL

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Per utilizzare Micro Focus COBOL, attenersi alla seguente procedura:

1. Aggiungere il modulo della libreria di runtime IBM MQ COBOL alla libreria di runtime utilizzando il seguente comando:

```
cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

Nota: Con `cicsmkcobol`, IBM MQ non consente di effettuare chiamate MQI nel linguaggio di programmazione C dall'applicazione COBOL.

Se le applicazioni esistenti dispongono di tali chiamate, si consiglia di spostare queste funzioni dalle applicazioni COBOL alla propria libreria, ad esempio, `myMQ.so`. Dopo lo spostamento delle funzioni, non includere la IBM MQ libreria `libmqmcbrt.o` quando si crea l'applicazione COBOL per CICS.

Inoltre, se l'applicazione COBOL non effettua alcuna chiamata COBOL MQI, non collegare `libmqmz_r` con `cicsmkcobol`.

Ciò crea il file del metodo del linguaggio Micro Focus COBOL e abilita la libreria CICS runtime COBOL a richiamare i sistemi IBM MQ for AIX or Linux .

Nota: Eseguire `cicsmkcobol` solo quando si installa uno dei seguenti prodotti:

- Nuova versione o release di Micro Focus COBOL
- Nuova versione o release di CICS per AIX
- Nuova versione o release di qualsiasi prodotto database supportato (solo per transazioni COBOL)
- Nuova versione o release di IBM MQ

2. Esportare la seguente variabile di ambiente:

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. Convertire, compilare e collegare il programma immettendo:

```
cicstcl -l COBOL -e yourprog.ccp
```

Preparazione dei programmi CICS C

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Creare programmi CICS C utilizzando le funzioni CICS standard:

1. Esportare ***una*** delle seguenti variabili di ambiente:
   - LDFLAGS= "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export USERLIB
2. Convertire, compilare e collegare il programma immettendo:

```
cicstcl -l C amqscic0.ccs
```

Transazione di esempio CICS C

L'origine C di esempio per una transazione AIX IBM MQ è fornita da AMQSCIC0.CCS. La transazione legge i messaggi dalla coda di trasmissione SYSTEM.SAMPLECICS.WORKQUEUE sul gestore code predefinito e le colloca nella coda locale con un nome coda contenuto nell'intestazione di trasmissione del messaggio. Eventuali errori vengono inviati alla coda SYSTEM.SAMPLE.CICS.DLQ. Utilizzare lo script MQSC di esempio AMQSCIC0.TST per creare queste code e le code di input di esempio.

IBM i Creazione dell'applicazione procedurale su IBM i

Le pubblicazioni IBM i descrivono come creare applicazioni eseguibili dai programmi scritti, da eseguire con IBM i su sistemi iSeries o System i .

Questo argomento descrive le attività aggiuntive e le modifiche alle attività standard che è necessario eseguire quando si creano applicazioni procedurali IBM MQ for IBM i da eseguire su sistemi IBM i . Sono supportati i linguaggi di programmazione COBOL, C, C + +, Java e RPG. Per informazioni sulla

preparazione dei programmi C + +, consultare Utilizzo di C++. Per informazioni sulla preparazione dei programmi Java , consultare Utilizzo di IBM MQ classes for Java.

Le attività che è possibile eseguire per creare un'applicazione IBM MQ for IBM i eseguibile dipendono dal linguaggio di programmazione in cui è scritto il codice sorgente. Oltre a codificare le chiamate MQI nel codice sorgente, è necessario aggiungere le istruzioni di lingua appropriate per includere i file di definizione dati IBM MQ for IBM i per il linguaggio che si sta utilizzando. Acquisire familiarità con il contenuto di questi file. Consultare “File di definizione dati IBM MQ” a pagina 718 per una descrizione completa.

IBM i *Preparazione dei programmi C in IBM i*

IBM MQ for IBM i supporta messaggi di dimensioni fino a 100 MB. I programmi applicativi scritti in ILE C, che supportano i messaggi IBM MQ superiori a 16 MB, devono utilizzare l'opzione del compilatore `Teraspace` per assegnare memoria sufficiente per questi messaggi.

Per ulteriori informazioni sulle opzioni del compilatore C, consultare *WebSphere Development Studio ILE C/C++ Programmer's Guide*.

Per compilare un modulo C, è possibile utilizzare il comando IBM i **CRTCMOD**. Assicurarsi che la libreria contenente i file di inclusione (QMQM) si trovi nell'elenco librerie durante la compilazione.

È necessario, quindi, collegare l'output del compilatore al programma di servizio utilizzando il comando **CRTPGM** .

Tabella 146. Esempio di CRTPGM in ambienti senza sottoprocessi e sottoprocessi

Tipo di ambiente	Comando	Tipo di programma / uscita
Ambiente senza thread	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM)`	Server o client per C
Ambiente con thread	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM_R)`	Server o client per C

dove *pgmname* è il nome del programma.

La Tabella 147 a pagina 1009 elenca le librerie necessarie quando si preparano programmi C su IBM i in un ambiente senza thread e in un ambiente con thread.

Tabella 147. Librerie necessarie per ambienti senza thread e con thread

Tipo di ambiente	File di libreria	Tipo di programma / uscita
Ambiente senza thread	LIBMQM	Server per C
	LIBMQIC & LIBMQM	Client per C
Ambiente con thread	LIBMQM_R	Server per C
	LIBMQIC_R & LIBMQM_R	Client per C

IBM i *Preparazione dei programmi COBOL in IBM i*

Informazioni sulla preparazione dei programmi COBOL in IBM i e sul metodo di accesso a MQI dall'interno del programma COBOL.

Informazioni su questa attività

Per accedere a MQI dai programmi COBOL, IBM MQ for IBM i fornisce un'interfaccia di chiamata procedurale collegata fornita dai programmi di servizio. Ciò fornisce l'accesso a tutte le funzioni MQI

in IBM MQ for IBM i e il supporto per le applicazioni con thread. Questa interfaccia può essere utilizzata solo con il compilatore ILE COBOL.

La sintassi COBOL CALL standard viene utilizzata per accedere alle funzioni MQI.

I file di copia COBOL contenenti le costanti denominate e le definizioni di struttura da utilizzare con MQI sono contenuti nel file fisico di origine QMQM/QCBLLESRC.

I file di copia COBOL utilizzano il carattere virgolette singole (') come delimitatore di stringa. I compilatori IBM i COBOL presuppongono che il delimitatore sia la virgoletta ("). Per evitare che i compilatori generino messaggi di avvertenza, specificare `OPTION (*APOST)` sui comandi **CRTCBLPGM**, **CRTBNDCBL**o **CRTCBLMOD**.

Per fare in modo che il compilatore accetti il carattere virgolette singole (') come delimitatore di stringa nei file di copia COBOL, utilizzare l'opzione del compilatore \APOST.

Nota: L'interfaccia di chiamata dinamica non è fornita in IBM MQ 9.0 o versioni successive.

Per utilizzare l'interfaccia di chiamata della procedura collegata, completare i seguenti passi.

Procedura

1. Creare un modulo utilizzando il compilatore **CRTCBLMOD** specificando il parametro:

   ```
   LINKLIT(*PRC)
   ```

2. Utilizzare il comando **CRTPGM** per creare l'oggetto del programma, specificando il parametro appropriato:

   Per applicazioni senza thread:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
   ```

   Per le applicazioni con thread:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
   ```

   Nota: Ad eccezione dei programmi creati utilizzando il compilatore ILE COBOL V4R4 e che contiene l'opzione THREAD (SERIALIZE) nell'istruzione PROCESS, i programmi COBOL non devono utilizzare le librerie IBM MQ con thread. Anche se un programma COBOL è stato reso thread safe in questo modo, prestare attenzione quando si progetta l'applicazione, poiché THREAD (SERIALIZE) forza la serializzazione delle procedure COBOL a livello di modulo e potrebbe influire sulle prestazioni generali.

   Per ulteriori informazioni, consultare *WebSphere Development Studio: ILE COBOL Programmer's Guide* e *WebSphere Development Studio: ILE COBOL Reference* .

   Per ulteriori informazioni sulla compilazione di un'applicazione CICS , consultare *CICS for IBM i Application Programming Guide*, SC41-5454.

IBM i *Preparazione dei programmi CICS in IBM i*

Informazioni sui passi richiesti durante la preparazione dei programmi CICS in IBM i.

Per creare un programma che include istruzioni EXEC CICS e chiamate MQI, attenersi alla seguente procedura:

1. Se necessario, preparare le mappe utilizzando il comando CRTCICSMAP.
2. Convertire i comandi EXEC CICS in istruzioni di lingua nativa. Utilizzare il comando CRTCICSC per un programma C. Utilizzare il comando CRTCICSCBL per un programma COBOL.

Includere `CICSOPT(*NOGEN)` nel comando CRTCICSC o CRTCICSCBL. Questa operazione arresta l'elaborazione per consentire di includere i programmi di servizio CICS e IBM MQ appropriati. Questo comando inserisce il codice, per impostazione predefinita, in QTEMP/QACYCICS.

3. Compilare il codice origine utilizzando il comando CRTCMOD (per un programma C) o il comando CRTCBLMOD (per un programma COBOL).
4. Utilizzare CRTPGM per collegare il codice compilato con i programmi di servizio CICS e IBM MQ appropriati. Ciò crea il programma eseguibile.

Di seguito è riportato un esempio di tale codice (viene compilato il programma di esempio CICS fornito):

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i *Preparazione dei programmi RPG in IBM i*

Se si utilizza IBM MQ for IBM i, è possibile scrivere le proprie applicazioni in RPG.

Per ulteriori informazioni, fare riferimento a “Codifica di programmi IBM MQ in RPG (soloIBM i)” a pagina 1060e a IBM i Application Programming Reference (ILE/RPG).

IBM i *Considerazioni sulla programmazione SQL per IBM i*

Informazioni sui passi richiesti durante la creazione di un'applicazione su IBM i utilizzando SQL.

Se il proprio programma contiene istruzioni EXEC SQL e chiamate MQI, attenersi alla seguente procedura:

1. Convertire i comandi EXEC SQL in istruzioni di linguaggio nativo. Utilizzare il comando CRTSQLCI per un programma C. Utilizzare il comando CRTSQLCBLI per un programma COBOL.

   Includere `OPTION(*NOGEN)` nel comando CRTSQLCI o CRTSQLCBLI. Questo arresta l'elaborazione per consentire all'utente di includere i programmi di servizio IBM MQ appropriati. Questo comando inserisce il codice, per impostazione predefinita, in QTEMP/QSQLTEMP.
2. Compilare il codice origine utilizzando il comando CRTCMOD (per un programma C) o il comando CRTCBLMOD (per un programma COBOL).
3. Utilizzare CRTPGM per collegare il codice compilato ai programmi di servizio IBM MQ appropriati. Ciò crea il programma eseguibile.

Di seguito è riportato un esempio di tale codice (che compila un programma, SQLTEST, nella libreria, SQLUSER):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux Creazione dell'applicazione procedurale su Linux

Queste informazioni descrivono le attività aggiuntive e le modifiche alle attività standard che è necessario eseguire durante la creazione di IBM MQ per l'esecuzione delle applicazioni Linux .

C e C++ sono supportati. Per informazioni sulla preparazione dei programmi C + +, consultare Utilizzo di C++.

Linux *Preparazione dei programmi C in Linux*

I programmi C precompilati vengono forniti nella directory *MQ_INSTALLATION_PATH*/samp/bin . Per creare un esempio dal codice sorgente, utilizzare il compilatore gcc .

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Lavorare nell'ambiente normale. Per ulteriori informazioni sulla programmazione di applicazioni a 64 bit, consultare Coding standards on 64 - bit platforms.

Collegamento di librerie

La seguente tabella elenca le librerie necessarie durante la preparazione dei programmi C su Linux.

- È necessario collegare i programmi con la libreria appropriata fornita da IBM MQ.

  In un ambiente non con thread, collegarsi solo a una delle seguenti librerie:

File di libreria	Tipo di programma / uscita
libmqm.so	Server per C
libmqic.so & libmqm.so	Client per C

  In un ambiente con thread, collegarsi solo a una delle seguenti librerie:

File di libreria	Tipo di programma / uscita
libmqm_r.so	Server per C
libmqic_r.so & libmqm_r.so	Client per C

  Nota:

  1. Non è possibile collegarsi a più di una libreria. In altre parole, non è possibile collegarsi contemporaneamente a una libreria con thread e non con thread.
  2. Se si sta scrivendo un servizio installabile (consultare Amministrazione IBM MQ per ulteriori informazioni), è necessario collegarsi alla libreria libmqmzf.so .
  3. Se si sta producendo un'applicazione per il coordinamento esterno mediante un gestore transazioni compatibile con XA, come IBM TXSeries Encina o BEA Tuxedo, è necessario collegarsi a libmqmxa.so (o libmqmxa64.so se il gestore transazioni considera il tipo 'long' come 64 bit) e alle librerie libmqz.so in un'applicazione non con thread e a libmqmxa_r.so (o libmqmxa64_r.so) e libmqz_r.so in un'applicazione con thread.
  4. È necessario collegare le librerie IBM MQ prima di qualsiasi altra libreria del prodotto.

Linux *Creazione di applicazioni a 31 bit*

Questo argomento contiene esempi di comandi utilizzati per creare programmi a 31 bit in vari ambienti.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Applicazione client C, a 31 bit, senza thread

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Applicazione client C, a 31 bit, con thread

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Applicazione server C, a 31 bit, senza thread

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Applicazione server C, a 31 bit, con thread

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Applicazione client C++, a 31 bit, senza thread

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Applicazione client C++, a 31 bit, con thread

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

Applicazione server C++, a 31 bit, senza thread

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Applicazione server C++, a 31 bit, con thread

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

Uscita client C, a 31 bit, senza thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Uscita client C, a 31 bit, con thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Uscita server C, a 31 bit, senza thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Uscita server C, a 31 bit, con thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Creazione di applicazioni a 32 bit*

Questo argomento contiene esempi dei comandi utilizzati per creare programmi a 32 - bit in vari ambienti.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Applicazione client C, a 32 bit, senza thread

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Applicazione client C, a 32 bit, con thread

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Applicazione server C, a 32 bit, senza thread

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Applicazione server C, 32 bit, con thread

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Applicazione client C + +, a 32 bit, senza thread

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Applicazione client C++, a 32 bit, con thread

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Applicazione server C + +, 32 bit, senza thread

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Applicazione server C + +, a 32 bit, con thread

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Uscita client C, a 32 bit, senza thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Uscita client C, 32 bit, con thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Uscita server C, 32 bit, senza thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Uscita server C, a 32 bit, con thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Creazione di applicazioni a 64 bit*

Questo argomento contiene esempi di comandi utilizzati per creare programmi a 64 bit in vari ambienti.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Applicazione client C, a 64 bit, senza thread

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

Applicazione client C, a 64 bit, con thread

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

Applicazione server C, 64 bit, senza thread

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

Applicazione server C, a 64 bit, con thread

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

Applicazione client C++, a 64 bit, senza thread

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Applicazione client C + +, 64 bit, con thread

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Applicazione server C++, a 64 bit, senza thread

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
```

```
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Applicazione server C++, a 64 bit, con thread

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Uscita client C, a 64 bit, senza thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

Uscita client C, a 64 bit, con thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

Uscita server C, 64 bit, senza thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

Uscita server C, 64 bit, con thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux *Preparazione dei programmi COBOL in Linux*

Scopri come preparare i programmi COBOL in Linux e preparare i programmi COBOL utilizzando IBM COBOL for Linux su x86 e Micro Focus COBOL.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

1. I copy book COBOL a 32 bit sono installati nella seguente directory:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   e i collegamenti simbolici vengono creati in:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. Su piattaforme a 64 bit, i copy book COBOL a 64 bit sono installati nella directory seguente:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. Nei seguenti esempi impostare COBCPY su:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

per applicazioni a 32 bit e:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

per applicazioni a 64 bit.

È necessario collegare il programma a uno dei seguenti:

File di libreria	Tipo di programma / uscita
libmqmcb.so	Server per COBOL
libmqicb.so	Client per COBOL
libmqmcb_r.so	Server per COBOL (applicazione con thread)
libmqicb_r.so	Client per COBOL (applicazione con thread)

Preparazione dei programmi COBOL utilizzando IBM COBOL for Linux on x86

I programmi COBOL di esempio vengono forniti con IBM MQ. Per compilare tale programma, immettere il comando appropriato dal seguente elenco:

Applicazione server senza thread a 32 bit

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

Applicazione client senza thread a 32 bit

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

Applicazione server con thread a 32 bit

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

Applicazione client con thread a 32 bit

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

Preparazione dei programmi COBOL utilizzando Micro Focus COBOL

Impostare le variabili di ambiente prima di compilare il proprio programma come segue:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Per compilare un programma COBOL a 32 bit, dove supportato, utilizzando Micro Focus COBOL, immettere:

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Per compilare un programma COBOL a 64 bit utilizzando Micro Focus COBOL, immettere:

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
```

```
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

dove `amqsput` è un programma di esempio

Per una descrizione delle variabili di ambiente necessarie, consultare la documentazione Micro Focus COBOL.

Windows Creazione dell'applicazione procedurale su Windows

Le pubblicazioni dei sistemi Windows descrivono come creare applicazioni eseguibili dai programmi scritti.

Questo argomento descrive le attività aggiuntive e le modifiche alle attività standard che è necessario eseguire quando si creano applicazioni IBM MQ for Windows da eseguire su sistemi Windows . Sono supportati i linguaggi di programmazione C, C + +, COBOL e Visual Basic. Per informazioni sulla preparazione dei programmi C + +, consultare Utilizzo di C++.

Le attività che devi eseguire per creare un'applicazione eseguibile utilizzando IBM MQ for Windows variano con il linguaggio di programmazione in cui è scritto il codice di origine. Oltre a codificare le chiamate MQI nel codice di origine, è necessario aggiungere le istruzioni di lingua appropriate per includere i file di inclusione IBM MQ for Windows per la lingua che si sta utilizzando. Acquisire familiarità con il contenuto di questi file. Consultare “File di definizione dati IBM MQ” a pagina 718 per una descrizione completa.

Windows *Creazione di applicazioni a 64 bit su Windows*

Le applicazioni a 32 bit e a 64 bit sono supportate su IBM MQ for Windows. L'eseguibile IBM MQ e i file della libreria vengono forniti in entrambi i formati a 32 bit e a 64 bit, utilizzare la versione appropriata a seconda dell'applicazione che si sta utilizzando.

File e librerie eseguibili

Le versioni a 32 bit e a 64 bit delle librerie IBM MQ vengono fornite nelle seguenti ubicazioni:

Tabella 148. Ubicazione delle librerie IBM MQ

Versione libreria	**Directory contenente i file della libreria**
32 bit	*MQ_INSTALLATION_PATH* `\Tools\Lib`
64 bit	*MQ_INSTALLATION_PATH* `\Tools\Lib64`

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Le applicazioni a 32 bit continuano a funzionare normalmente dopo la migrazione. I file a 32 bit si trovano nella stessa directory delle versioni precedenti del prodotto.

Se si desidera creare una versione a 64 bit, è necessario assicurarsi che il proprio ambiente sia configurato per utilizzare i file della libreria in *MQ_INSTALLATION_PATH* `\Tools\Lib64`. Assicurarsi che la variabile di ambiente LIB non sia impostata per la ricerca nella cartella contenente le librerie a 32 bit.

Windows *Preparazione dei programmi C in Windows*

Lavorare nel tipico ambiente Windows ; IBM MQ for Windows non richiede nulla di speciale.

Per ulteriori informazioni sulla programmazione di applicazioni a 64 bit, consultare Coding standards on 64 - bit platforms.

- Collegare i programmi con le librerie appropriate fornite da IBM MQ:

File di libreria	Tipo di programma / uscita
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	server per C a 32 bit
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	client per C a 32 bit
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	client per C a 32 bit con coordinamento della transazione
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	server per C a 64 bit
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	client per C a 64 bit
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	client per C a 64-bit con coordinamento delle operazioni

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Il seguente comando fornisce un esempio di compilazione del programma di esempio amqsget0 (utilizzando Microsoft Visual C++ compiler).

Per applicazioni a 32 bit:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

Per le applicazioni a 64 bit:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

Nota:

- Se si sta scrivendo un servizio installabile (consultare Amministrazione IBM MQ per ulteriori informazioni), è necessario collegarsi alla libreria mqmzf.lib .
- Se si sta producendo un'applicazione per il coordinamento esterno da un gestore transazioni compatibile con XA come IBM TXSeries Encina o BEA Tuxedo, è necessario collegarsi alla libreria mqmxa.lib o mqmxa.lib .
- Se si sta scrivendo un'uscita CICS , collegarsi alla libreria mqmcics4.lib .
- È necessario collegare le librerie IBM MQ prima di qualsiasi altra libreria del prodotto.

- Le DLL devono trovarsi nel percorso (PATH) specificato.
- Se si utilizzano caratteri minuscoli quando possibile, è possibile passare da IBM MQ for Windows a IBM MQ for AIX or Linux , dove è necessario utilizzare caratteri minuscoli.

Preparazione dei programmi CICS e Transaction Server

L'origine C di esempio per una transazione CICS IBM MQ è fornita da AMQSCIC0.CCS. È possibile crearlo utilizzando le funzionalità standard di CICS . Ad esempio, per TXSeries per Windows 2000:

1. Impostare la variabile di ambiente (immettere il codice seguente su una sola riga):

```
set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. Impostare la variabile di ambiente USERLIB:

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. Convertire, compilare e collegare il programma di esempio:

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Ciò è descritto nel manuale *Transaction Server for Windows NT Application Programming Guide (CICS) V4*.

Ulteriori informazioni sul supporto delle transazioni CICS sono disponibili in Amministrazione IBM MQ.

Windows *Preparazione dei programmi COBOL in Windows*

Utilizzare queste informazioni per preparare i programmi COBOL in Windowse preparare i programmi CICS e Transaction Server.

1. I copy book COBOL a 32 bit sono installati nella seguente directory: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook.
2. I copy book COBOL a 64 bit sono installati nella seguente directory: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64
3. Nei seguenti esempi impostare CopyBook su:

```
CopyBook
```

   per applicazioni a 32 bit e:

```
CopyBook64
```

   per applicazioni a 64 bit.

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Per preparare i programmi COBOL sui sistemi Windows , collegare il proprio programma a una delle seguenti librerie fornite da IBM MQ:

File di libreria	Tipo di programma o uscita
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	Server a 32 bit per Micro Focus COBOL
MQ_INSTALLATION_PATH \Tools\Lib\mqiccb	Client a 32 bit per Micro Focus COBOL
MQ_INSTALLATION_PATH \Tools\Lib64\mqmcb	Server a 64 bit per Micro Focus COBOL
MQ_INSTALLATION_PATH \Tools\Lib64\mqiccb	Client a 64 bit per Micro Focus COBOL

Quando si esegue un programma nell'ambiente del client MQI, assicurarsi che la libreria DOSCALLS venga visualizzata prima di qualsiasi libreria COBOL o IBM MQ .

Preparazione dei programmi COBOL utilizzando Micro Focus COBOL

Ricollegare eventuali programmi IBM MQ Micro Focus COBOL a 32 bit esistenti utilizzando `mqmcb.lib` o `mqiccb.lib`, piuttosto che le librerie `mqmcbb` e `mqiccbb` .

Per compilare, ad esempio, il programma di esempio `amq0put0`, utilizzando Micro Focus COBOL:

1. Impostare la variabile di ambiente COBCPY in modo che punti ai copybook COBOL IBM MQ (immettere il seguente codice su una riga):

   ```
   set COBCPY= MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook
   ```

2. Compilare il programma per fornire un file oggetto:

   ```
   cobol amq0put0 LITLINK
   ```

3. Collegare il file oggetto al sistema di runtime.
   - Impostare la variabile di ambiente LIB per puntare alle librerie COBOL del compilatore.
   - Collegare il file oggetto da utilizzare sul server IBM MQ :

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - In alternativa, collegare il file oggetto da utilizzare sul client IBM MQ :

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

Preparazione dei programmi CICS e Transaction Server

Per compilare e collegare un programma TXSeries per Windows NT, V5.1 utilizzando IBM VisualAge COBOL:

1. Impostare la variabile di ambiente (immettere il codice seguente su una sola riga):

   ```
   set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
   Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
   ```

2. Impostare la variabile di ambiente USERLIB:

   ```
   set USERLIB=MQMCBB.LIB
   ```

3. Tradurre, compilare e collegare il programma:

   ```
   cicstcl -l IBMCOB myprog.ccp
   ```

Ciò è descritto nel manuale *Transaction Server per Windows NT, V4 Application Programming* Guide.

Per compilare e collegare un programma CICS per Windows V5 utilizzando Micro Focus COBOL:

- Impostare la variabile INCLUDE:

  ```
  set
  INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
          drive:\opt\cics\include;%INCLUDE%
  ```

- Impostare la variabile di ambiente COBCPY:

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- Impostare le opzioni COBOL:
  - `set`
  - `COBOPTS=/LITLINK /NOTRUNC`

ed eseguire il seguente codice:

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows *Preparazione dei programmi Visual Basic in Windows*

Informazioni da considerare quando si utilizzano programmi Microsoft Visual Basic su Windows.

Deprecated Da IBM MQ 9.0, il supporto per Microsoft Visual Basic 6.0 è obsoleto. Le classi IBM MQ per .NET sono la tecnologia sostitutiva consigliata. Per ulteriori informazioni, consultare Sviluppo di applicazioni .NET.

Nota: Non vengono fornite versioni a 64 bit dei file del modulo Visual Basic .

Per preparare i programmi Visual Basic su Windows:

1. Creare un nuovo progetto.
2. Aggiungere il file del modulo fornito, CMQB.BAS, al progetto.
3. Aggiungere altri file del modulo forniti, se necessario:
   - CMQBB.BAS: Supporto MQAI
   - CMQCFB.BAS: Supporto PCF
   - CMQXB.BAS: Supporto uscite canale
   - CMQPSB.BAS: Pubblicazione / sottoscrizione

Consultare “Codifica in Visual Basic” a pagina 1055 per informazioni sull'utilizzo della chiamata MQCONNXAny dall'interno di Visual Basic.

Richiamare la procedura MQ_SETDEFAULTS prima di effettuare qualsiasi chiamata MQI nel codice del progetto. Questa procedura imposta le strutture predefinite richieste dalle chiamate MQI.

Specificare se si sta creando un client o un server IBM MQ , prima di compilare o eseguire il progetto, impostando la variabile di compilazione condizionale *MqType*. Impostare *MqType* in un progetto Visual Basic su 1 per un server o 2 per un client come segue:

1. Selezionare il menu Progetto.
2. Selezionare *Name* Proprietà (dove *Name* è il nome del progetto corrente).
3. Selezionare la scheda Crea nella finestra di dialogo.
4. Nel campo Argomenti di compilazione condizionale, immettere quanto segue per un server:

   ```
   MqType=1
   ```

   o questo per un client:

   ```
   MqType=2
   ```

Concetti correlati

“Codifica in Visual Basic” a pagina 1055

Informazioni da considerare quando si codificano i programmi IBM MQ in Microsoft Visual Basic. Visual Basic è supportata solo su Windows.

Riferimenti correlati

“Collegamento delle applicazioni Visual Basic con il codice IBM MQ MQI client” a pagina 922
È possibile collegare le applicazioni Microsoft Visual Basic con il codice di IBM MQ MQI client su Windows.

Windows *Uscita di sicurezza SSPI*

IBM MQ for Windows fornisce un'uscita di sicurezza sia per IBM MQ MQI client che per il server IBM MQ . Si tratta di un programma di uscita del canale che fornisce l'autenticazione per i canali IBM MQ utilizzando SSPI (Security Services Programming Interface). SSPI fornisce le funzionalità di sicurezza integrate dei sistemi Windows .

I pacchetti di sicurezza vengono caricati da security.dll o secur32.dll. Queste DLL vengono fornite con il sistema operativo.

L'autenticazione unidirezionale viene fornita utilizzando i servizi di autenticazione NTLM. L'autenticazione bidirezionale viene fornita utilizzando i servizi di autenticazione Kerberos .

Il programma di uscita di sicurezza viene fornito in formato origine e oggetto. È possibile utilizzare il codice oggetto così com' è oppure è possibile utilizzare il codice origine come punto di partenza per creare i propri programmi di uscita utente.

Consultare anche “Utilizzo dell'uscita di sicurezza SSPI su Windows” a pagina 1137.

Introduzione alle uscite di sicurezza

Un'uscita di sicurezza forma una connessione sicura tra due programmi di uscita di sicurezza, in cui un programma funga da MCA (message channel agent) mittente e da MCA ricevente.

Il programma che avvia la connessione sicura, ovvero il primo programma che acquisisce il controllo dopo aver stabilito la sessione MCA, è noto come *iniziatore del contesto*. Il programma partner è noto come *acceptor di contesto*.

La seguente tabella mostra alcuni tipi di canale che sono iniziatori di contesto e i relativi accettatori di contesto associati.

Tabella 149. Iniziatori di contesto e relativi accettatori di contesto associati

Iniziatore contesto	**Acceptor contesto**
CLNTCONN MQCHT	SVRCONN MQCHT
MQCH_DESTINATARIO	MQCH_SENDER
CLUSRCVR MQCHT	MQCHT_CLUSSDR

Il programma di uscita di sicurezza ha due punti di ingresso:

- **SC_NTLM**

  Utilizza i servizi di autenticazione NTLM, che forniscono l'autenticazione unidirezionale. NTLM consente ai server di verificare le identità dei propri client. Non consente ai client di verificare l'identità di un server o a un server di verificare l'identità di un altro. L'autenticazione NTLM è stata progettata per un ambiente di rete in cui si presume che i server siano originali.

- **SCY_KERBEROS**

  Utilizza i servizi di autenticazione reciproca Kerberos . Il protocollo Kerberos non presuppone che i server in un ambiente di rete siano originali. Le parti alle due estremità di una connessione di rete possono verificare l'identità dell'altra parte. In altre parole, i server possono verificare l'identit ... dei client e di altri server e i client possono verificare l'identit ... di un server.

Cosa fa l'uscita di sicurezza

Questo argomento descrive le operazioni dei programmi di uscita canale SSPI.

I programmi di uscita canale forniti forniscono l'autenticazione unidirezionale o bidirezionale (reciproca) di un sistema partner quando viene stabilita una sessione. Per un particolare canale, ogni programma di uscita ha un *principal* associato (simile a un ID utente, consultare “Controllo accessi IBM MQ e principal Windows” a pagina 1024). Una connessione tra due programmi di uscita è un'associazione tra i due principal.

Una volta stabilita la sessione sottostante, viene stabilita una connessione sicura tra due programmi di uscita di sicurezza (uno per l'MCA mittente e uno per l'MCA ricevente). La sequenza delle operazioni è la seguente:

1. Ogni programma è associato ad un particolare principal, ad esempio come risultato di un'operazione di login esplicita.
2. L'iniziatore del contesto richiede una connessione sicura con il partner dal pacchetto di sicurezza (per Kerberos, il partner indicato) e riceve un token (denominato token1). Il token viene inviato, utilizzando la sessione sottostante già stabilita, al programma partner.
3. Il programma partner (il programma di accettazione del contesto) passa token1 al pacchetto di sicurezza, che verifica che l'iniziatore del contesto sia autentico. Per NTLM, la connessione è ora stabilita.
4. Per l'uscita di sicurezza fornita da Kerberos(ovvero, per l'autenticazione reciproca), il pacchetto di sicurezza genera anche un secondo token (denominato token2), che l'acceptor del contesto restituisce all'iniziatore del contesto utilizzando la sessione sottostante.
5. L'iniziatore di contesto utilizza token2 per verificare che l'acceptor di contesto sia autentico.
6. A questo punto, se entrambe le applicazioni sono soddisfatte dell'autenticità del token del partner, viene stabilita la connessione sicura (autenticata).

Controllo accessi IBM MQ e principal Windows

Il controllo accessi fornito da IBM MQ si basa sull'utente e sul gruppo. L'autenticazione fornita da Windows si basa sui principal, come utente e SPN (servicePrincipalName). Nel caso del nome servicePrincipal, molti di questi potrebbero essere associati a un singolo utente.

L'uscita di sicurezza SSPI utilizza i principal Windows pertinenti per l'autenticazione. Se l'autenticazione di Windows ha esito positivo, l'uscita passa l'ID utente associato al principal Windows a IBM MQ per il controllo accessi.

I principal Windows rilevanti per l'autenticazione variano a seconda del tipo di autenticazione utilizzato.

- Per l'autenticazione NTLM, il principal Windows per l'iniziatore del contesto è l'ID utente associato al processo in esecuzione. Poiché questa autenticazione è un modo, il principal associato a Context Acceptor è irrilevante.
- Per l'autenticazione Kerberos , sui canali CLNTCONN, il principal Windows è l'ID utente associato con il processo in esecuzione. Altrimenti, il principal Windows è il nome servicePrincipalformato aggiungendo il prefisso seguente al nome QueueManager.

  ```
  ibmMQSeries/
  ```

z/OS Creazione dell'applicazione procedurale su z/OS

Le pubblicazioni CICS, IMSe z/OS descrivono come creare applicazioni in esecuzione in questi ambienti.

Questa raccolta di argomenti descrive le attività aggiuntive e le modifiche alle attività standard che è necessario eseguire quando si creano le applicazioni IBM MQ for z/OS per questi ambienti. I linguaggi di programmazione COBOL, C, C + +, Assembler e PL/I sono supportati. (Per informazioni sulla creazione di applicazioni C++, vedere Utilizzo di C++.)

Le attività che è necessario eseguire per creare un'applicazione eseguibile IBM MQ for z/OS dipendono sia dal linguaggio di programmazione in cui è scritto il programma, sia dall'ambiente in cui verrà eseguita l'applicazione.

Oltre a codificare le chiamate MQI nel programma, aggiungere le istruzioni di linguaggio appropriate per includere il file di definizione dei dati IBM MQ for z/OS per il linguaggio che si sta utilizzando. Acquisire familiarità con il contenuto di questi file. Consultare "File di definizione dati IBM MQ" a pagina 718 per una descrizione completa.

Nota

Il nome **thlqual** è il qualificatore di alto livello della libreria di installazione su z/OS.

z/OS *Preparazione del programma per l'esecuzione*

Dopo aver scritto il programma per l'applicazione IBM MQ per creare un'applicazione eseguibile, è necessario compilarlo o assemblarlo, quindi modificare il codice oggetto risultante con il programma stub fornito da IBM MQ for z/OS per ciascun ambiente supportato.

La modalità di preparazione del programma dipende dall'ambiente (batch, CICS, IMS(BMP o MPP), Linux o z/OS UNIX System Services) in cui viene eseguita l'applicazione e dalla struttura dei dataset sull'installazione di z/OS .

"Richiamo dinamico dello stub IBM MQ" a pagina 1031 descrive un metodo alternativo per effettuare chiamate MQI nei programmi in modo che non sia necessario modificare tramite link uno stub IBM MQ . Questo metodo non è disponibile per tutte le lingue e gli ambienti.

Non collegare - modificare un livello superiore di programma stub rispetto alla versione di IBM MQ for z/OS su cui è in esecuzione il proprio programma. Ad esempio, un programma in esecuzione su MQSeries per OS/390, V5.2 non deve essere modificato dal link con un programma stub fornito con IBM MQ for z/OS V7.

z/OS *Creazione di applicazioni C a 64 bit*

In z/OS, le applicazioni C a 64 bit vengono create utilizzando il compilatore LP64 e le opzioni binder. Il file di intestazione IBM MQ for z/OS *cmqc.h* riconosce quando questa opzione viene fornita al compilatore e genera strutture e tipi di dati IBM MQ appropriati per l'operazione a 64 bit.

Il codice C creato con questa opzione deve essere creato per utilizzare le DLL (dynamic - link libraries) appropriate per la semantica di coordinamento richiesta. Per ottenere ciò, si collega il codice compilato con il gruppo laterale appropriato definito nella seguente tabella:

Tabella 150. Nome del ponte laterale richiesto per ogni semantica di coordinamento

coordination	**Nome gruppo laterale**
MQI commit a fase singola	CSQBMQ2X
Commit a due fasi con coordinamento RRS, utilizzando i verbi RRS	CSQBRR2X
Commit a due fasi con coordinamento RRS, utilizzando i verbi MQI	CSQBRI2X

Nota: Per le applicazioni C a 31 bit, è anche possibile impostare le opzioni del compilatore per l'interfaccia chiamante (Language Environment o XPLINK), come descritto in "Creazione di applicazioni batch z/OS utilizzando XPLINK (31 - bit Language Environment)" a pagina 1027. Per applicazioni C a 64 - bit non si specifica l'interfaccia chiamante, poiché l'unico collegamento supportato è XPLINK.

Utilizzare la procedura EDCQCB JCL, fornita con *z/OS XL C/C++*, per creare un programma IBM MQ di commit a fase singola come un lavoro batch, nel modo seguente:

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
```

```
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',            < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',            < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//        DD
//        DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

Per creare un programma coordinato RRS in z/OS UNIX System Services, compilare e collegarsi come segue:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Creazione di applicazioni batch z/OS*

Scopri come creare applicazioni batch z/OS e i passi da considerare in questo caso.

Per creare un'applicazione per IBM MQ for z/OS eseguita in batch z/OS , creare un JCL (job control language) che esegua queste attività:

1. Compilare (o assemblare) il programma per produrre il codice oggetto. Il JCL per la compilazione deve includere istruzioni SYSLIB che rendono i file di definizione dei dati del prodotto disponibili per il compilatore. Le definizioni dei dati vengono fornite nelle seguenti librerie IBM MQ for z/OS :
   - Per COBOL, **thlqual**.SCSQCOBC
   - Per il linguaggio assembler, **thlqual**.SCSQMACS
   - Per C, **thlqual**.SCSQC370
   - Per PL/I, **thlqual**.SCSQPLIC
2. Per un'applicazione C, precollegare il codice oggetto creato nel passo “1” a pagina 1026.
3. Per le applicazioni PL/I, utilizzare l'opzione compilatore EXTRN (SHORT).
4. Modificare tramite link il codice oggetto creato nel passo “1” a pagina 1026 (o nel passo “2” a pagina 1026 per un'applicazione C) per produrre un modulo di caricamento. Quando si collega - modifica il codice, è necessario includere uno dei programmi stub batch IBM MQ for z/OS (CSQBSTUB o uno dei programmi stub RRS: CSQBRRSI o CSQBRSTB).

   CSQBSTUB
   commit a fase singola fornito da IBM MQ for z/OS

   CSQBRRSI
   commit a due fasi fornito da RRS utilizzando MQI

   CSQBRSTB
   commit a due fasi fornito direttamente da RRS

   Note:

   a. Se si utilizza CSQBRSTB, è necessario anche collegare - modificare l'applicazione con ATRSCSS da SYS1.CSSLIB. Figura 113 a pagina 1027 e Figura 114 a pagina 1027 mostrano i frammenti di JCL per eseguire questa operazione. Gli stub sono indipendenti dalla lingua e vengono forniti nella libreria **thlqual**.SCSQLOAD.

   b. Se l'applicazione viene eseguita in Language Environment, è necessario assicurarsi di eseguire il link - edit con la DLL Language Environment, come descritto in “Creazione di applicazioni batch z/OS utilizzando XPLINK (31 - bit Language Environment)” a pagina 1027.
5. Memorizzare il modulo di caricamento in una libreria di caricamento dell'applicazione.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Figura 113. Frammenti di JCL per collegare - modificare il modulo oggetto nell'ambiente batch, utilizzando il commit a fase singola

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

Figura 114. Frammenti di JCL per il link - modificare il modulo oggetto nell'ambiente batch, utilizzando il commit a due fasi

Per eseguire un programma batch o RRS, è necessario includere le librerie **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD nella concatenazione di dataset STEPLIB o JOBLIB.

Per eseguire un programma TSO, è necessario includere le librerie **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD nella STEPLIB utilizzata dalla sessione TSO.

Per eseguire un programma batch dalla shell z/OS UNIX System Services , aggiungere le librerie **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD alla specifica STEPLIB nel proprio $HOME? .profile come segue:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Creazione di applicazioni batch z/OS utilizzando XPLINK (31 - bit Language Environment)*

IBM MQ for z/OS fornisce una serie di DLL (dynamic link libraries) che devono essere utilizzate quando si collegano - modificano le applicazioni.

Ci sono due varianti delle librerie che consentono all'applicazione di utilizzare una delle seguenti interfacce di chiamata:

- L'interfaccia di chiamata di Language Environment a 31 bit.
- L'interfaccia di chiamata XPLINK a 31 bit. z/OS XPLINK è una convenzione di chiamata ad alte prestazioni disponibile per le applicazioni C. Consultare XPLINK | NOXPLINK nella documentazione di z/OS 2.2 .

Per utilizzare le DLL, l'applicazione è collegata o collegata rispetto ai cosiddetti *sidedeck*, invece degli stub forniti con le versioni precedenti. I sidedeck si trovano nella libreria SCSQDEFS (invece della libreria SCSQLOAD).

Tabella 151. Varianti di DLL (Dynamic Link Library)			
Commit	**DLL (Language Environment) a 31 - bit**	**DLL XPLINK a 31 bit**	**Nome stub equivalente**
Librerie MQI commit a 1 fase	CSQBMQ1	CSQBMQ1X	CSQBSTUB
Commit a 2 fasi con coordinamento RRS utilizzando verbi di controllo transazione RRS	CSQBRR1	CSQBRR1X	CSQBRSTB
Commit a 2 fasi con coordinazione RRS utilizzando i verbi di controllo della transazione MQI	CSQBRI1	CSQBRI1X	CSQBRRSI

Nota: Tutti i sidedeck contengono una definizione del punto di ingresso di conversione dati, MQXCNVC, precedentemente risolto includendo CSQASTUB.

Problemi comuni:

- Il seguente messaggio viene visualizzato nel log del lavoro se l'applicazione utilizza il messaggio asincrono (chiamate MQCB, MQCTL o MQSUB) e l'interfaccia DLL precedente non viene utilizzata:

  ```
  CSQB001E I programmi dell'ambiente di linguaggio in esecuzione in batch z/OS o z/OS UNIX System
  Services devono utilizzare l'interfaccia DLL per IBM MQ
  ```

  Soluzione: rigenerare l'applicazione utilizzando i sidedeck invece degli stub come descritto in precedenza.

- Al momento della creazione del programma, viene visualizzato il seguente messaggio

  ```
  IEW2469E Gli attributi di un riferimento a MQAPI - NAME dalla sezione your - code non
  corrispondono agli attributi di:
  il simbolo di destinazione
  ```

  Causa: significa che il programma XPLINK è stato compilato con V701 (o versione successiva) di cmqc.h, ma non è stato eseguito il bind con i sidedeck.

  Soluzione: modificare il file di build del programma per eseguire il bind rispetto al sidedeck appropriato da SCSQDEFS invece di uno stub da SCSQLOAD

Il seguente JCL di esempio dimostra come è possibile compilare e modificare il collegamento di un programma C per utilizzare l'interfaccia di chiamata DLL dell'ambiente di linguaggio a 31 bit:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

Nota: La compilazione utilizza l'opzione **DLL** . La modifica del link utilizza l'opzione **DYNAM=DLL** e fa riferimento alla libreria **CSQBMQ1** .

Il seguente JCL di esempio dimostra come è possibile compilare e modificare il collegamento di un programma C per utilizzare l'interfaccia di chiamata DLL XPLINK a 31 bit:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

Nota: La compilazione utilizza le opzioni **XPLINK** e **DLL** . La modifica del link utilizza l'opzione **DYNAM=DLL** e fa riferimento alla libreria **CSQBMQ1X** .

Assicurarsi di aggiungere l'opzione di compilazione DLL a ciascun programma nel modulo. I messaggi come IEW2456E 9207 SYMBOL CSQ1BAK UNRISOLTO sono un'indicazione che è necessario verificare che tutti i programmi siano stati compilati con l'opzione DLL.

z/OS *Creazione di applicazioni CICS in z/OS*

Utilizzare queste informazioni quando si creano applicazioni CICS in z/OS.

Per creare un'applicazione per IBM MQ for z/OS eseguita in CICS, è necessario:

- Tradurre i comandi CICS nel proprio programma nella lingua in cui è scritto il resto del programma.
- Compilare o assemblare l'output dal convertitore per produrre il codice oggetto.
  - Per i programmi PL/I, utilizzare l'opzione compilatore EXTRN (SHORT).
  - Per le applicazioni C, se l'applicazione non utilizza XPLINK, utilizzare l'opzione del compilatore DEFINE (MQ_OS_LINKAGE=1).
- Link - modifica il codice oggetto per creare un modulo di caricamento.

CICS fornisce una procedura per eseguire questi passi in sequenza per ciascuno dei linguaggi di programmazione supportati.

- Per CICS Transaction Server per z/OS, *CICS Transaction Server per z/OS System Definition Guide* descrive come utilizzare queste procedure e *CICS/ESA Application Programming Guide* fornisce ulteriori informazioni sul processo di traduzione.

È necessario includere:

- Nell'istruzione SYSLIB dello stage di compilazione (o di assemblaggio), le istruzioni che rendono i file di definizione dei dati del prodotto disponibili per il compilatore. Le definizioni dei dati vengono fornite nelle seguenti librerie IBM MQ for z/OS :
  - Per COBOL, **thlqual**.SCSQCOBC
  - Per il linguaggio assembler, **thlqual**.SCSQMACS
  - Per C, **thlqual**.SCSQC370
  - Per PL/I, **thlqual**.SCSQPLIC

- Nel JCL di modifica del collegamento, il programma stub IBM MQ for z/OS CICS (CSQCSTUB). Figura 115 a pagina 1030 mostra i frammenti di codice JCL per eseguire questa operazione. Lo stub è indipendente dalla lingua ed è fornito nella libreria **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Figura 115. Frammenti di JCL per collegare - modificare il modulo oggetto nell'ambiente CICS

- Per le versioni CICS successive a CICS TS 3.2oppure, se si desidera utilizzare le API della proprietà del messaggio IBM MQ o le API IBM MQ MQCB, MQCTL, MQSTAT, MQSUB o MQSUBR, è necessario modificare il codice oggetto con lo stub fornito da CICS , DFHMQSTB e non con CSQCSTUB fornito da IBM MQ . Per ulteriori informazioni sulla creazione di programmi IBM MQ per CICS, consultare API stub program to access IBM MQ MQI calls nella documentazione del prodotto CICS .

Una volta completate queste operazioni, memorizzare il modulo di caricamento in una libreria di caricamento dell'applicazione e definire il programma in CICS nel solito modo.

Prima di eseguire un programma CICS , l'amministratore di sistema deve definirlo in CICS come programma e transazione IBM MQ , quindi è possibile eseguirlo nel modo tipico.

z/OS *Creazione di applicazioni IMS (BMP o MPP)*

Utilizzare queste informazioni quando si creano applicazioni IMS (BMP o MPP).

Se si stanno creando programmi DL/I batch, consultare “Creazione di applicazioni batch z/OS” a pagina 1026. Per creare altre applicazioni in esecuzione in IMS (come BMP o MPP), creare JCL che esegua queste attività:

1. Compilare (o assemblare) il programma per produrre il codice oggetto. Il JCL per la compilazione deve includere istruzioni SYSLIB che rendono i file di definizione dei dati del prodotto disponibili per il compilatore. Le definizioni dei dati vengono fornite nelle seguenti librerie IBM MQ for z/OS :
   - Per COBOL, **thlqual**.SCSQCOBC
   - Per il linguaggio assembler, **thlqual**.SCSQMACS
   - Per C, **thlqual**.SCSQC370
   - Per PL/I, **thlqual**.SCSQPLIC
2. Per un'applicazione C, precollegare il modulo oggetto creato nel passo “1” a pagina 1030.
3. Per i programmi PL/I, utilizzare l'opzione compilatore EXTRN (SHORT).
4. Per un'applicazione C, se l'applicazione non utilizza XPLINK, utilizzare l'opzione del compilatore DEFINE (MQ_OS_LINKAGE=1).
5. Modificare tramite link il codice oggetto creato nel passo “1” a pagina 1030 (o nel passo “2” a pagina 1030 per un'applicazione C/370) per produrre un modulo di caricamento:
   a. Includere il modulo dell'interfaccia di lingua IMS (DFSLI000).
   b. Includere il programma stub di IBM MQ for z/OS IMS (CSQQSTUB). Figura 116 a pagina 1031 mostra i frammenti di JCL per eseguire questa operazione. Lo stub è indipendente dalla lingua ed è fornito nella libreria **thlqual**.SCSQLOAD.

      Nota: Se si sta utilizzando COBOL, selezionare l'opzione del compilatore NODYNAM per consentire all'editor di collegamento di risolvere i riferimenti a CSQQSTUB a meno che non si intenda utilizzare il collegamento dinamico come descritto in “Richiamo dinamico dello stub IBM MQ” a pagina 1031.
6. Memorizzare il modulo di caricamento in una libreria di caricamento dell'applicazione.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Figura 116. Frammenti di JCL per collegare - modificare il modulo oggetto nell'ambiente IMS

Prima di eseguire un programma IMS , l'amministratore di sistema deve definirlo in IMS come programma e transazione IBM MQ : è quindi possibile eseguirlo nel modo tipico.

z/OS *Creazione di applicazioni z/OS UNIX System Services*

Utilizzare queste informazioni quando si creano applicazioni z/OS UNIX System Services .

Per creare un'applicazione C per IBM MQ for z/OS eseguita in z/OS UNIX System Services, compilare e collegare l'applicazione nel modo seguente:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

dove **thlqual** è il qualificatore di alto livello utilizzato dall'installazione.

Per eseguire il programma C, è necessario aggiungere quanto riportato di seguito al file `.profile` ; deve trovarsi nella directory root:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Tenere presente che è necessario uscire da z/OS UNIX System Servicese immettere nuovamente z/OS UNIX System Services per consentire il riconoscimento della modifica.

Se si desidera eseguire più shell, aggiungere la parola export all'inizio della riga, ovvero:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Una volta completato correttamente, è possibile collegare le chiamate CSQBSTUB e problema IBM MQ .

“Richiamo dinamico dello stub IBM MQ” a pagina 1031 descrive un metodo alternativo per effettuare chiamate MQI nei programmi in modo che non sia necessario modificare tramite link uno stub IBM MQ . Questo metodo non è disponibile per tutte le lingue e gli ambienti.

Non collegare - modificare un livello superiore di programma stub rispetto alla versione di IBM MQ for z/OS su cui è in esecuzione il proprio programma. Ad esempio, un programma in esecuzione su IBM WebSphere MQ for z/OS 7.1 non deve essere modificato tramite link con un programma stub fornito con IBM MQ for z/OS 8.0.

z/OS *Richiamo dinamico dello stub IBM MQ*

Invece di collegare il programma stub IBM MQ con il codice oggetto, è possibile richiamare dinamicamente lo stub dall'interno del programma.

È possibile effettuare questa operazione negli ambienti batch, IMSe CICS . Questa funzione non è supportata nell'ambiente RRS. Se il programma applicativo utilizza RRS per coordinare gli aggiornamenti, consultare “Considerazioni RRS” a pagina 1036.

Tuttavia, questo metodo:

- Aumenta la complessità dei programmi
- Aumenta la memoria richiesta dai programmi al momento dell'esecuzione

- Riduce le prestazioni dei programmi
- Indica che non è possibile utilizzare gli stessi programmi in altri ambienti

Se si richiama lo stub in modo dinamico, il programma stub appropriato e i relativi alias devono essere disponibili al momento dell'esecuzione. Per garantire ciò, includere il dataset IBM MQ for z/OS SCSQLOAD:

- Per batch e IMS, nella concatenazione STEPLIB del JCL.
- Per CICS, nella concatenazione DFHRPL CICS .

Per IMS, accertarsi che la libreria contenente lo stub dinamico (creato come descritto nelle informazioni sull'installazione dell'adattatore IMS in Impostazione dell'adattatore IMS) è più avanti del dataset SCSQLOAD nella concatenazione STEPLIB del JCL della regione.

Utilizzare i nomi mostrati in Tabella 152 a pagina 1032 quando si richiama lo stub dinamicamente. In PL/I, dichiarare solo i nomi di chiamata utilizzati nel programma.

Tabella 152. Nomi di chiamata per il collegamento dinamico

chiamata MQI	Nomi di chiamate dinamiche batch (non RRS)	Nomi di chiamate dinamiche CICS	Nomi di chiamate dinamiche IMS
MQBACK	CSQBBACK	non supportato	Non supportato
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Non supportato
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	non supportato	Non supportato
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Non supportato
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGET	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB

Tabella 152. Nomi di chiamata per il collegamento dinamico (Continua)			
chiamata MQI	**Nomi di chiamate dinamiche batch (non RRS)**	**Nomi di chiamate dinamiche CICS**	**Nomi di chiamate dinamiche IMS**
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Nota: 1. Queste chiamate API sono disponibili solo quando si utilizza CICS TS 3.2 o versione successiva e deve essere utilizzato CSQCSTUB fornito con CICS . Per CICS TS 3.2, è necessario applicare l'APAR PK66866 . Per CICS TS 4.1, è necessario applicare l'APAR PK89844 .

Per esempi su come utilizzare questa tecnica, vedere le seguenti figure:

- Batch e COBOL: consultare Figura 117 a pagina 1033
- CICS e COBOL: consultare Figura 118 a pagina 1033
- IMS e COBOL: consultare Figura 119 a pagina 1034
- Batch e assembler: consultare Figura 120 a pagina 1034
- CICS e assembler: consultare Figura 121 a pagina 1034
- IMS e assembler: consultare Figura 122 a pagina 1034
- Batch e C: Figura 123 a pagina 1035
- CICS e C: consultare Figura 124 a pagina 1035
- IMS e C: consultare Figura 125 a pagina 1035
- Batch e PL/I: consultare Figura 126 a pagina 1035
- IMS e PL/I: consultare Figura 127 a pagina 1036

```
...
      WORKING-STORAGE SECTION.
...
         05 WS-MQOPEN                    PIC X(8) VALUE 'CSQBOPEN'.
...
      PROCEDURE DIVISION.
...
          CALL WS-MQOPEN WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
```

Figura 117. Collegamento dinamico utilizzando COBOL nell'ambiente batch

```
...
      WORKING-STORAGE SECTION.
...
         05 WS-MQOPEN                    PIC X(8) VALUE 'CSQCOPEN'.
...
      PROCEDURE DIVISION.
...
          CALL WS-MQOPEN WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
```

Figura 118. Collegamento dinamico utilizzando COBOL nell'ambiente CICS

```
...
      WORKING-STORAGE SECTION.
...
         05 WS-MQOPEN                    PIC X(8) VALUE 'MQOPEN'.
...
      PROCEDURE DIVISION.
...
          CALL WS-MQOPEN WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
     * ------------------------------------------------------------ *
     *
     *    If the compilation option 'DYNAM' is specified
     *    then you may code the MQ calls as follows
     *
     * ------------------------------------------------------------ *
...
          CALL 'MQOPEN'  WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
```

Figura 119. Collegamento dinamico utilizzando COBOL nell'ambiente IMS

```
...
        LOAD   EP=CSQBOPEN
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        DELETE EP=CSQBOPEN
...
```

Figura 120. Collegamento dinamico mediante linguaggio assembly nell'ambiente batch

```
...
        EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Figura 121. Collegamento dinamico utilizzando il linguaggio di assemblaggio nell'ambiente CICS

```
...
        LOAD   EP=MQOPEN
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        DELETE EP=MQOPEN
...
```

Figura 122. Collegamento dinamico utilizzando il linguaggio di assemblaggio nell'ambiente IMS

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 123. Collegamento dinamico mediante linguaggio C nell'ambiente batch

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 124. Collegamento dinamico mediante linguaggio C nell'ambiente CICS

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 125. Collegamento dinamico mediante linguaggio C nell'ambiente IMS

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Figura 126. Collegamento dinamico mediante PL/I nell'ambiente batch

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Figura 127. Collegamento dinamico mediante PL/I nell'ambiente di IMS

z/OS *Considerazioni RRS*

Utilizzare queste informazioni se il programma applicativo utilizza RRS per coordinare gli aggiornamenti.

IBM MQ fornisce due diversi stub per programmi batch che richiedono il coordinamento RRS - consultare “L'adattatore batch RRS” a pagina 893. La differenza di funzionamento delle chiamate API successive è determinata all'ora MQCONN dall'adattatore batch dalle informazioni trasmesse dalla routine stub sull'API MQCONN o MQCONNX. Ciò significa che le chiamate API dinamiche sono disponibili per i programmi batch che richiedono il coordinamento RRS, purché la connessione iniziale a IBM MQ sia stata effettuata utilizzando lo stub appropriato. Il seguente esempio illustra quanto segue:

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
                               W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS *Debug dei programmi*

Utilizzare queste informazioni per informazioni sul debug dei programmi TSO e CICS e per informazioni dettagliate sulla traccia CICS .

I principali supporti per il debug dei programmi applicativi IBM MQ for z/OS sono i codici di errore restituiti da ogni chiamata API. Per un elenco di tali elementi, incluse le idee per un'azione correttiva, consultare:

- IBM MQ for z/OS , completamento, e codici di errore per IBM MQ for z/OS
- Messaggi e codici di errore per tutte le altre piattaforme IBM MQ

Questo argomento suggerisce anche altri strumenti di debug da utilizzare in ambienti particolari.

Debug dei programmi TSO

I seguenti strumenti di debug interattivi sono disponibili per i programmi TSO:

- strumento TEST
- Strumento di debug interattivo VS COBOL II
- Strumento di debug interattivo INSPECT per programmi C e PL/I

Debug dei programmi CICS

È possibile utilizzare CEDF (CICS Execution Diagnostic Facility) per testare i programmi CICS in modo interattivo senza dover modificare il programma o la procedura di preparazione del programma.

Per ulteriori informazioni su EDF, consultare *CICS Transaction Server for z/OS CICS Application Programming Guide*.

CICS traccia

Probabilmente sarà utile anche utilizzare la transazione CICS Controllo traccia (CETR) per controllare l'attività di traccia CICS .

Per ulteriori informazioni su CETR, consultare il manuale *CICS Transaction Server for z/OS CICS-Supplied Transactions* .

Per determinare se la traccia CICS è attiva, visualizzare lo stato della connessione utilizzando il pannello CKQC. Questo pannello mostra anche il numero di traccia.

Per interpretare le voci di traccia CICS , consultare Tabella 153 a pagina 1037.

La voce di traccia CICS per questi valori è AP0 *xxx* (dove *xxx* è il numero di traccia specificato quando l'adattatore CICS è stato abilitato). Tutte le voci di traccia tranne CSQCTEST vengono emesse da CSQCTRUE. CSQCTEST viene emesso da CSQCRST e CSQCDSP.

Tabella 153. Voci di traccia dell'adattatore CICS

Nome	Descrizione	Sequenza traccia	Dati di traccia
CSQCABT	Interruzione anomala	Prima di emettere END_THREAD ANORMALE per IBM MQ. Ciò è dovuto alla fine dell'attività e all'esecuzione di un backout implicito da parte dell'applicazione. Una richiesta ROLLBACK è inclusa nella chiamata END_THREAD in questo caso.	Informazioni sull'unità di lavoro. È possibile utilizzare queste informazioni quando si individua lo stato del lavoro. (Ad esempio, può essere verificato rispetto all'output prodotto dal comando DISPLAY THREAD o dal programma di utilità di stampa del log IBM MQ for z/OS .)
CSQCBACK	Backout punto di sincronizzazione	Prima di emettere BACKOUT per IBM MQ for z/OS. Ciò è dovuto a una richiesta di backout esplicita dall'applicazione.	Informazioni sull'unità di lavoro.
CSQCCCRC	Codice di completamento e codice motivo	Dopo la restituzione non riuscita dalla chiamata API.	Codice di completamento e codice motivo.

Tabella 153. Voci di traccia dell'adattatore CICS (Continua)

Nome	Descrizione	Sequenza traccia	Dati di traccia
CSQCCOMM	Commit punto di sincronizzazione	Prima di emettere COMMIT per IBM MQ for z/OS. Ciò può essere dovuto a una richiesta di commit a fase singola o alla seconda fase di una richiesta di commit a due fasi. La richiesta è dovuta ad una richiesta esplicita del punto di sincronizzazione dall'applicazione.	Informazioni sull'unità di lavoro.
CSQCEXER	Esegui risoluzione	Prima di emettere EXECUTE_RESOLVE in IBM MQ for z/OS.	Le informazioni sull'unità di lavoro dell'unità di lavoro che emette EXECUTE_RESOLVE. È l'ultima unità di lavoro dubbia nel processo di risincronizzazione.
CSQCGETW	Attesa GET	Prima di emettere CICS attendere.	Indirizzo della BCE da attendere.
CSQCGMGD	GET dati messaggio	Dopo la riuscita del ritorno da MQGET.	Fino a 40 byte dei dati del messaggio.
CSQCGMGH	Gestione messaggio GET	Prima di emettere MQGET per IBM MQ for z/OS.	Handle oggetto.
CSQCGMGI	Ottieni ID messaggio	Dopo la riuscita del ritorno da MQGET.	ID messaggio e ID correlazione del messaggio.
CSQCINDL	Elenco in dubbio	Dopo un ritorno positivo dal secondo INQUIRE_INDOUBT.	L'elenco delle unità di lavoro in dubbio.
CSQCINDO	IBM utilizza solo		
CSQCINDS	Dimensione elenco in dubbio	Dopo un esito positivo, il primo INQUIRE_INDOUBT e l'elenco dei dubbi non sono vuoti.	Lunghezza dell'elenco. Diviso per 64 indica il numero di unità di lavoro in dubbio.
CSQCINQH	Handle INQ	Prima di emettere MQINQ per IBM MQ for z/OS.	Handle oggetto.
CSQCLOSH	Handle CLOSE	Prima di emettere MQCLOSE per IBM MQ for z/OS.	Handle oggetto.
CSQCLOST	Disposizione persa	Durante il processo di risincronizzazione, CICS informa l'adattatore che è stato riavviato in modo che non siano disponibili informazioni di disposizione relative all'unità di lavoro risincronizzata.	ID unità di lavoro noto a CICS per l'unità di lavoro che si sta risincronizzando.
CSQCNIND	Disposizione non dubbia	Durante il processo di risincronizzazione, CICS informa l'adattatore che l'unità di lavoro che si sta risincronizzando non dovrebbe essere dubbia (ovvero, è ancora in esecuzione).	ID unità di lavoro noto a CICS per l'unità di lavoro che si sta risincronizzando.

Tabella 153. Voci di traccia dell'adattatore CICS (Continua)

Nome	Descrizione	Sequenza traccia	Dati di traccia
CSQCNORT	Terminazione normale	Prima di immettere END_THREAD NORMAL per IBM MQ for z/OS. Ciò è dovuto alla fine dell'attivit ... e quindi l'applicazione potrebbe eseguire un commit del punto di sincronizzazione implicito. Una richiesta COMMIT è inclusa nella chiamata END_THREAD in questo caso.	Informazioni sull'unità di lavoro.
CSQCOPNH	Handle OPEN	Dopo il corretto ritorno da MQOPEN.	Handle oggetto.
CSQCOPNO	Oggetto aperto	Prima di emettere MQOPEN per IBM MQ for z/OS.	Il nome dell'oggetto.
CSQCPMGD	Dati del messaggio PUT	Prima di emettere MQPUT per IBM MQ for z/OS.	Fino a 40 byte dei dati del messaggio.
CSQCPMGH	Handle del messaggio PUT	Prima di emettere MQPUT per IBM MQ for z/OS.	Handle oggetto.
CSQCPMGI	ID messaggio PUT	Dopo MQPUT eseguito correttamente da IBM MQ for z/OS.	ID messaggio e ID correlazione del messaggio.
CSQCPREP	Preparazione punto di sincronizzazione	Prima di emettere PREPARE a IBM MQ for z/OS nella prima fase dell'elaborazione del commit a due fasi. Questa chiamata può essere emessa anche dal componente di accodamento distribuito come una chiamata API.	Informazioni sull'unità di lavoro.
CSQCP1MD	Dati del messaggio PUTONE	Prima di emettere MQPUT1 per IBM MQ for z/OS.	Fino a 40 byte di dati del messaggio.
CSQCP1MI	ID messaggio PUTONE	Dopo il corretto ritorno da MQPUT1.	ID messaggio e ID correlazione del messaggio.
CSQCP1ON	Nome oggetto PUTONE	Prima di emettere MQPUT1 per IBM MQ for z/OS.	Il nome dell'oggetto.
CSQCRBAK	Backout risolto	Prima di immettere RESOLVE_ROLLBACK su IBM MQ for z/OS.	Informazioni sull'unità di lavoro.
CSQCRCMT	Commit risolto	Prima di immettere RESOLVE_COMMIT in IBM MQ for z/OS.	Informazioni sull'unità di lavoro.

Tabella 153. Voci di traccia dell'adattatore CICS (Continua)			
Nome	**Descrizione**	**Sequenza traccia**	**Dati di traccia**
CSQCRMIR	Risposta RMI	Prima di tornare a CICS RMI (resource manager interface) da un richiamo specifico.	Valore di risposta RMI progettato. Il suo significato dipende dal tipo di chiamata. Questi valori sono documentati in *CICS Transaction Server for z/OS Customization Guide*. Per determinare il tipo di richiamo, esaminare le precedenti voci di traccia prodotte dal componente CICS RMI.
CSQCRSYN	Risincronizzazione	Prima che il processo di risincronizzazione venga avviato per l'attività.	ID unità di lavoro noto a CICS per l'unità di lavoro che si sta risincronizzando.
CSQCSETH	Handle SET	Prima di emettere MQSET per IBM MQ for z/OS.	Handle oggetto.
SSQCTASE	IBM utilizza solo		
CSQCTEST	Test di traccia	Utilizzato nella chiamata EXEC CICS ENTER TRACE per verificare il numero di traccia fornito dall'utente o lo stato di traccia della connessione.	Nessun dato.
CSQCDCFF	IBM utilizza solo		

Gestione degli errori del programma procedurale

Queste informazioni spiegano gli errori associati alle chiamate MQI delle applicazioni quando effettua una chiamata o quando il suo messaggio viene consegnato alla destinazione finale.

Quando possibile, il gestore code restituisce eventuali errori non appena viene effettuata una chiamata MQI. Questi sono *errori determinati localmente*.

Quando si inviano messaggi a una coda remota, gli errori potrebbero non essere evidenti quando viene effettuata la chiamata MQI. In questo caso, il gestore code che identifica gli errori li riporta inviando un altro messaggio al programma di origine. Si tratta di *errori determinati in remoto*.

Errori determinati localmente

Informazioni sugli errori determinati localmente che includono: errore su una chiamata MQI, interruzioni del sistema e messaggi contenenti dati non corretti.

Le tre cause più comuni di errori che il gestore code può segnalare immediatamente sono:

- Errore di una chiamata MQI; ad esempio, perché una coda è piena
- Un'interruzione dell'esecuzione di alcune parti del sistema da cui dipende l'applicazione; ad esempio, il gestore code
- Messaggi che contengono dati che non possono essere elaborati correttamente

Se si sta utilizzando la funzione di inserimento asincrono, gli errori non vengono notificati immediatamente. Utilizzare la chiamata MQSTAT per richiamare le informazioni sullo stato delle precedenti operazioni di immissione asincrona.

Errore di una chiamata MQI

Il gestore code può segnalare immediatamente eventuali errori nella codifica di una chiamata MQI. Lo fa utilizzando una serie di codici di ritorno predefiniti. Questi sono divisi in codici di completamento e codici di errore.

Per mostrare se una chiamata ha esito positivo, il gestore code restituisce un *codice di completamento* quando la chiamata viene completata. Esistono tre codici di completamento, che indicano l'esito positivo, il completamento parziale e l'errore della chiamata. Il gestore code restituisce anche un *codice motivo* che indica il motivo per il completamento parziale o l'errore della chiamata.

I codici di completamento e motivo per ogni chiamata vengono elencati con la descrizione di tale chiamata in Codici di ritorno. Per informazioni più dettagliate, incluse le idee per un'azione correttiva, consultare:

- z/OS IBM MQ for z/OS , completamento, e codici di errore per IBM MQ for z/OS
- Messaggi e codici di errore per tutte le altre piattaforme IBM MQ

Progetta i tuoi programmi per gestire tutti i codici di ritorno che possono derivare da ogni chiamata.

System interruptions

L'applicazione potrebbe non essere a conoscenza di eventuali interruzioni se il gestore code a cui è connesso deve essere ripristinato da un malfunzionamento del sistema. Tuttavia, è necessario progettare l'applicazione per garantire che i dati non vadano persi se si verifica tale interruzione.

I metodi che è possibile utilizzare per assicurarsi che i propri dati rimangano congruenti dipendono dalla piattaforma su cui è in esecuzione il gestore code:

z/OS z/OS

Negli ambienti CICS e IMS , è possibile effettuare chiamate MQPUT e MQGET all'interno di unità di lavoro gestite da CICS o IMS. Nell'ambiente batch, è possibile effettuare chiamate MQPUT e MQGET nello stesso modo, ma è necessario dichiarare i punti di sincronizzazione utilizzando:

- Le chiamate IBM MQ for z/OS MQCMIT e MQBACK (consultare “Commit e backout delle unità di lavoro” a pagina 854) o
- RRS (z/OS Transaction Management and Recoverable Resource Manager Services) per fornire il supporto punto di sincronizzazione a due fasi. RRS consente di aggiornare sia IBM MQ che altre risorse del prodotto abilitate per RRS, come le risorse della procedura memorizzata Db2 , all'interno di una singola unità logica di lavoro. Per informazioni sul supporto del punto di sincronizzazione RRS, consultare “Gestione delle transazioni e servizi di gestione delle risorse recuperabili” a pagina 859.

IBM i IBM i

È possibile effettuare chiamate MQPUT e MQGET all'interno di unità di lavoro globali gestite dal controllo del commit IBM i . È possibile dichiarare i punti di sincronizzazione utilizzando i comandi nativi IBM i COMMIT e ROLLBACK o i comandi specifici della lingua. Le unità di lavoro locali sono gestite da IBM MQ utilizzando le chiamate MQCMIT e MQBACK.

Sistemi AIX, Linux, and Windows

In questi ambienti, è possibile effettuare le chiamate MQPUT e MQGET nel solito modo, ma è necessario dichiarare i punti di sincronizzazione utilizzando le chiamate MQCMIT e MQBACK (consultare “Commit e backout delle unità di lavoro” a pagina 854). Nell'ambiente CICS , i comandi MQCMIT e MQBACK sono disabilitati, perché è possibile effettuare le proprie chiamate MQPUT e MQGET all'interno di unità di lavoro gestite da CICS.

Utilizzare i messaggi persistenti per trasportare tutti i dati che non ci si può permettere di perdere. I messaggi persistenti vengono reintegrati nelle code se il gestore code deve eseguire il ripristino da un malfunzionamento. ALW Con IBM MQ su AIX, Linux, and Windows, una chiamata MQGET o MQPUT all'interno dell'applicazione avrà esito negativo al punto di riempire tutti i file di log, con il messaggio MQRC_RESOURCE_PROBLEM. Per ulteriori informazioni sui file di log su AIX, Linux, and

Windows, consultare Amministrazione IBM MQ. z/OS Per z/OS consultare Pianificazione dei messaggi z/OS.

Se il gestore code viene arrestato da un operatore mentre un'applicazione è in esecuzione, di solito viene utilizzata l'opzione di sospensione. Il gestore code entra in uno stato di sospensione in cui le applicazioni possono continuare a lavorare, ma devono terminare non appena possibile. Le applicazioni piccole e rapide possono probabilmente ignorare lo stato di sospensione e continuare fino a quando non terminano normalmente. Le applicazioni in esecuzione più a lungo, o quelle che attendono l'arrivo dei messaggi, devono utilizzare l'opzione *fail if quiescing* quando utilizzano le chiamate MQOPEN, MQPUT, MQPUT1e MQGET. Queste opzioni indicano che le chiamate hanno esito negativo quando il gestore code viene disattivato, ma l'applicazione potrebbe avere ancora tempo per terminare in modo pulito emettendo chiamate che ignorano lo stato di disattivazione. Tali applicazioni potrebbero anche eseguire il commit o il backout delle modifiche apportate e quindi terminare.

Se viene forzato l'arresto del gestore code (ovvero, l'arresto senza disattivazione), le applicazioni riceveranno il codice motivo MQRC_CONNECTION_BROKEN quando effettuano chiamate MQI. Uscire dall'applicazione o, in alternativa, sui sistemi IBM i IBM MQ for IBM i, AIX, Linux, and Windows , emettere una chiamata MQDISC.

Messaggi contenenti dati non corretti

Quando si utilizzano le unità di lavoro nella propria applicazione, se un programma non può elaborare correttamente un messaggio che richiama da una coda, viene eseguito il backout della chiamata MQGET.

Il gestore code conserva un contatore (nel campo *BackoutCount* del descrizione del messaggio) del numero di volte che si verifica. Mantiene questo conteggio nel descrittore di ogni messaggio interessato. Questo conteggio può fornire informazioni utili sull'efficienza di una applicazione. I messaggi con conteggi di backout in aumento nel tempo vengono ripetutamente rifiutati; progettare l'applicazione in modo che analizzi i motivi e gestisca tali messaggi di conseguenza.

z/OS Su IBM MQ for z/OS, per fare in modo che il conteggio di backout sopravviva ai riavvii del gestore code, impostare l'attributo **HardenGetBackout** su MQQA_BACKOUT_HARDENED; altrimenti, se il gestore code deve essere riavviato, non conserva un conteggio di backout accurato per ciascun messaggio. Impostando l'attributo in questo modo si aggiunge la penalità dell'elaborazione supplementare.

Su IBM MQ per sistemi IBM i IBM i, AIX, Linux, and Windows , il conteggio di backout sopravvive sempre al riavvio del gestore code.

z/OS Inoltre, su IBM MQ for z/OS, quando si rimuovono i messaggi da una coda all'interno di un'unità di lavoro, è possibile contrassegnare un messaggio in modo che non venga reso nuovamente disponibile se l'unità di lavoro viene ripristinata dall'applicazione. Il messaggio contrassegnato viene considerato come se fosse stato richiamato in una nuova unità di lavoro. Contrassegnare il messaggio per ignorare il backout utilizzando l'opzione MQGMO_MARK_SKIP_BACKOUT(nella struttura MQGMO) quando si utilizza la chiamata MQGET. Consultare “Backout ignorato” a pagina 800 per ulteriori informazioni su questa tecnica.

Utilizzo dei messaggi di report per la determinazione dei problemi

Il gestore code remoto non è in grado di riportare errori quali l'errore di inserimento di un messaggio in una coda quando si effettua la chiamata MQI, ma può inviare un messaggio di report per indicare come è stato elaborato il messaggio.

All'interno dell'applicazione è possibile creare messaggi di report (MQPUT) e selezionare l'opzione per riceverli (nel qual caso vengono inviati da un'altra applicazione o da un gestore code).

Creazione di messaggi di report

I messaggi di report consentono a un'applicazione di comunicare a un'altra applicazione che non può gestire il messaggio inviato.

Tuttavia, il campo *Report* deve essere inizialmente analizzato per determinare se l'applicazione che ha inviato il messaggio è interessata ad essere informata di eventuali problemi. Dopo aver determinato che è richiesto un messaggio di report, è necessario decidere:

- Se si desidera includere l'intero messaggio originale, solo i primi 100 byte di dati o nessuno dei messaggi originali.
- Cosa fare con il messaggio originale. È possibile eliminarlo o lasciarlo andare alla coda di messaggi non recapitabili.
- Se anche il contenuto dei campi *MsgId* e *CorrelId* è necessario.

Utilizzare il campo *Feedback* per indicare il motivo della generazione del messaggio di report. Inserire i messaggi di report in una coda di risposta dell'applicazione. Per ulteriori informazioni, consultare Feedback .

Richiesta e ricezione di messaggi di report (MQGET)

Quando si invia un messaggio a un'altra applicazione, non si viene informati di eventuali problemi a meno che non si completi il campo *Report* per indicare il feedback richiesto. Consultare Struttura del campo del report per le opzioni disponibili.

I gestori code inserisano sempre i messaggi di report nella coda di risposta di un'applicazione e si consiglia alle proprie applicazioni di fare lo stesso. Quando si utilizza la funzione del messaggio di report, specificare il nome della propria coda di risposta nel descrittore del messaggio; in caso contrario, la chiamata MQPUT non riesce.

L'applicazione deve contenere procedure che monitorano la coda di risposta ed elaborano i messaggi che arrivano su di essa. Tenere presente che un messaggio di report può contenere tutti i messaggi originali, i primi 100 byte del messaggio originale o nessuno dei messaggi originali.

Il gestore code imposta il campo *Feedback* del messaggio di report per indicare il motivo dell'errore; ad esempio, la coda di destinazione non esiste. I programmi dovrebbero fare lo stesso.

Per ulteriori informazioni sui messaggi di report, consultare “Messaggi di report” a pagina 21.

Errori determinati in remoto

Quando si inviano messaggi a una coda remota, anche quando il gestore code locale ha elaborato la chiamata MQI senza trovare un errore, altri fattori possono influenzare il modo in cui il messaggio viene gestito da un gestore code remoto.

Ad esempio, la coda di destinazione potrebbe essere piena o non esistere. Se il messaggio deve essere gestito da altri gestori code intermedi sull'instradamento alla coda di destinazione, uno di questi potrebbe rilevare un errore.

Problemi nella consegna di un messaggio

Quando una chiamata MQPUT ha esito negativo, è possibile provare a inserire nuovamente il messaggio nella coda, restituirlo al mittente o inserirlo nella coda di messaggi non recapitabili.

Ogni opzione ha i suoi meriti, ma è possibile che non si desideri provare a inserire di nuovo un messaggio se il motivo per cui MQPUT non è riuscito è che la coda di destinazione era piena. In questa istanza, l'inserimento nella coda di messaggi non recapitabili consente di consegnarla successivamente alla coda di destinazione corretta.

Riprova consegna messaggio

Prima che il messaggio venga inserito in una coda di messaggi non instradabili, un gestore code remoto tenta di inserire di nuovo il messaggio nella coda se gli attributi *MsgRetryCount* e

MsgRetryInterval sono stati impostati per il canale o se esiste un programma di uscita per i tentativi da utilizzare (il cui nome è contenuto nel campo *MsgRetryExitId* dell'attributo del canale).

Se il campo *MsgRetryExitId* è vuoto, vengono utilizzati i valori negli attributi *MsgRetryCount* e *MsgRetryInterval* .

Se il campo *MsgRetryExitId* non è vuoto, viene eseguito il programma di uscita con questo nome. Per ulteriori informazioni sull'utilizzo dei propri programmi di uscita, consultare "Programmi di uscita canale per canali di messaggistica" a pagina 963.

Restituisci messaggio al mittente

Si restituisce un messaggio al mittente richiedendo che venga generato un messaggio di report per includere tutto il messaggio originale.

Consultare "Messaggi di report" a pagina 21 per i dettagli sulle opzioni dei messaggi di report.

Utilizzo della coda dei messaggi non recapitabili (messaggi non recapitati)

Quando un gestore code non è in grado di consegnare un messaggio, tenta di inserire il messaggio nella coda di messaggi non recapitabili. Questa coda deve essere definita quando il gestore code è installato.

I programmi possono utilizzare la coda di messaggi non recapitabili nello stesso modo in cui viene utilizzata dal gestore code. È possibile trovare il nome della coda di messaggi non recapitabili aprendo l'oggetto gestore code (utilizzando la chiamata MQOPEN) e verificando l'attributo **DeadLetterQName** (utilizzando la chiamata MQINQ).

Quando il gestore code inserisce un messaggio su questa coda, aggiunge un'intestazione al messaggio, il cui formato è descritto dalla struttura MQDLH (dead - letter header); consultare MQDLH - Dead - letter header. Questa intestazione include il nome della coda di destinazione e il motivo per cui il messaggio è stato inserito sulla coda di messaggi non recapitabili. È necessario rimuoverlo e risolvere il problema prima di inserire il messaggio nella coda desiderata. Inoltre, il gestore code modifica il campo *Format* del descrittore del messaggio (MQMD) per indicare che il messaggio contiene una struttura MQDLH.

Struttura MQDLH

si consiglia di aggiungere una struttura MQDLH a tutti i messaggi inseriti nella coda di messaggi non recapitabili; tuttavia, se si intende utilizzare il gestore di messaggi non recapitabili fornito da determinati prodotti IBM MQ , è necessario aggiungere una struttura MQDLH ai messaggi.

L'aggiunta dell'intestazione a un messaggio potrebbe rendere il messaggio troppo lungo per la coda di messaggi non recapitabili, quindi accertarsi sempre che i messaggi siano più brevi della dimensione massima consentita per la coda di messaggi non recapitabili, almeno del valore della costante MQ_MSG_HEADER_LENGTH. La dimensione massima dei messaggi consentiti in una coda è determinata dal valore dell'attributo **MaxMsgLength** della coda. Per la coda di messaggi non recapitabili, assicurarsi di impostare questo attributo sul valore massimo consentito dal gestore code. Se l'applicazione non è in grado di consegnare un messaggio e il messaggio è troppo lungo per essere inserito nella coda di messaggi non recapitabili, seguire i consigli forniti nella descrizione della struttura MQDLH.

Verificare che la coda di messaggi non recapitabili sia monitorata e che tutti i messaggi in arrivo su di essa vengano elaborati. Il gestore code di messaggi non instradabili viene eseguito come un programma di utilità batch e può essere utilizzato per eseguire varie azioni sui messaggi selezionati sulla coda di messaggi non instradabili. Per ulteriori dettagli, fare riferimento a "Elaborazione coda di messaggi non recapitabili" a pagina 1045.

Se la conversione dati è necessaria, il gestore code converte le informazioni di intestazione quando si utilizza l'opzione MQGMO_CONVERT nella chiamata MQGET. Se il processo di inserimento del messaggio è un MCA, l'intestazione è seguita da tutto il testo del messaggio originale.

I messaggi inseriti nella coda di messaggi non recapitabili potrebbero essere troncati se sono troppo lunghi per questa coda. Una possibile indicazione di questa situazione è che i messaggi sulla coda di messaggi non instradabili hanno la stessa lunghezza del valore dell'attributo **MaxMsgLength** della coda.

Elaborazione coda di messaggi non recapitabili
Queste informazioni contengono informazioni generali sull'interfaccia di programmazione quando si utilizza l'elaborazione della coda di messaggi non recapitabili.

L'elaborazione della coda dei messaggi non instradati dipende dai requisiti del sistema locale, ma considerare quanto segue quando si stabilisce la specifica:

- Il messaggio può essere identificato come avente un'intestazione della coda di messaggi non instradabili poiché il valore del campo formato in MQMD è MQFMT_DEAD_LETTER_HEADER.
- Su IBM MQ for z/OS utilizzo CICS, se un MCA inserisce questo messaggio nella coda di messaggi non recapitabili, il campo *PutApplType* è MQAT_CICS e il campo *PutApplName* è il *ApplId* del sistema CICS seguito dal nome transazione dell'MCA.
- Il motivo per cui il messaggio viene instradato alla coda di messaggi non instradati è contenuto nel campo *Reason* dell'intestazione della coda di messaggi non instradati.
- L'intestazione della coda di messaggi non instradabili contiene i dettagli del nome della coda di destinazione e del nome del gestore code.
- L'intestazione della coda di messaggi non instradabili contiene campi che devono essere ripristinati nel descrittore del messaggio prima che il messaggio venga inserito nella coda di destinazione. Essi sono:
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- Il descrittore del messaggio è uguale a PUT dall'applicazione originale, ad eccezione dei tre campi mostrati (Codifica, CodedCharSetIde Formato).

L'applicazione della coda di messaggi non recapitabili deve eseguire una o più delle seguenti operazioni:

- Esaminare il campo *Reason* . Un messaggio potrebbe essere stato inserito da un MCA per i seguenti motivi:
  - Il messaggio era più lungo della dimensione massima del messaggio per il canale

    Il motivo è MQRC_MSG_TOO_BIG_FOR_CHALLEGATO
  - Non è stato possibile inserire il messaggio nella coda di destinazione

    Il motivo è un qualsiasi codice motivo MQRC_ * che può essere restituito da un'operazione MQPUT
  - Questa azione è stata richiesta da un'uscita utente

    Il codice di errore è quello fornito dall'uscita utente oppure il codice MQRC_SUPPRESSED_BY_EXIT predefinito
- Provare ad inoltrare il messaggio alla destinazione desiderata, dove ciò è possibile.
- Conservare il messaggio per un certo periodo di tempo prima di scartare quando viene determinato il motivo della deviazione, ma non immediatamente correggibile.
- Fornire istruzioni agli amministratori per correggere i problemi in cui sono stati determinati.
- Eliminare i messaggi danneggiati o non elaborabili.

Esistono due modi per gestire i messaggi recuperati dalla coda di messaggi non recapitabili:

1. Se il messaggio è per una coda locale:
   - Eseguire tutte le traduzioni di codice richieste per estrarre i dati dell'applicazione
   - Eseguire le conversioni di codice su tali dati se si tratta di una funzione locale
   - Inserire il messaggio risultante nella coda locale con tutti i dettagli del descrittore del messaggio ripristinato
2. Se il messaggio è per una coda remota, inserirlo nella coda.

Per informazioni sul modo in cui vengono gestiti i messaggi non recapitati in un ambiente di accodamento distribuito, consultare Cosa succede quando un messaggio non può essere consegnato?.

Programmazione multicast

Utilizzare queste informazioni per informazioni sulle IBM MQ attività di programmazione multicast come la connessione a un gestore code e la creazione di report di eccezioni.

IBM MQ Multicast è stato progettato per essere il più trasparente possibile per l'utente e allo stesso tempo ancora compatibile con le applicazioni esistenti. Definire un oggetto COMMINFO e impostare i parametri **MCAST** e **COMMINFO** dell'oggetto TOPIC significa che le applicazioni IBM MQ esistenti non richiedono una riscrittura sostanziale per utilizzare il multicast. Tuttavia, potrebbero essere presenti alcune limitazioni (consultare “Multicast e MQI” a pagina 1046 per ulteriori informazioni) e alcuni problemi di sicurezza da considerare (consultare Sicurezza multicast per ulteriori informazioni).

Multicast e MQI

Utilizzare queste informazioni per comprendere i concetti principali di MQI (Message Queue Interface) e come sono correlati a IBM MQ Multicast.

Le sottoscrizioni multicast non sono durevoli; poiché non vi sono code fisiche coinvolte, non è possibile memorizzare i messaggi non in linea creati dalle sottoscrizioni durevoli.

Dopo che un'applicazione ha effettuato la sottoscrizione a un argomento multicast, le viene restituito un handle di oggetto che può utilizzare o MQGET da, come se fosse un handle per una coda. Ciò significa che sono supportate solo le sottoscrizioni multicast gestite (sottoscrizioni create con MQSO_MANAGED), ossia non è possibile effettuare una sottoscrizione e 'puntare ' i messaggi su una coda. Ciò significa che i messaggi devono essere utilizzati dall'handle dell'oggetto restituito sulla chiamata di sottoscrizione. Sul client, i messaggi vengono memorizzati in un buffer di messaggi fino a quando non vengono utilizzati dal client; per ulteriori informazioni, consultare la stanza MessageBuffer del file di configurazione del client . Se il client non si mantiene al passo con la velocità di pubblicazione, i messaggi vengono eliminati come richiesto, con i messaggi meno recenti eliminati per primi.

Di solito è una decisione di amministrazione se un'applicazione utilizza o meno Multicast, specificata impostando l'attributo MCAST di un oggetto TOPIC. Se un'applicazione di pubblicazione deve garantire che il multicast non venga utilizzato, può utilizzare l'opzione `MQOO_NO_MULTICAST` . Allo stesso modo, un'applicazione di sottoscrizione può garantire che il multicast non venga utilizzato sottoscrivendo con l'opzione `MQSO_NO_MULTICAST` .

IBM MQ Multicast supporta l'uso di selettori di messaggi. Un selettore viene utilizzato da una applicazione per registrare il suo interesse solo in quei messaggi con proprietà che soddisfano la query SQL92 rappresentata dalla stringa di selezione. Per ulteriori informazioni sui selettori di messaggi, consultare “Selettori” a pagina 31.

La seguente tabella elenca tutti i principali concetti MQI e come sono correlati a Multicast:

Tabella 154. Concetti MQI e modalità di correlazione con multicast

Concetto MQI	**Azione quando si tenta di utilizzare il multicast**	**Codice di errore**
Inserimento di un messaggio di lunghezza zero	Rifiutato	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Raggruppamento	Rifiutato	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentazione	Rifiutato	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
Liste di distribuzione	Rifiutato	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR

Tabella 154. Concetti MQI e modalità di correlazione con multicast (Continua)

Concetto MQI	Azione quando si tenta di utilizzare il multicast	Codice di errore
MQINQ	Rifiutato per handle di argomenti: MQINQ e MQSET di argomenti non sono supportati.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	Accettato per l'handle gestito. È possibile interrogare solo `Profondità corrente` .	• Se il valore è Profondità corrente, non esiste alcun codice motivo applicabile. • Se il valore è diverso da Profondità corrente, il codice motivo è 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR.
MQSET	Rifiutato per tutti gli handle.	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
Transazioni (XA o meno)	Rifiutato	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
Ricerca messaggi	Rifiutato	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Blocca messaggi	Rifiutato	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Sfoglia con contrassegno	Rifiutato	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Passa contesto	Rifiutato	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	Rifiutato. Non è valido provare MQPUT1 per un argomento solo multicast.	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
Sottoscrizione durevole	Rifiutato se l'argomento è contrassegnato come "Solo multicast", altrimenti viene effettuata una sottoscrizione non multicast.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	Rifiutato. Se la stringa dell'argomento contiene più di 255 caratteri, viene rifiutata nel client.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR

Tabella 154. Concetti MQI e modalità di correlazione con multicast (Continua)		
Concetto MQI	**Azione quando si tenta di utilizzare il multicast**	**Codice di errore**
Sottoscrizione non gestita effettuata	Rifiutato se l'argomento è contrassegnato come "Solo multicast", altrimenti viene effettuata una sottoscrizione non multicast.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	Rifiutato	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

I seguenti elementi espandono alcuni dei concetti MQI della precedente tabella e forniscono informazioni su alcuni concetti MQI non presenti nella tabella:

Persistenza messaggio
Per sottoscrittori multicast non durevoli, i messaggi persistenti del publisher vengono consegnati in modo irreversibile.

Troncamento dei messaggi
Il troncamento del messaggio è supportato, il che significa che è possibile per un'applicazione:

1. Emettere un MQGET.
2. Richiamare MQRC_TRUNCATED_MSG_FAILED.
3. Assegnare un buffer più grande.
4. Emettere nuovamente MQGET per recuperare il messaggio.

Scadenza sottoscrizione
La scadenza della sottoscrizione non è supportata. Qualsiasi tentativo di impostare una scadenza viene ignorato.

Alta disponibilità per multicast

Utilizzare queste informazioni per comprendere l'operazione peer-to-peer continua IBM MQ Multicast; sebbene IBM MQ si colleghi a un gestore code IBM MQ , i messaggi non passano attraverso tale gestore code.

Anche se è necessario stabilire una connessione a un gestore code per consentire a MQOPEN o MQSUB l'oggetto argomento multicast, i messaggi stessi non passano attraverso il gestore code. Pertanto, dopo che MQOPEN o MQSUB è stato completato sull'oggetto dell'argomento multicast, è possibile continuare a trasmettere messaggi multicast anche se la connessione al gestore code è stata persa. Esistono due modalità di funzionamento:

Viene stabilita una connessione normale al gestore code
La comunicazione multicast è possibile mentre esiste la connessione al gestore code. Se la connessione non riesce, vengono applicate le normali regole MQI, ad esempio; un MQPUT all'handle dell'oggetto multicast restituisce 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

Viene effettuata una connessione client al gestore code
La comunicazione multicast è possibile anche durante il ciclo di riconnessione. Ciò significa che anche quando la connessione al gestore code è stata interrotta, l'immissione e l'utilizzo di messaggi multicast non vengono influenzati. Il client tenta di riconnettersi a un gestore code e, se la riconnessione ha esito negativo, l'handle di connessione si interrompe e tutte le chiamate MQI, incluse quelle multicast, non riescono. Per ulteriori informazioni, consultare: Automatic client reconnection

Se un'applicazione emette esplicitamente un MQDISC, tutte le sottoscrizioni multicast e gli handle degli oggetti vengono chiusi.

Operazione peer-to-peer continua multicast

Uno dei vantaggi della comunicazione peer - to - peer tra i client è che i messaggi non devono passare attraverso il gestore code; pertanto, se la connessione al gestore code si interrompe, il trasferimento dei messaggi continua. Le seguenti limitazioni si applicano ai requisiti dei messaggi continui di questa modalità:

- La connessione deve essere effettuata utilizzando una delle opzioni MQCNO_RECONNECT_ * per l'operazione continua. Questo processo indica che sebbene la sessione di comunicazioni potrebbe essere interrotta, l'handle di connessione effettivo non è interrotto e si trova invece nello stato di riconnessione. Se la riconnessione non riesce, l'handle di connessione è ora interrotto e ciò impedisce tutte le ulteriori chiamate MQI.
- In questa modalità sono supportati solo MQPUT, MQGET, MQINQ e Async Consume. Qualsiasi comando MQOPEN, MQCLOSE o MQDISC richiede il completamento della riconnessione al gestore code.
- Lo stato passa all'arresto del gestore code; qualsiasi stato nel gestore code potrebbe quindi essere obsoleto o mancante. Ciò significa che i client potrebbero inviare e ricevere messaggi e che non vi è alcuno stato noto sul gestore code. Per ulteriori informazioni, consultare: Monitoraggio dell'applicazione multicast

Conversione dati in MQI per la messaggistica multicast

Utilizzare queste informazioni per comprendere il funzionamento della conversione dei dati per la messaggistica IBM MQ Multicast.

IBM MQ Multicast è un protocollo condiviso, senza connessione e quindi non è possibile per ogni client effettuare richieste specifiche per la conversione dei dati. Ogni client sottoscritto allo stesso flusso multicast riceve gli stessi dati binari; pertanto, se è richiesta la conversione dei dati IBM MQ , la conversione viene eseguita localmente su ciascun client.

I dati vengono convertiti sul client per il traffico multicast IBM MQ . Se viene specificata l'opzione **MQGMO_CONVERT** , la conversione dei dati viene eseguita come richiesto. I formati definiti dall'utente richiedono l'uscita di conversione dati installata sul client; consultare “Scrittura delle uscite di conversione dati” a pagina 985 per informazioni su quali librerie si trovano ora nei pacchetti client e server.

Per informazioni sulla gestione della conversione dati, consultare Abilitazione della conversione dati per la messaggistica multicast.

Per ulteriori informazioni sulla conversione dei dati, consultare Conversione dei dati.

Per ulteriori informazioni sulle uscite di conversione dati e `ClientExitPath`, consultare la sezione ClientExitPercorso del file di configurazione client.

Report di eccezioni multicast

Utilizzare queste informazioni per informazioni sui gestori eventi multicast IBM MQ e sulla notifica delle eccezioni multicast IBM MQ .

IBM MQ Multicast assiste nella determinazione dei problemi richiamando il gestore eventi per notificare eventi multicast riportati utilizzando il meccanismo del gestore eventi IBM MQ standard.

Un singolo evento multicast può determinare la chiamata di più di un evento IBM MQ perché potrebbero essere presenti più handle di connessione `MQHCONN` che utilizzano lo stesso trasmettitore o ricevitore multicast. Tuttavia, ogni eccezione multicast causa il richiamo di un solo gestore eventi per connessione IBM MQ .

La costante IBM MQ `MQCBDO_EVENT_CALL` consente alle applicazioni di registrare un callback per ricevere solo IBM MQ eventi e `MQCBDO_MC_EVENT_CALL` consente alle applicazioni di registrare un

callback per ricevere solo eventi multicast. Se vengono utilizzate entrambe le costanti, vengono ricevuti entrambi i tipi di evento.

Richiesta di eventi multicast

IBM MQ Gli eventi multicast utilizzano la costante MQCBDO_MC_EVENT_CALL nel campo cbd.Options . Il seguente esempio illustra come richiedere eventi multicast:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

Quando l'opzione MQCBDO_MC_EVENT_CALL viene specificata per il campo cbd.Options , il gestore eventi viene inviato solo agli eventi multicast IBM MQ invece che agli eventi del livello di connessione. Per richiedere che entrambi i tipi di eventi vengano inviati al gestore eventi, l'applicazione deve specificare la costante MQCBDO_EVENT_CALL nel campo cbd.Options e la costante MQCBDO_MC_EVENT_CALL come mostrato nel seguente esempio:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

Se nessuna di queste costanti viene utilizzata, solo gli eventi a livello di connessione vengono inviati al gestore eventi.

Per ulteriori informazioni sui valori per il campo Options , consultare Opzioni (MQLONG).

Formato evento multicast

IBM MQ Le eccezioni Multicast includono alcune informazioni di supporto restituite nel parametro **Buffer** della funzione di callback. Il puntatore **Buffer** punta ad un array di puntatori e il campo MQCBC.DataLength specifica la dimensione, in byte, dell'array. Il primo elemento dell'array punta sempre a una breve descrizione di testo dell'evento. Ulteriori parametri potrebbero essere forniti in base al tipo di evento. La seguente tabella elenca le eccezioni:

Tabella 155. Descrizioni codice evento multicast

Codice evento	Descrizione	Dati aggiuntivi
CARICO PACCHETTO MQMCEV	Perdita di pacchetti non recuperabile	Numero di pacchetti persi
MQMCEV_HEARTBEAT_TIMEOUT	Lunga assenza del pacchetto di controllo heartbeat	N/A
MQMCEV_VERSION_CONFLICT	Ricezione di pacchetti di versioni di protocollo più recenti	N/A
MQMCEV_AFFIDABILITÀ	Diverse modalità di affidabilità del trasmettitore e del ricevitore	N/A
TRANS MQMCEV_CLOSED_	La trasmissione dell'argomento è chiusa da 1 origine	N/A
ERRORE MQMCEV_STREAM_	Errore rilevato sul flusso	N/A
MQMCEV_NEW_SOURCE	Una nuova origine inizia a trasmettere sull'argomento	Struttura di origine

Tabella 155. Descrizioni codice evento multicast (Continua)

Codice evento	Descrizione	Dati aggiuntivi
MQMCEV_RECEIVE_QUEUE_TRIMMED	Pacchetti rimossi da PacketQ a causa della scadenza di tempo o spazio	Numero di pacchetti ritagliati
MQMCEV_PACKET_LOSS_NACK_EXPIRE	Perdita di pacchetti non recuperabile a causa della scadenza NACK	Numero di pacchetti persi
MQMCEV_ACK_RETRIES_SUPERATO	Pacchetti rimossi dalla cronologia dopo il superamento di **max_ack_retries**	Numero di pacchetti rimossi
MQMCEV_STREAM_SUSPEND_NACK	I NACK sono stati sospesi su un flusso accettato da questo argomento	Sospendi ID flusso Tempo in millisecondi in cui il flusso è sospeso per
MQMCEV_STREAM_RESUME_NACK	I NACK sono stati ripresi dopo essere stati sospesi su un flusso	ID flusso
MQMCEV_STREAM_ESPULSO	Un flusso accettato da questo argomento è stato rifiutato a causa di una richiesta di espulsione	ID flusso
MQMCEV_PRIMO_MESSAGGIO	Primo messaggio da un'origine	Numero messaggio
MQMCEV_LATE_JOIN_FAILURE	Impossibile avviare la sessione di unione in ritardo	N/A
MQMCEV_MESSAGE_LOSS	Perdita del messaggio non recuperabile	Numero di messaggi persi
MQMCEV_SEND_PACKET_FAILURE	Il trasmettitore multicast non è riuscito a inviare un pacchetto multicast	N/A
MQMCEV_REPAIR_DELAY	Il ricevitore multicast non ha ricevuto un pacchetto di riparazione per un NAK in sospeso	N/A
MQMCEV_MEMORY_ALERT_ATTIVO	I buffer di ricezione del ricevitore si stanno riempiendo	Percentuale di utilizzo bufferpool
MQMCEV_MEMORY_ALERT_OFF	I buffer di ricezione del ricevitore sono inattivi al livello normale	Percentuale di utilizzo bufferpool
MQMCEV_NACK_ALERT_ATTIVO	La frequenza della richiesta del pacchetto di riparazione del ricevitore ha raggiunto il limite massimo	La frequenza corrente delle richieste di riparazione in pacchetti al secondo
MQMCEV_NACK_ALERT_OFF	La velocità di richiesta del pacchetto di riparazione del destinatario è inferiore al normale	La frequenza corrente delle richieste di riparazione in pacchetti al secondo

Tabella 155. Descrizioni codice evento multicast (Continua)

Codice evento	Descrizione	Dati aggiuntivi
MQMCEV_REPAIR_ALERT_ON	La velocità di invio del pacchetto di riparazione del trasmettitore ha raggiunto il limite massimo	N/A
MQMCEV_REPAIR_ALERT_OFF	La velocità di invio del pacchetto di riparazione del trasmettitore è inferiore al normale	N/A
MQMCEV_SHM_DEST_UNUSABLE	È stato rilevato che la regione di memoria condivisa utilizzata da una destinazione argomento del trasmettitore è inutilizzabile	N/A
MQMCEV_SHM_PORT_UNUSABLE	È stato rilevato che la porta di memoria condivisa utilizzata da un'istanza del destinatario è inutilizzabile	N/A
MQMCEV_CCT_GETTIME_NON RIUSCITO	L'ora di richiamo dall'ora del cluster coordinato non è riuscita	N/A
MQMCEV_DEST_INTERFACE_FAILURE	L'interfaccia di rete utilizzata da una destinazione argomento del trasmettitore ha avuto esito negativo e un'interfaccia di rete di backup non è disponibile	
MQMCEV_DEST_INTERFACE_FAILOVER	L'interfaccia di rete utilizzata da una destinazione argomento del trasmettitore ha avuto esito negativo ed è stato completato correttamente un failover su un'altra interfaccia	
MQMCEV_PORT_INTERFACE - ERRORE	L'interfaccia di rete utilizzata da un ricevitore rmmPort ha avuto esito negativo e un'interfaccia di rete di backup non è disponibile (o ha avuto esito negativo)	Configurazione diRMM
MQMCEV_PORT_INTERFACE_FAILOVER	L'interfaccia di rete utilizzata da un ricevitore rmmPort ha avuto esito negativo ed è stato completato un failover corretto su un'altra interfaccia	Configurazione diRMM

Codifica in C

Prendere nota delle informazioni nelle sezioni seguenti quando si codificano i programmi IBM MQ in C.

- "Parametri delle chiamate MQI" a pagina 1053
- "Parametri con tipo di dati non definito" a pagina 1053
- "Tipi di dati" a pagina 1053
- "Manipolazione di stringhe binarie" a pagina 1053
- "Manipolazione delle stringhe di caratteri" a pagina 1054
- "Valori iniziali per le strutture" a pagina 1054

- “Valori iniziali per le strutture dinamiche” a pagina 1054
- “Utilizza da C++” a pagina 1055

Parametri delle chiamate MQI

I parametri che sono *solo input* e di tipo MQHCONN, MQHOBJ, MQHMSG o MQLONG vengono passati per valore; per tutti gli altri parametri, l' *indirizzo* del parametro viene passato per valore.

Non tutti i parametri che vengono passati per indirizzo devono essere specificati ogni volta che viene richiamata una funzione. Quando un particolare parametro non è richiesto, è possibile specificare un puntatore null come parametro sul richiamo della funzione, al posto dell'indirizzo dei dati del parametro. I parametri per i quali ciò è possibile sono identificati nelle descrizioni delle chiamate.

Non viene restituito alcun parametro come valore della funzione; nella terminologia C, ciò significa che tutte le funzioni restituiscono void.

Gli attributi della funzione sono definiti dalla variabile macro MQENTRY; il valore di questa variabile macro dipende dall'ambiente.

Parametri con tipo di dati non definito

Le funzioni MQGET, MQPUT e MQPUT1 hanno ognuna un parametro **Buffer** che ha un tipo di dati non definito. Questo parametro viene utilizzato per inviare e ricevere i dati del messaggio dell'applicazione.

I parametri di questo tipo vengono mostrati negli esempi C come array di MQBYTE. È possibile dichiarare i parametri in questo modo, ma è più conveniente dichiararli come la struttura che descrive il layout dei dati nel messaggio. Il parametro della funzione viene dichiarato come un puntatore a void e quindi l'indirizzo di qualsiasi dato può essere specificato come parametro nel richiamo della funzione.

Tipi di dati

Tutti i tipi di dati sono definiti con l'istruzione `typedef` .

Per ogni tipo di dati, viene definito anche il tipo di dati puntatore corrispondente. Il nome del tipo di dati del puntatore è il nome del tipo di dati elementari o della struttura preceduto dalla lettera P per indicare un puntatore. Gli attributi del puntatore sono definiti dalla variabile macro MQPOINTER; il valore di questa variabile macro dipende dall'ambiente. Il seguente codice illustra come dichiarare i tipi di dati del puntatore:

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

Manipolazione di stringhe binarie

Le stringhe di dati binari vengono dichiarate come uno dei tipi di dati MQBYTEn.

Ogni volta che si copiano, confrontano o impostano campi di questo tipo, utilizzare le funzioni C `memcpy`, `memcmpo` `memset`:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

Non utilizzare le funzioni di stringa strcpy, strcmp, strncpy o strncmp perché non funzionano correttamente con i dati dichiarati come MQBYTE24.

Manipolazione delle stringhe di caratteri

Quando il gestore code restituisce i dati carattere all'applicazione, il gestore code riempierà sempre i dati carattere con spazi vuoti fino alla lunghezza definita del campo. Il gestore code non restituisce stringhe con terminazione null, ma è possibile utilizzarle nell'input. Pertanto, durante la copia, il confronto o la concatenazione di tali stringhe, utilizzare le funzioni stringa strncpy, strncmp o strncat.

Non utilizzare le funzioni stringa che richiedono che la stringa termini con un valore null (strcpy, strcmp e strcat). Inoltre, non utilizzare la funzione strlen per determinare la lunghezza della stringa; utilizzare invece la funzione sizeof per determinare la lunghezza del campo.

Valori iniziali per le strutture

Il file di inclusione <cmqc.h> definisce varie variabili macro che è possibile utilizzare per fornire valori iniziali per le strutture quando si dichiarano istanze di tali strutture. Queste variabili macro hanno i nomi nel formato MQxxx_DEFAULT, dove MQxxx rappresenta il nome della struttura. Usali come questo:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

Per alcuni campi di caratteri, l'MQI definisce valori particolari che sono validi (ad esempio, per i campi *StrucId* o per il campo *Format* in MQMD). Per ognuno dei valori validi, vengono fornite due variabili macro:

- Una variabile macro definisce il valore come una stringa con una lunghezza, escluso il valore null implicito, che corrisponde esattamente alla lunghezza definita del campo. Nei seguenti esempi, il simbolo ¬ rappresenta un singolo carattere vuoto:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Utilizzare questo modulo con le funzioni memcpy e memcmp.
- L'altra variabile macro definisce il valore come un array di caratteri; il nome di questa variabile macro è il nome del formato stringa con suffisso _ARRAY. Ad esempio:

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  Utilizzare questo modulo per inizializzare il campo quando un'istanza della struttura viene dichiarata con valori diversi da quelli forniti dalla variabile macro MQMD_DEFAULT.

Valori iniziali per le strutture dinamiche

Quando è richiesto un numero variabile di istanze di una struttura, le istanze vengono generalmente create nella memoria principale ottenuta dinamicamente utilizzando le funzioni calloc o malloc.

Per inizializzare i campi in tali strutture, si raccomanda la seguente tecnica:

1. Dichiarare un'istanza della struttura utilizzando la variabile macro MQxxx_DEFAULT appropriata per inizializzare la struttura. Questa istanza diventa il *modello* per altre istanze:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                    /* declare model instance */
   ```

   Codificare le parole chiave statiche o automatiche sulla dichiarazione per fornire la durata statica o dinamica dell'istanza del modello, come richiesto.

2. Utilizzare le funzioni calloc o malloc per ottenere lo storage per un'istanza dinamica della struttura:

```
PMQMD InstancePtr;
InstancePtr = malloc(sizeof(MQMD));
                                  /* get storage for dynamic instance */
```

3. Utilizzare la funzione memcpy per copiare l'istanza del modello nell'istanza dinamica:

```
memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                  /* initialize dynamic instance */
```

Utilizza da C++

Per il linguaggio di programmazione C + +, i file di intestazione contengono le seguenti istruzioni aggiuntive incluse solo quando viene utilizzato il compilatore C + +:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Codifica in Visual Basic

Informazioni da considerare quando si codificano i programmi IBM MQ in Microsoft Visual Basic. Visual Basic è supportata solo su Windows.

Nota:

Stabilized Da IBM WebSphere MQ 7.0, all'esterno dell'ambiente .NET , il supporto per Visual Basic (VB) è stato stabilizzato al livello IBM WebSphere MQ 6.0 . La maggior parte delle nuove funzioni aggiunte a IBM WebSphere MQ 7.0 o successive non è disponibile per le applicazioni VB. Se si sta programmando in VB.NET, utilizzare le classi IBM MQ per .NET. Per ulteriori informazioni, consultare Sviluppo di applicazioni .NET.

Deprecated Da IBM MQ 9.0, il supporto per Microsoft Visual Basic 6.0 è obsoleto. Le classi IBM MQ per .NET sono la tecnologia sostitutiva consigliata.

Per evitare la conversione non intenzionale dei dati binari tra Visual Basic e IBM MQ, utilizzare una definizione MQBYTE invece di MQSTRING. CMQB.BAS definisce diversi nuovi tipi MQBYTE che sono equivalenti a una definizione di byte C e li utilizza all'interno delle strutture IBM MQ . Ad esempio, per la struttura MQMD (message descriptor), MsgId (message identifier) è definito come MQBYTE24.

Visual Basic non dispone di un tipo di dati puntatore, quindi i riferimenti ad altre strutture dati IBM MQ sono per offset piuttosto che per puntatore. Dichiarare una struttura composta dalle due strutture componenti e specificare la struttura composta nella chiamata. Il supporto IBM MQ per Visual Basic fornisce una chiamata MQCONNXAny per rendere questo possibile e consentire alle applicazioni client di specificare le proprietà del canale su una connessione client. Accetta una struttura non tipizzata (MQCNOCD) al posto della tipica struttura MQCNO.

La struttura MQCNOCD è una struttura composta da un MQCNO seguito da un MQCD. Questa struttura viene dichiarata nel file di intestazione delle uscite CMQXB. Utilizzare la routine MQCNOCD_DEFAULTS per inizializzare una struttura MQCNOCD. Viene fornito un esempio di chiamate MQCONNX (amqscnxb.vbp).

MQCONNXAny ha gli stessi parametri di MQCONNX, ad eccezione del fatto che il parametro **ConnectOpts** è dichiarato come qualsiasi tipo di dati anziché di tipo MQCNO. Ciò consente alla funzione di accettare la struttura MQCNO o MQCNOCD. Questa funzione viene dichiarata nel file di intestazione principale CMQB.

Concetti correlati

“Preparazione dei programmi Visual Basic in Windows” a pagina 1022
Informazioni da considerare quando si utilizzano programmi Microsoft Visual Basic su Windows.

Riferimenti correlati

“Collegamento delle applicazioni Visual Basic con il codice IBM MQ MQI client” a pagina 922
È possibile collegare le applicazioni Microsoft Visual Basic con il codice di IBM MQ MQI client su Windows.

Codifica in COBOL

Prendere nota delle informazioni riportate nella seguente sezione durante la codifica dei programmi IBM MQ in COBOL.

Costanti con nome

I nomi delle costanti vengono visualizzati contenenti il carattere di sottolineatura (_) come parte del nome. In COBOL, è necessario utilizzare il trattino (-) al posto del carattere di sottolineatura. Le costanti che hanno valori stringa di caratteri utilizzano il carattere virgolette singole (') come delimitatore di stringa. Per far sì che il compilatore accetti questo carattere, utilizzare l'opzione APOST del compilatore.

Il file di copia CMQV contiene le dichiarazioni delle costanti denominate come voci level-10 . Per utilizzare le costanti, dichiarare esplicitamente l'elemento level-01 , quindi utilizzare l'istruzione COPY per copiare le dichiarazioni delle costanti:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

Tuttavia, questo metodo fa sì che le costanti occupino la memoria nel programma anche se non vi si fa riferimento. Se le costanti sono incluse in molti programmi separati all'interno della stessa unità di esecuzione, esisteranno più copie delle costanti; ciò potrebbe comportare l'utilizzo di una quantità significativa di memoria principale. È possibile evitare ciò aggiungendo la clausola GLOBAL alla dichiarazione level-01 :

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

Questo assegna la memoria solo per *una* serie di costanti all'interno dell'unità di esecuzione; le costanti, tuttavia, possono essere indicate da *qualsiasi* programma all'interno dell'unità di esecuzione, non solo dal programma che contiene la dichiarazione level-01 .

Garantire l'allineamento della struttura

È necessario assicurarsi che le strutture IBM MQ trasmesse per l'avvio sulla chiamata MQ siano allineate sui limiti delle parole. Un limite di parola è di 4 byte per i processi a 32 bit, 8 byte per i processi a 64 bit e 16 byte per i processi a 128 bit (IBM i).

Dove possibile, posizionare tutte le strutture IBM MQ insieme in modo che siano tutte allineate ai limiti.

Codifica in linguaggio assembler System/390 (Message queue interface)

Notare le informazioni nelle seguenti sezioni quando si codificano i programmi IBM MQ for z/OS in linguaggio assembler.

- “Nomi” a pagina 1057
- “Utilizzo delle chiamate MQI” a pagina 1057
- “Dichiarazione delle costanti” a pagina 1057
- “Specifica del nome di una struttura” a pagina 1058

- “Specifica della forma di una struttura” a pagina 1058
- “Controllo dell'elenco” a pagina 1058
- “Specifica dei valori iniziali per i campi” a pagina 1058
- “Scrittura di programmi reimscrivibili” a pagina 1059
- “Utilizzo di CEDF” a pagina 1059

Nomi

I nomi dei parametri nelle descrizioni delle chiamate e i nomi dei campi nelle descrizioni delle strutture vengono mostrati in maiuscolo e minuscolo. Nelle macro del linguaggio assembler fornite con IBM MQ, tutti i nomi sono in maiuscolo.

Utilizzo delle chiamate MQI

La MQI è un'interfaccia di chiamata, quindi i programmi in linguaggio assembler devono osservare la convenzione di collegamento SO.

In particolare, prima di emettere una chiamata MQI, i programmi in linguaggio assembler devono puntare il registro R13 ad un'area di salvataggio di almeno 18 parole complete. Questa area di salvataggio fornisce la memoria per il programma richiamato. Memorizza i registri del chiamante prima che il loro contenuto venga distrutto e ripristina il contenuto dei registri del chiamante al ritorno.

Nota: Ciò è importante per i programmi in linguaggio assembler CICS che utilizzano la macro DFHEIENT per configurare la memoria dinamica, ma che scelgono di sovrascrivere il DATAREG predefinito da R13 ad altri registri. Quando l'interfaccia di CICS Resource Manager riceve il controllo dallo stub, salva il contenuto corrente dei registri all'indirizzo indicato da R13 . La mancata prenotazione di un'area di salvataggio per questo scopo fornisce risultati imprevedibili e probabilmente causerà una fine anomala in CICS.

Dichiarazione delle costanti

La maggior parte delle costanti sono dichiarate come equivalenze nella macro CMQA.

Tuttavia, le seguenti costanti non possono essere definite come equivalenze e non vengono incluse quando si richiama la macro utilizzando le opzioni predefinite:

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- MMQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- EVENTO MQFMT
- IMS MQFMT
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

Per includerli, aggiungere la parola chiave EQUONLY=NO quando si richiama la macro.

CMQA è protetto contro più dichiarazioni, quindi è possibile includerlo molte volte. Tuttavia, la parola chiave EQUONLY ha effetto solo la prima volta che la macro viene inclusa.

Specifica del nome di una struttura

Per consentire la dichiarazione di più di un'istanza di una struttura, la macro che genera la struttura antepone il nome di ciascun campo con una stringa specificabile dall'utente e un carattere di sottolineatura (_).

Specificare la stringa quando si richiama la macro. Se non si specifica una stringa, la macro utilizza il nome della struttura per costruire il prefisso:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

Le dichiarazioni di struttura in Descrizioni chiamata mostrano il prefisso predefinito.

Specifica della forma di una struttura

Le macro possono creare dichiarazioni di struttura in due formati, controllati dal parametro DSECT:

DSECT = SÌ
Un'istruzione DSECT in linguaggio assembler viene utilizzata per avviare una nuova sezione dati; la definizione della struttura segue immediatamente l'istruzione DSECT. Non è stata assegnata alcuna memoria, quindi non è possibile alcuna inizializzazione. L'etichetta sul richiamo della macro viene utilizzata come nome della sezione dati; se non viene specificata alcuna etichetta, viene utilizzato il nome della struttura.

DSECT = NO
Le istruzioni DC in linguaggio Assembler vengono utilizzate per definire la struttura nella posizione corrente nella routine. I campi vengono inizializzati con valori, che è possibile specificare codificando i parametri rilevanti sul richiamo della macro. I campi per cui non è specificato alcun valore nel richiamo della macro vengono inizializzati con valori predefiniti.

DSECT = NO viene assunto se il parametro DSECT non è specificato.

Controllo dell'elenco

È possibile controllare l'aspetto della dichiarazione di struttura nell'elenco del linguaggio assembler con il parametro LIST:

ELENCO = SÌ
La dichiarazione di struttura viene visualizzata nell'elenco del linguaggio assembler.

ELENCO = NO
La dichiarazione di struttura non viene visualizzata nell'elenco del linguaggio assembler. Ciò viene assunto se non si specifica il parametro LIST.

Specifica dei valori iniziali per i campi

È possibile specificare il valore da utilizzare per inizializzare un campo in una struttura codificando il nome di tale campo (senza il prefisso) come parametro sul richiamo della macro, accompagnato dal valore richiesto.

Ad esempio, per dichiarare una struttura del descrittore del messaggio con il campo *MsgType* inizializzato con MQMT_REQUEST e il campo *ReplyToQ* inizializzato con la stringa MY_REPLY_TO_QUEUE, utilizzare il seguente codice:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

Se si specifica una costante con nome (o un'equiparazione) come valore nel richiamo della macro, utilizzare la macro CMQA per definire la costante con nome. Non racchiudere tra virgolette singole ('') i valori che sono stringhe di caratteri.

Scrittura di programmi reimscrivibili

IBM MQ utilizza le sue strutture sia per l'input che per l'output. Se si desidera che il proprio programma rimanga reimmesso:

1. Definire le versioni di memoria di lavoro delle strutture come DSECT o definire le strutture in linea all'interno di un DSECT già definito. Quindi, copiare il DSECT nella memoria ottenuta utilizzando:
   - Per i programmi batch e TSO, le macro assembler STORAGE o GETMAIN z/OS
   - Per CICS, il DSECT di memoria di lavoro (DFHEISTG) o il comando EXEC CICS GETMAIN

   Per inizializzare correttamente queste strutture di memoria di lavoro, copiare una versione costante della struttura corrispondente nella versione di memoria di lavoro.

   Nota: Le strutture MQMD e MQXQH hanno una lunghezza superiore a 256 byte. Per copiare queste strutture nella memoria, utilizzare l'istruzione assembler MVCL.
2. Riservare spazio in memoria utilizzando il modulo ELENCA (`MF=L`) della macro CALL. Quando si utilizza la macro CALL per effettuare una chiamata MQI, utilizzare il modulo EXECUTE (`MF=E`) della macro, utilizzando la memoria riservata in precedenza, come mostrato nell'esempio in “Utilizzo di CEDF” a pagina 1059. Per ulteriori esempi su come eseguire questa operazione, consultare i programmi di esempio del linguaggio assembler forniti con IBM MQ.

Utilizzare l'opzione RENT del linguaggio assembler per determinare se il programma può essere reimmesso.

Per informazioni sulla scrittura di programmi riutilizzabili, consultare *z/OS MVS Application Development Guide: Assembler Language Programs*.

Utilizzo di CEDF

Se si desidera utilizzare la transazione fornita da CICS, CEDF (CICS Execution Diagnostic Facility) per eseguire il debug del programma, aggiungere la parola chiave `,VL` a ciascuna istruzione `CALL` , ad esempio:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

L'esempio precedente è un codice di linguaggio assembler riutilizzabile, dove PARMAREA è un'area nella memoria di lavoro specificata.

Utilizzo delle chiamate MQI

La MQI è un'interfaccia di chiamata, quindi i programmi in linguaggio assembler devono osservare la convenzione di collegamento SO. In particolare, prima di emettere una chiamata MQI, i programmi in linguaggio assembler devono puntare il registro R13 ad un'area di salvataggio di almeno 18 parole complete. Questa area di salvataggio fornisce la memoria per il programma richiamato. Memorizza i registri del chiamante prima che il loro contenuto venga distrutto e ripristina il contenuto dei registri del chiamante al ritorno.

Nota: Ciò è importante per i programmi in linguaggio assembler CICS che utilizzano la macro DFHEIENT per configurare la memoria dinamica, ma che scelgono di sovrascrivere il DATAREG predefinito da R13 ad altri registri. Quando l'interfaccia di CICS Resource Manager riceve il controllo dallo stub, salva il contenuto corrente dei registri all'indirizzo indicato da R13 . La mancata prenotazione di un'area di salvataggio appropriata per questo scopo fornisce risultati imprevedibili e probabilmente causerà una fine anomala in CICS.

IBM i Codifica di programmi IBM MQ in RPG (soloIBM i)

Nella documentazione di IBM MQ , i parametri delle chiamate, i nomi dei tipi di dati, i campi delle strutture e i nomi delle costanti sono tutti descritti utilizzando i rispettivi nomi estesi. In RPG, questi nomi sono abbreviati in sei o meno caratteri maiuscoli.

Ad esempio, il campo *MsgType* diventa *MDMT* in RPG. Per ulteriori informazioni, consultare IBM i Application Programming Reference (ILE/RPG).

Codifica in PL/I (soloz/OS)

Informazioni utili quando si codifica per IBM MQ in PL/I.

Strutture

Le strutture vengono dichiarate con l'attributo BASED e quindi non occupano alcuna memoria a meno che il programma non dichiari una o più istanze di una struttura.

Un'istanza di una struttura può essere dichiarata utilizzando l'attributo `like` , ad esempio:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

I campi della struttura vengono dichiarati con l'attributo INITIAL; quando l'attributo `like` viene utilizzato per dichiarare una istanza di una struttura, tale istanza eredita i valori iniziali definiti per tale struttura. È necessario impostare solo i campi in cui il valore richiesto è diverso dal valore iniziale.

PL/I non è sensibile al maiuscolo / minuscolo e quindi i nomi delle chiamate, dei campi di struttura e delle costanti possono essere codificati in lettere minuscole, maiuscole o maiuscole / minuscole.

Costanti con nome

Le costanti denominate vengono dichiarate come variabili macro; di conseguenza, le costanti denominate a cui il programma non fa riferimento non occupano alcuna memoria nella procedura compilata.

Tuttavia, l'opzione del compilatore che fa sì che l'origine venga elaborata dal preprocessore macro deve essere specificata quando il programma viene compilato.

Tutte le variabili macro sono variabili carattere, anche quelle che rappresentano valori numerici. Sebbene ciò possa sembrare controintuitivo, non risulta in alcun conflitto di tipo di dati dopo che le variabili macro sono state sostituite dal processore macro, ad esempio:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

Utilizzo dei programmi procedurali di esempio IBM MQ

Questi programmi di esempio sono scritti in linguaggi procedurali e dimostrano gli usi tipici di MQI (Message Queue Interface). Programmi IBM MQ su diverse piattaforme.

Informazioni su questa attività

Esistono due serie di campioni:

- Programmi di esempio per sistemi distribuiti e IBM i.
- Programmi di esempio per z/OS.

Procedura

- Utilizzare i seguenti collegamenti per ulteriori informazioni sui programmi di esempio:
  - "Utilizzo dei programmi di esempio su Multiplatforms" a pagina 1061
  - z/OS "Utilizzo dei programmi di esempio per z/OS" a pagina 1164

Concetti correlati

"Concetti dello sviluppo di applicazioni" a pagina 7
È possibile utilizzare una scelta di linguaggi procedurali o orientati agli oggetti per scrivere applicazioni IBM MQ . Prima di iniziare a progettare e scrivere le tue applicazioni IBM MQ , familiarizza con i concetti IBM MQ di base.

"Sviluppo di applicazioni per IBM MQ" a pagina 5
È possibile sviluppare applicazioni per inviare e ricevere messaggi e per gestire i gestori code e le relative risorse. IBM MQ supporta applicazioni scritte in diversi linguaggi e framework.

"Considerazioni sulla progettazione per applicazioni IBM MQ" a pagina 51
Una volta stabilito in che modo le applicazioni possono trarre vantaggio dalle piattaforme e dagli ambienti disponibili, è necessario decidere come utilizzare le funzioni offerte da IBM MQ.

"Scrittura di un'applicazione procedurale per l'accodamento" a pagina 721
Utilizzare queste informazioni per informazioni sulla scrittura delle applicazioni di accodamento, la connessione e la disconnessione da un gestore code, la pubblicazione / sottoscrizione e l'apertura e la chiusura di oggetti.

"Scrittura di applicazioni procedurali client" a pagina 915
Cosa devi sapere per scrivere le applicazioni client su IBM MQ utilizzando un linguaggio procedurale.

"Scrittura di applicazioni di pubblicazione / sottoscrizione" a pagina 809
Avviare la scrittura delle applicazioni IBM MQ di pubblicazione / sottoscrizione.

"Creazione di un'applicazione procedurale" a pagina 1002
È possibile scrivere un'applicazione IBM MQ in uno dei diversi linguaggi procedurali ed eseguire l'applicazione su diverse piattaforme.

"Gestione degli errori del programma procedurale" a pagina 1040
Queste informazioni spiegano gli errori associati alle chiamate MQI delle applicazioni quando effettua una chiamata o quando il suo messaggio viene consegnato alla destinazione finale.

Multi Utilizzo dei programmi di esempio su Multiplatforms

Questi programmi procedurali di esempio vengono forniti con il prodotto. Gli esempi sono scritti in C e COBOL e dimostrano gli utilizzi tipici di MQI (Message Queue Interface).

Informazioni su questa attività

Gli esempi non sono destinati a dimostrare tecniche di programmazione generali, quindi viene omesso il controllo degli errori che si potrebbe voler includere in un programma di produzione.

Il codice sorgente per tutti gli esempi viene fornito con il prodotto; questa sorgente include commenti che spiegano le tecniche di accodamento dei messaggi dimostrate nei programmi.

IBM i Per la programmazione RPG, consultare IBM i Application Programming Reference (ILE/RPG).

I nomi degli esempi iniziano con il prefisso `amq`. Il quarto carattere indica il linguaggio di programmazione e, se necessario, il compilatore:

- s: linguaggio C
- 0: linguaggio COBOL su compilatori IBM e Micro Focus
- i: linguaggio COBOL solo su compilatori IBM
- m: linguaggio COBOL solo su compilatori Micro Focus

L'ottavo carattere dell'eseguibile indica se l'esempio viene eseguito in modalità di collegamento locale o in modalità client. Se non è presente un ottavo carattere, l'esempio viene eseguito in modalità bind locale. Se l'ottavo carattere è 'c', l'esempio viene eseguito in modalità client.

Prima di poter eseguire le applicazioni di esempio, è necessario creare e configurare un gestore code. Per impostare il gestore code per accettare connessioni client, consultare “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072.

Procedura

- Utilizzare i seguenti collegamenti per ulteriori informazioni sui programmi di esempio:
  - “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063
  - “Preparazione ed esecuzione dei programmi di esempio” a pagina 1072
  - “Il programma di esempio di uscita API” a pagina 1078
  - “Il programma di esempio di consumo asincrono” a pagina 1079
  - “Il programma di esempio Put asincrono” a pagina 1080
  - “Programmi di esempio Sfoglia” a pagina 1081
  - “Il programma di esempio Browser” a pagina 1082
  - “L'esempio di transazione CICS” a pagina 1084
  - “Il programma di esempio Connect” a pagina 1084
  - “Il programma di esempio Conversione dati” a pagina 1085
  - “Esempi di coordinamento database” a pagina 1086
  - “Esempio di gestore code di messaggi non instradabili” a pagina 1093
  - “Programma di esempio Elenco di distribuzione” a pagina 1093
  - “I programmi di esempio Echo” a pagina 1094
  - “I programmi di esempio Get” a pagina 1095
  - “Programmi di esempio HA (High Availability)” a pagina 1097
  - “I programmi di esempio Inquire” a pagina 1101
  - “Il programma di esempio Inquire Properties of a Message Handle” a pagina 1102
  - “Programmi di esempio di pubblicazione / sottoscrizione” a pagina 1102
  - “Il programma di esempio Pubblica uscita” a pagina 1106
  - “Programmi di esempio Put” a pagina 1107
  - “Programmi di esempio del messaggio di riferimento” a pagina 1109
  - “I programmi di esempio Richiesta” a pagina 1116
  - “I programmi di esempio Set” a pagina 1121
  - “Il programma di esempio TLS” a pagina 1122
  - “I programmi di esempio Trigger” a pagina 1125
  - “Utilizzo degli esempi di TUXEDO su AIX, Linux, and Windows” a pagina 1127
  - “Utilizzo dell'uscita di sicurezza SSPI su Windows” a pagina 1137
  - “Esecuzione degli esempi utilizzando le code remote” a pagina 1138
  - “Programma di esempio Monitoraggio coda cluster (AMQSCLM)” a pagina 1138
  - “Programma di esempio per Connection Endpoint Lookup (CEPL)” a pagina 1148

Concetti correlati

“Programmi di esempio C++” a pagina 531
Vengono forniti quattro programmi di esempio, per dimostrare la ricezione e l'inserimento di messaggi.

Attività correlate

“Utilizzo dei programmi di esempio per z/OS” a pagina 1164
Le applicazioni procedurali di esempio fornite con IBM MQ for z/OS dimostrano gli utilizzi tipici di MQI (Message Queue Interface).

Multi *Funzioni dimostrate nei programmi di esempio su Multiplatforms*

Una raccolta di tabelle che mostra le tecniche dimostrate dai programmi di esempio IBM MQ .

Tutti gli esempi aprono e chiudono code utilizzando le chiamate MQOPEN e MQCLOSE, quindi queste tecniche non sono elencate separatamente nelle tabelle. Vedere l'intestazione che include la piattaforma a cui si è interessati.

z/OS Per la piattaforma z/OS , consultare “Utilizzo dei programmi di esempio per z/OS” a pagina 1164.

Linux AIX *Esempi per sistemi AIX and Linux*

Le tecniche dimostrate dai programmi di esempio per IBM MQ for AIX or Linux.

Consultare “Preparazione ed esecuzione di programmi di esempio su AIX and Linux” a pagina 1076 per scoprire dove sono archiviati i programmi di esempio per IBM MQ for AIX or Linux .

Tabella 156 a pagina 1063 La tabella elenca i file di origine C e COBOL forniti e se è incluso un server o un client eseguibile.

Tabella 156. Programmi di esempio che dimostrano l'utilizzo di MQ (C e COBOL) su AIX and Linux ..

Una tabella con quattro colonne. Le prime colonne elencano le tecniche dimostrate dai campioni. La seconda colonna elenca gli esempi C e la terza colonna elenca gli esempi COBOL che dimostrano ciascuna delle tecniche elencate nella prima colonna. La quarta colonna indica se un eseguibile C del server è incluso o meno e la quinta colonna indica se un eseguibile C del client è incluso o meno.

Tecnica	**C (origine) (“1” a pagina 1066)**	**COBOL (origine) (“2” a pagina 1066)**	**Server (eseguibile C)**	**Client (eseguibile C)**
Utilizzo dell'interfaccia di pubblicazione / sottoscrizione	amqspuba amqssuba amqssbxa	nessun campione	amqspub amqssub amqssbx	nessun campione
Inserimento di messaggi utilizzando la chiamata MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Inserimento di un singolo messaggio utilizzando la chiamata MQPUT1	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
Inserimento di messaggi in un elenco di distribuzione (“3” a pagina 1066)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Risposta a un messaggio di richiesta	amqsinqa	amqminqx amqiinqx	amqsinq	nessun campione
Ricezione dei messaggi utilizzando la ricerca (nessuna attesa)	amqsgbr0	amq0gbr0	amqsgbr	nessun campione
Ricezione dei messaggi (attendere con un limite di tempo)	amqsget0	amq0get0	amqsget	amqsgetc
Ricezione messaggi (attesa illimitata)	amqstrg0	nessun campione	amqstrg	amqstrgc

Tabella 156. Programmi di esempio che dimostrano l'utilizzo di MQ (C e COBOL) su AIX and Linux ..

Una tabella con quattro colonne. Le prime colonne elencano le tecniche dimostrate dai campioni. La seconda colonna elenca gli esempi C e la terza colonna elenca gli esempi COBOL che dimostrano ciascuna delle tecniche elencate nella prima colonna. La quarta colonna indica se un eseguibile C del server è incluso o meno e la quinta colonna indica se un eseguibile C del client è incluso o meno.

(Continua)

Tecnica	**C (origine) (“1” a pagina 1066)**	**COBOL (origine) (“2” a pagina 1066)**	**Server (eseguibile C)**	**Client (eseguibile C)**
Ricezione di messaggi (con conversione dati)	amqsecha	nessun campione	amqsech	nessun campione
Inserimento di messaggi di riferimento in una coda (“3” a pagina 1066)	amqsprma	nessun campione	amqsprm	amqsprmc
Richiamo dei messaggi di riferimento da una coda (“3” a pagina 1066)	amqsgrma	nessun campione	amqsgrm	amqsgrmc
Uscita canale messaggi di riferimento (“3” a pagina 1066)	amqsqrma amqsxrma	nessun campione	amqsxrm	nessun campione
Ricerca dei primi 20 caratteri di un messaggio	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Visualizzazione dei messaggi completi	amqsbcg0	nessun campione	amqsbcg	amqsbcgc
Utilizzo di una coda di input condivisa	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
Utilizzo di una coda di immissione esclusiva	amqstrg0	amq0req0	amqstrg	amqstrgc
Utilizzo della chiamata MQINQ	amqsinqa	amqminqx amqiinqx	amqsinq	nessun campione
Utilizzo della chiamata MQSET	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
Utilizzo di una coda di risposta	amqsreq0	amq0req0	amqsreq	amqsreqc
Richiesta di eccezioni ai messaggi	amqsreq0	amq0req0	amqsreq	nessun campione
Accettazione di un messaggio troncato	amqsgbr0	amq0gbr0	amqsgbr	nessun campione
Utilizzo di un nome coda risolto	amqsgbr0	amq0gbr0	amqsgbr	nessun campione
Attivazione di un processo	amqstrg0	nessun campione	amqstrg	amqstrgc
Utilizzo della conversione dati	(“4” a pagina 1066)	nessun campione	nessun campione	nessun campione
IBM MQ (coordinando i gestori database compatibili con XA) che accedono a un singolo database utilizzando SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	nessun campione	nessun campione

Tabella 156. Programmi di esempio che dimostrano l'utilizzo di MQ (C e COBOL) su AIX and Linux ..

Una tabella con quattro colonne. Le prime colonne elencano le tecniche dimostrate dai campioni. La seconda colonna elenca gli esempi C e la terza colonna elenca gli esempi COBOL che dimostrano ciascuna delle tecniche elencate nella prima colonna. La quarta colonna indica se un eseguibile C del server è incluso o meno e la quinta colonna indica se un eseguibile C del client è incluso o meno.

(Continua)

Tecnica	**C (origine) (“1” a pagina 1066)**	**COBOL (origine) (“2” a pagina 1066)**	**Server (eseguibile C)**	**Client (eseguibile C)**
IBM MQ (coordinando gestori database compatibili con XA) che accedono a due database utilizzando SQL	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	nessun campione	nessun campione
Transazione CICS (“5” a pagina 1066)	amqscic0.ccs	nessun campione	amqscic0	nessun campione
Transazione Encina (“3” a pagina 1066)	amqsxae0	nessun campione	amqsxae0	nessun campione
Transazione TUXEDO per inserire messaggi “6” a pagina 1066)	amqstxpx	nessun campione	nessun campione	nessun campione
Transazione TUXEDO per richiamare i messaggi (“6” a pagina 1066)	amqstxgx	nessun campione	nessun campione	nessun campione
Server per TUXEDO (“6” a pagina 1066)	amqstxsx	nessun campione	nessun campione	nessun campione
gestore coda di messaggi non instradabili (DLQ, dead-letter queue)	Directory `./tools/c/Samples/dlq` (“7” a pagina 1066)	nessun campione	amqsdlq	nessun campione
Da un client MQI, inserimento di un messaggio	nessun campione	nessun campione	nessun campione	amqsputc
Da un client MQI, ricezione di un messaggio	nessun campione	nessun campione	nessun campione	amqsgetc
Connessione al gestore code mediante MQCONNX	amqscnxc	nessun campione	nessun campione	amqscnxc
Utilizzo delle uscite API	amqsaxe0	nessun campione	amqsaxe	nessun campione
Uscita bilanciamento carico di lavoro cluster	amqswlm0	nessun campione	amqswlm	nessun campione
Inserimento dei messaggi in modo asincrono e richiamo dello stato utilizzando la chiamata MQSTAT	amqsapt0	nessun campione	amqsapt	amqsaptc
Client riconnettibili	amqsphac amqsghac amqsmhac	nessun campione	non applicabile	amqsphac amqsghac amqsmhac

Tabella 156. Programmi di esempio che dimostrano l'utilizzo di MQ (C e COBOL) su AIX and Linux ..

Una tabella con quattro colonne. Le prime colonne elencano le tecniche dimostrate dai campioni. La seconda colonna elenca gli esempi C e la terza colonna elenca gli esempi COBOL che dimostrano ciascuna delle tecniche elencate nella prima colonna. La quarta colonna indica se un eseguibile C del server è incluso o meno e la quinta colonna indica se un eseguibile C del client è incluso o meno.

(Continua)

Tecnica	C (origine) (“1” a pagina 1066)	COBOL (origine) (“2” a pagina 1066)	Server (eseguibile C)	Client (eseguibile C)
Utilizzo dei destinatari dei messaggi per l'utilizzo asincrono dei messaggi da più code	amqscbf0	nessun campione	amqscbf	amqscbfc
Specifica delle informazioni di connessione TLS su MQCONNX	amqssslc	nessun campione	non applicabile	amqssslc

Note:

1. La versione eseguibile degli esempi IBM MQ MQI client condividono la stessa origine degli esempi eseguiti in un ambiente server.
2. Compilare i programmi che iniziano con 'amqm' con il compilatore Micro Focus COBOL, quelli che iniziano con 'amqi' con il compilatore IBM COBOL e quelli che iniziano con 'amq0'.
3. AIX Supportato solo su IBM MQ for AIX .
4. AIX Su IBM MQ for AIX questo programma è denominato `amqsvfc0.c`
5. AIX CICS è supportato solo da IBM MQ for AIX .
6. Linux TUXEDO non è supportato da IBM MQ per Linux su System p.
7. L'origine per il gestore code di messaggi non recapitabili è costituita da diversi file ed è fornita in una directory separata.

Per informazioni dettagliate sul supporto per i sistemi AIX and Linux , consultare Requisiti di sistema per IBM MQ.

Windows *Esempi per IBM MQ for Windows*

Le tecniche dimostrate dai programmi di esempio per IBM MQ for Windows.

Tabella 157 a pagina 1066 elenca i file di origine C e COBOL forniti e se è incluso un server o un client eseguibile.

Tabella 157. Programmi di esempio IBM MQ for Windows che dimostrano l'utilizzo di MQI (C e COBOL)

Tecnica	C (origine)	COBOL (origine)	Server (eseguibile C)	Client (eseguibile C)
Utilizzo dell'interfaccia di pubblicazione / sottoscrizione	amqspuba amqssuba amqssbxa	nessun campione	amqspub amqssub amqssbx	nessun campione
Inserimento di messaggi utilizzando la chiamata MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Inserimento di un singolo messaggio utilizzando la chiamata MQPUT1	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc

Tabella 157. Programmi di esempio IBM MQ for Windows che dimostrano l'utilizzo di MQI (C e COBOL) (Continua)

Tecnica	C (origine)	COBOL (origine)	Server (eseguibile C)	Client (eseguibile C)
Inserimento di messaggi in un elenco di distribuzione	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Risposta a un messaggio di richiesta	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Ricezione messaggi (nessuna attesa)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Ricezione dei messaggi (attendere con un limite di tempo)	amqsget0	amq0get0	amqsget	amqsgetc
Ricezione messaggi (attesa illimitata)	amqstrg0	nessun campione	amqstrg	amqstrgc
Ricezione di messaggi (con conversione dati)	amqsecha	nessun campione	amqsech	amqsechc
Inserimento di messaggi di riferimento in una coda	amqsprma	nessun campione	amqsprm	amqsprmc
Richiamo dei messaggi di riferimento da una coda	amqsgrma	nessun campione	amqsgrm	amqsgrmc
Uscita canale messaggi di riferimento	amqsqrma amqsxrma	nessun campione	amqsxrm	nessun campione
Ricerca dei primi 20 caratteri di un messaggio	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Visualizzazione dei messaggi completi	amqsbcg0	nessun campione	amqsbcg	amqsbcgc
Utilizzo di una coda di input condivisa	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Utilizzo di una coda di immissione esclusiva	amqstrg0	amq0req0	amqstrg	amqstrgc
Utilizzo della chiamata MQINQ	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Utilizzo della chiamata MQSET	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
Utilizzo della chiamata MQINQMP	amqsiqma	nessun campione	nessun campione	nessun campione
Utilizzo di una coda di risposta	amqsreq0	amq0req0	amqsreq	amqsreqc
Richiesta di eccezioni ai messaggi	amqsreq0	amq0req0	amqsreq	amqsreqc
Accettazione di un messaggio troncato	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Utilizzo di un nome coda risolto	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Attivazione di un processo	amqstrg0	nessun campione	amqstrg	amqstrgc
Utilizzo della conversione dati	amqsvfc0	nessun campione	nessun campione	nessun campione

Tabella 157. Programmi di esempio IBM MQ for Windows che dimostrano l'utilizzo di MQI (C e COBOL) (Continua)

Tecnica	C (origine)	COBOL (origine)	Server (eseguibile C)	Client (eseguibile C)
IBM MQ (coordinando i gestori database compatibili con XA) che accedono a un singolo database utilizzando SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	nessun campione	nessun campione
IBM MQ (coordinando gestori database compatibili con XA) che accedono a due database utilizzando SQL	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	nessun campione	nessun campione
Transazione TUXEDO per inserire messaggi	amqstxpx	nessun campione	nessun campione	nessun campione
Transazione TUXEDO per richiamare i messaggi	amqstxgx	nessun campione	nessun campione	nessun campione
Server per TUXEDO	amqstxsx	nessun campione	nessun campione	nessun campione
gestore coda di messaggi non instradabili (DLQ, dead-letter queue)	Directory `./tools/c/Samples/dlq` ("1" a pagina 1069)	nessun campione	amqsdlq	nessun campione
Da un IBM MQ MQI client, inserimento di un messaggio	nessun campione	nessun campione	nessun campione	amqsputc
Da un IBM MQ MQI client, ottenendo un messaggio	nessun campione	nessun campione	nessun campione	amqsgetc
Connessione al gestore code mediante MQCONNX	amqscnxc	nessun campione	nessun campione	amqscnxc
Utilizzo delle uscite API	amqsaxe0	nessun campione	amqsaxe	nessun campione
Bilanciamento carico di lavoro cluster	amqswlm0	nessun campione	amqswlm	nessun campione
Routine di sicurezza SSPI	amqsspin	nessun campione	amqrspin.dll	amqrspin.dll
Inserimento dei messaggi in modo asincrono e richiamo dello stato utilizzando la chiamata MQSTAT	amqsapt0	nessun campione	amqsapt	amqsaptc
Client riconnettibili	amqsphac amqsghac amqsmhac	nessun campione	Non applicabile	amqsphac amqsghac amqsmhac

Tabella 157. Programmi di esempio IBM MQ for Windows che dimostrano l'utilizzo di MQI (C e COBOL) (Continua)				
Tecnica	**C (origine)**	**COBOL (origine)**	**Server (eseguibile C)**	**Client (eseguibile C)**
Utilizzo dei destinatari dei messaggi per l'utilizzo asincrono dei messaggi da più code	amqscbf0	nessun campione	amqscbf	amqscbfc
Specifica delle informazioni di connessione TLS su MQCONNX	amqssslc	nessun campione	non applicabile	amqssslc

Note:

1. L'origine per il gestore code di messaggi non recapitabili è costituita da diversi file ed è fornita in una directory separata.

Windows *Esempi Visual Basic per IBM MQ for Windows*

Le tecniche dimostrati dai programmi di esempio per IBM MQ su sistemi Windows .

Tabella 158 a pagina 1069 mostra le tecniche dimostrate dai IBM MQ for Windows programmi di esempio.

Un progetto può contenere diversi file. Quando si apre un progetto in Visual Basic, gli altri file vengono caricati automaticamente. Non viene fornito alcun programma eseguibile.

Tutti i progetti di esempio, eccetto mqtrivc.vbp, sono configurati per funzionare con il server IBM MQ . Per informazioni su come modificare i progetti di esempio per utilizzare i client IBM MQ , vedere “Preparazione dei programmi Visual Basic in Windows” a pagina 1022.

Tabella 158. Programmi di esempio IBM MQ for Windows che dimostrano l'utilizzo di MQI (Visual Basic)	
Tecnica	**Nome file di progetto**
Inserimento di messaggi utilizzando la chiamata MQPUT	amqsputb.vbp
Richiamo dei messaggi utilizzando la chiamata MQGET	amqsgetb.vbp
Esplorazione di una coda utilizzando la chiamata MQGET	amqsbcgb.vbp
Esempio MQGET e MQPUT semplice (client)	mqtrivc.vbp
Esempio MQGET e MQPUT semplice (server)	mqtrivs.vbp
Inserimento e ottenimento di stringhe e strutture definite dall'utente utilizzando MQPUT e MQGET	strings.vbp
Utilizzo di strutture PCF per avviare e arrestare un canale	pcfsamp.vbp
Creazione di una coda utilizzando MQAI	amqsaicq.vbp
Elenco delle code di un gestore code utilizzando MQAI	amqsailq.vbp
Monitoraggio degli eventi utilizzando MQAI	amqsaiem.vbp

IBM i *Esempi per IBM i*

Le tecniche dimostrati dai programmi di esempio per IBM MQ su sistemi IBM i .

Tabella 159 a pagina 1070 mostra le tecniche dimostrate dai IBM MQ for IBM i programmi di esempio. Alcune tecniche si verificano in più di un programma di esempio, ma solo un programma è elencato nella tabella.

Tabella 159. Programmi di esempio che dimostrano l'uso di MQ (C e COBOL) su IBM i

Tecnica	**C (origine) ("1" a pagina 1071)**	**COBOL (origine) ("2" a pagina 1071)**	**RPG (origine) ("3" a pagina 1071)**	**Client (eseguibile C) (4)**
Inserimento di messaggi utilizzando la chiamata MQPUT	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
Inserimento di messaggi da un file di dati utilizzando la chiamata MQPUT	AMQSPUT4	nessun campione	nessun campione	nessun campione
Inserimento di un singolo messaggio utilizzando la chiamata MQPUT1	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
Inserimento di messaggi in un elenco di distribuzione	AMQSPTL4	nessun campione	nessun campione	AMQSPTLC
Risposta a un messaggio di richiesta	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Ricezione messaggi (nessuna attesa)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	BRCAMQSGC
Ricezione dei messaggi (attendere con un limite di tempo)	AMQSGET4	AMQ0GET4	AMQ3GET4	GETC AMQS
Ricezione messaggi (attesa illimitata)	AMQSTRG4	nessun campione	AMQ3TRG4	AMQSTRGC
Ricezione di messaggi (con conversione dati)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
Inserimento di messaggi di riferimento in una coda	AMQSPRM4	nessun campione	nessun campione	MMQSPRMC
Richiamo dei messaggi di riferimento da una coda	AMQSGRM4	nessun campione	nessun campione	GRMAMQS
Uscita canale messaggi di riferimento	AMQSQRM4, AMQSXRM4	nessun campione	nessun campione	nessun esempio
Uscita messaggi	AMQSCMX4	nessun campione	nessun campione	nessun esempio
Ricerca dei primi 49 caratteri di un messaggio	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	BRCAMQSGC
Visualizzazione dei messaggi completi	AMQSBCG4	nessun campione	nessun campione	AMQSBCGC
Utilizzo di una coda di input condivisa	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Utilizzo di una coda di immissione esclusiva	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Utilizzo della chiamata MQINQ	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Utilizzo della chiamata MQSET	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSEC
Utilizzo di una coda di risposta	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Richiesta di eccezioni ai messaggi	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Accettazione di un messaggio troncato	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	BRCAMQSGC
Utilizzo di un nome coda risolto	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	BRCAMQSGC
Attivazione di un processo	AMQSTRG4	nessun campione	AMQ3TRG4	AMQSTRGC

Tabella 159. Programmi di esempio che dimostrano l'uso di MQ (C e COBOL) su IBM i (Continua)

Tecnica	**C (origine) ("1" a pagina 1071)**	**COBOL (origine) ("2" a pagina 1071)**	**RPG (origine) ("3" a pagina 1071)**	**Client (eseguibile C) (4)**
Server trigger	AMQSERV4	nessun campione	AMQ3SRV4	nessun campione
Utilizzo di un server di trigger (incluse le transazioni CICS)	AMQSERV4	nessun campione	AMQ3SRV4	nessun campione
Utilizzo della conversione dati	AMQSVFC4	nessun campione	nessun campione	nessun campione
Utilizzo delle uscite API	AMQSAXE0	nessun campione	nessun campione	nessun campione
Bilanciamento carico di lavoro cluster	AMQSWLM0	nessun campione	nessun campione	nessun campione
Inserimento dei messaggi in modo asincrono e richiamo dello stato utilizzando la chiamata MQSTAT	AMQSAPT0	nessun campione	nessun campione	AMQSAPC
Utilizzo dell'interfaccia di pubblicazione / sottoscrizione	AMQSPUBA, AMQSSUBA, AMQSSBXA	nessun campione	nessun campione	AMQSPUBC, AMQSSUBC, AMQSSBXC
Client ricollegabili (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	nessun campione	nessun campione	nessun campione
Utilizzo dei consumer dei messaggi per l'utilizzo asincrono dei messaggi da più code (5)	AMQSCBFO	nessun campione	nessun campione	nessun campione
Specifica delle informazioni di connessione TLS su MQCONNX	SSLC AMQS	nessun campione	nessun campione	SSLC AMQS
Connessione al gestore code mediante MQCONNX	AMQSCNXC	nessun campione	nessun campione	AMQSCNXC
Interrogare le proprietà di un handle del messaggio, utilizzando MQINQMP, da una coda messaggi	MMQISQMA	nessun campione	nessun campione	AMQISQMC
Impostare le proprietà di un handle del messaggio utilizzando MQSETMP e inserirlo in una coda messaggi	AMQSSQMA	nessun campione	nessun campione	AMQSSQC

Note:

1. L'origine per gli esempi C si trova nel file QMQMSAMP/QCSRC. I file di inclusione esistono come membri nel file QMQM/H.
2. L'origine per gli esempi COBOL si trova nei file QMQMSAMP/QCBLLESRC. I membri sono denominati AMQ0 xxx 4, dove xxx indica la funzione di esempio.
3. L'origine per gli esempi RPG si trova in QMQMSAMP/QRPGLESRC. I membri sono denominati AMQ3 xxx 4, dove xxx indica la funzione di esempio. I membri di copia esistono in QMQM/QRPGLESRC. Ogni nome membro ha il suffisso G.

4. La versione eseguibile degli esempi IBM MQ MQI client condividono la stessa origine degli esempi eseguiti in un ambiente server. L'origine per gli esempi nell'ambiente client è la stessa del server. Gli esempi IBM MQ MQI client sono collegati alla libreria client LIBMQIC e gli esempi server IBM MQ sono collegati alla libreria server LIBMQM.
5. Se è necessario eseguire l'eseguibile client per l'applicazione di esempio del client ricollegabile e dell'applicazione consumer asincrona, è necessario compilarlo e collegarlo con la libreria con thread LIBMQIC_R. Quindi, deve essere eseguito in un ambiente con thread. Impostare la variabile di ambiente QIBM_MULTI_THREADED su 'Y' ed eseguire l'applicazione da qsh.

   Per ulteriori informazioni, consultare Impostazione di IBM MQ con Java e JMS .

Per ulteriori informazioni, consultare “Preparazione ed esecuzione di programmi di esempio su IBM i” a pagina 1074.

Oltre a questi, l'opzione di esempio IBM MQ for IBM i include un file di dati di esempio, che viene utilizzato come input per i programmi di esempio, AMQSDATA e i programmi CL di esempio che dimostrano le attività di amministrazione. Gli esempi CL sono descritti in Amministrazione IBM i . È possibile utilizzare il programma CL di esempio `amqsamp4` per creare code da utilizzare con i programmi di esempio descritti in questo argomento.

Multi

Preparazione ed esecuzione dei programmi di esempio

Dopo aver completato la preparazione iniziale, è possibile eseguire i programmi di esempio.

Informazioni su questa attività

Prima di eseguire i programmi di esempio, è necessario creare un gestore code e creare anche le code necessarie. Potrebbe anche essere necessario eseguire un'ulteriore preparazione, ad esempio, se si desidera eseguire gli esempi COBOL. Dopo aver completato la preparazione necessaria, è possibile eseguire i programmi di esempio.

Procedura

Per informazioni su come preparare ed eseguire i programmi di esempio, consultare i seguenti argomenti:

- “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072
- “Preparazione ed esecuzione di programmi di esempio su IBM i” a pagina 1074
- “Preparazione ed esecuzione di programmi di esempio su AIX and Linux” a pagina 1076
- “Preparazione ed esecuzione di programmi di esempio su Windows” a pagina 1077

Multi

Configurazione di un gestore code per accettare connessioni client su Multiplatforms

Prima di poter eseguire le applicazioni di esempio, è necessario creare un gestore code. È quindi possibile configurare il gestore code per accettare in modo sicuro le richieste di connessione in entrata dalle applicazioni in esecuzione in modalità client.

Prima di iniziare

Verificare che il gestore code esista già e che sia stato avviato. Determinare se i record di autenticazione di canale sono già abilitati emettendo il comando MQSC:

```
DISPLAY QMGR CHLAUTH
```

Importante: Questa attività prevede che i record di autenticazione di canale siano abilitati. Se si tratta di un gestore code utilizzato da altri utenti e applicazioni, la modifica di questa impostazione influirà su tutti gli altri utenti e applicazioni. Se il gestore code non utilizza i record di autenticazione di canale, il passo 4 può essere sostituito con un metodo di autenticazione alternativo (ad esempio, un'uscita di sicurezza) che imposta MCAUSER sull' *ID utente non privilegiato* che verrà ottenuto nel passo “1” a pagina 1073.

È necessario conoscere il nome del canale che l'applicazione prevede di utilizzare in modo che l'applicazione possa utilizzare il canale. È inoltre necessario conoscere quali oggetti, ad esempio code o argomenti, l'applicazione prevede di utilizzare in modo che l'applicazione possa utilizzarli.

Informazioni su questa attività

Questa attività crea un ID utente non privilegiato da utilizzare per un'applicazione client che si connette al gestore code. L'accesso viene concesso all'applicazione client solo per essere in grado di utilizzare il canale di cui ha bisogno e la coda di cui ha bisogno utilizzando questo ID utente.

Procedura

1. Ottenere un ID utente sul sistema su cui è in esecuzione il gestore code. Per questa attività questo ID utente non deve essere un utente di gestione privilegiato. Questo ID utente sarà l'autorizzazione con cui la connessione client verrà eseguita sul gestore code.
2. Avviare un programma listener con i seguenti comandi, dove:

   qmgr - name è il nome del tuo gestore code
   nnnn è il numero di porta scelto

   a)
   Per sistemi AIX, Linux, and Windows :

   ```
   runmqlsr -t tcp -m qmgr-name -p nnnn
   ```

   b)
   Per IBM i:

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. Se l'applicazione utilizza il SISTEMA SYSTEM.DEF.SVRCONN quindi questo canale è già definito. Se la tua applicazione utilizza un altro canale, crealo immettendo il seguente comando MQSC:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   dove *channel - name* è il nome del tuo canale.
4. Creare una regola di autenticazione del canale consentendo solo all'indirizzo IP del sistema client di utilizzare il canale emettendo il seguente comando MQSC:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   dove

   channel - name è il nome del tuo canale.
   client - machine - IP - address è l'indirizzo IP del sistema client. Se l'applicazione client di esempio è in esecuzione sulla stessa macchina del gestore code, utilizzare l'indirizzo IP '127.0.0.1' se l'applicazione si connetterà utilizzando 'localhost'. Se diverse macchine client si collegheranno, è possibile utilizzare un modello o un intervallo invece di un singolo indirizzo IP. Consultare Indirizzi IP generici per i dettagli.
   non - privileged - user - id è l'ID utente ottenuto nel passo “1” a pagina 1073
5. Se l'applicazione utilizza il SISTEMA SYSTEM.DEFAULT.LOCAL.QUEUE questa coda è già definita. Se l'applicazione utilizza un'altra coda, crearla emettendo il comando MQSC riportato di seguito:

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

dove *nome - coda* è il nome della coda.

6. Concedere l'accesso per connettersi e interrogare il gestore code immettendo il seguente comando MQSC:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

   dove *non - privileged - user - id* è l'ID utente ottenuto nel passo “1” a pagina 1073 .

7. Se l'applicazione è un'applicazione point - to - point, ovvero utilizza le code, concedere l'accesso per consentire l'interrogazione e l'inserimento e il richiamo dei messaggi utilizzando la coda dall'ID utente da utilizzare, emettendo i seguenti comandi MQSC:

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

   dove

   nome - coda è il nome della coda

   non - privileged - user - id è l'ID utente ottenuto nel passo “1” a pagina 1073

8. Se l'applicazione è un'applicazione di pubblicazione / sottoscrizione, ossia utilizza gli argomenti, concedere l'accesso per consentire la pubblicazione e la sottoscrizione utilizzando l'argomento dall'ID utente da utilizzare, immettendo i comandi MQSC:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

   dove

   non - privileged - user - id è l'ID utente ottenuto nel passo “1” a pagina 1073

   Ciò fornirà a *non - privileged - user - id* l'accesso a qualsiasi argomento nella struttura ad albero degli argomenti, in alternativa, è possibile definire un oggetto argomento utilizzando **DEFINE TOPIC** e concedere gli accessi solo alla parte della struttura ad albero degli argomenti a cui fa riferimento tale oggetto argomento. Per i dettagli, consultare Controllo dell'accesso utente agli argomenti .

Operazioni successive

L'applicazione client può ora connettersi al gestore code e inserire o richiamare i messaggi utilizzando la coda.

Concetti correlati

ALW Accesso a un oggetto IBM MQ su AIX, Linux, and Windows

Riferimenti correlati

SET CHLAUTH

Definire il canale

DEFINE QLOCAL

SET AUTHREC

IBM i Autorizzazioni IBM MQ su IBM i

IBM i *Preparazione ed esecuzione di programmi di esempio su IBM i*

Prima di poter eseguire i programmi di esempio su IBM i, è necessario creare un gestore code e anche le code necessarie. Se si desidera eseguire gli esempi COBOL, potrebbe essere necessario eseguire un'ulteriore preparazione.

Informazioni su questa attività

L'origine per i programmi di esempio IBM MQ for IBM i viene fornita nella libreria QMQMSAMP come membri di QCSRC, QCLSRC, QCBLLESRC e QRPGLESRC.

È possibile utilizzare le proprie code quando si eseguono gli esempi oppure è possibile eseguire il programma di esempio AMQSAMP4 per creare alcune code di esempio. L'origine per il programma AMQSAMP4 è incluso nel file QCLSRC nella libreria QMQMSAMP. È possibile compilarlo utilizzando il comando CRTCLPGM.

Per eseguire gli esempi, utilizzare le versioni eseguibili C fornite nella libreria QMQM oppure compilarli in modo simile a qualsiasi altra applicazione IBM MQ .

Procedura

1. Creare un gestore code e impostare le definizioni predefinite.

   È necessario eseguire questa operazione prima di poter eseguire uno qualsiasi dei programmi di esempio. Per ulteriori informazioni sulla creazione di un gestore code, consultare Amministrazione IBM MQ. Per informazioni sulla configurazione di un gestore code per accettare in modo sicuro richieste di connessioni in entrata dalle applicazioni in esecuzione in modalità client, consultare “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072.
2. Per richiamare uno dei programmi di esempio utilizzando i dati dal membro PUT nel file AMQSDATA della libreria QMQMSAMP, utilizzare un comando come:

   ```
   CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
   ```

   Nota: Per un modulo compilato per utilizzare il file system IFS, specificare l'opzione SYSIFCOPT (*IFSIO) su CRTCMOD, quindi il nome file, passato come parametro, deve essere specificato nel seguente formato:

   ```
   home/me/myfile
   ```
3. Se si desidera utilizzare le versioni COBOL degli esempi Inquire, Set ed Echo, modificare le definizioni di processo prima di eseguire questi esempi.

   Per gli esempi Inquire, Set ed Echo, le definizioni di esempio attivano le versioni C di questi esempi. Se si desidera utilizzare le versioni COBOL, è necessario modificare le definizioni del processo:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   Su IBM i, è possibile utilizzare il comando **CHGMQMPRC** (per i dettagli, vedere Modifica del processo MQ (CHGMQMPRC)) oppure modificare ed eseguire il comando **AMQSAMP4** con la definizione alternativa.
4. Eseguire i programmi di esempio.

   Per ulteriori informazioni sui parametri previsti da ciascun campione, consultare le descrizioni dei singoli campioni.

   Nota: Per i programmi di esempio COBOL, quando si passano i nomi delle code come parametri, è necessario fornire 48 caratteri, se necessario con spazi vuoti. Qualsiasi elemento diverso da 48 caratteri causa il malfunzionamento del programma con codice di errore 2085.

Riferimenti correlati

“Esempi per IBM i” a pagina 1069
Le tecniche dimostrati dai programmi di esempio per IBM MQ su sistemi IBM i .

Linux AIX *Preparazione ed esecuzione di programmi di esempio su AIX and Linux*

Prima di poter eseguire i programmi di esempio su AIX and Linux, è necessario creare un gestore code e anche le code necessarie. Se si desidera eseguire gli esempi COBOL, potrebbe essere necessario eseguire un'ulteriore preparazione.

Informazioni su questa attività

I file di esempio dei sistemi IBM MQ su AIX and Linux si trovano nelle directory elencate in Tabella 160 a pagina 1076 se i valori predefiniti sono stati utilizzati al momento dell'installazione.

Tabella 160. Dove trovare gli esempi per IBM MQ su sistemi AIX and Linux

Contenuto	**Directory**
file di origine	*MQ_INSTALLATION_PATH*/samp
file di origine del gestore code di messaggi non instradabili	*MQ_INSTALLATION_PATH*/samp/dlq
file eseguibili	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Gli esempi necessitano di una serie di code da gestire. È possibile utilizzare le proprie code o eseguire il file MQSC di esempio amqscos0.tst per creare una serie. Per eseguire gli esempi, utilizzare le versioni eseguibili fornite o compilare le versioni di origine come qualsiasi altra applicazione, utilizzando un compilatore ANSI.

Procedura

1. Creare un gestore code e impostare le definizioni predefinite.

   È necessario eseguire questa operazione prima di poter eseguire uno qualsiasi dei programmi di esempio. Per ulteriori informazioni sulla creazione di un gestore code, consultare Amministrazione IBM MQ. Per informazioni sulla configurazione di un gestore code per accettare in modo sicuro richieste di connessioni in entrata dalle applicazioni in esecuzione in modalità client, consultare “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072.
2. Se non si utilizzano le proprie code, eseguire il file MQSC di esempio amqscos0.tst per creare una serie di code.

   Per eseguire questa operazione sui sistemi AIX and Linux , immettere:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   Controllare il file `sampobj.out` per assicurarsi che non vi siano errori.
3. Se si desidera utilizzare le versioni COBOL degli esempi Inquire, Set ed Echo, modificare le definizioni di processo prima di eseguire questi esempi.

   Per gli esempi Inquire, Set ed Echo, le definizioni di esempio attivano le versioni C di questi esempi. Se si desidera utilizzare le versioni COBOL, è necessario modificare le definizioni del processo:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   Su Windows, eseguire questa operazione modificando il file `amqscos0.tst` e modificando i nomi file eseguibili C nei nomi file eseguibili COBOL prima di utilizzare il comando **runmqsc** per eseguire questi esempi.
4. Eseguire i programmi di esempio.

   Per eseguire un esempio, immetterne il nome seguito da qualsiasi parametro, ad esempio:

```
amqsput myqueue qmanagername
```

dove *myqueue* è il nome della coda in cui verranno inseriti i messaggi e *qmanagername* è il gestore code proprietario di *myqueue*.

Per ulteriori informazioni sui parametri previsti da ciascun campione, consultare le descrizioni dei singoli campioni.

Nota: Per i programmi di esempio COBOL, quando si passano i nomi delle code come parametri, è necessario fornire 48 caratteri, se necessario con spazi vuoti. Qualsiasi elemento diverso da 48 caratteri causa il malfunzionamento del programma con codice di errore 2085.

Riferimenti correlati

“Esempi per sistemi AIX and Linux” a pagina 1063
Le tecniche dimostrate dai programmi di esempio per IBM MQ for AIX or Linux.

Windows *Preparazione ed esecuzione di programmi di esempio su Windows*

Prima di poter eseguire i programmi di esempio su Windows, è necessario creare un gestore code e anche le code necessarie. Se si desidera eseguire gli esempi COBOL, potrebbe essere necessario eseguire un'ulteriore preparazione.

Informazioni su questa attività

I file di esempio IBM MQ for Windows si trovano nelle directory elencate in Tabella 161 a pagina 1077, se i valori predefiniti sono stati utilizzati al momento dell'installazione. L'unità di installazione assume il valore predefinito < c:>.

Tabella 161. Dove trovare gli esempi per IBM MQ for Windows

Contenuto	**Directory**
Codice sorgente C	*MQ_INSTALLATION_PATH*\Tools\C\Esempi
Codice di origine per l'esempio di gestore di lettere non recapitate	*MQ_INSTALLATION_PATH*\Tools\C\Esempi \DLQ
Codice sorgente COBOL	*MQ_INSTALLATION_PATH*\Tools\Cobol \ Esempi
File eseguibili C [1]	*MQ_INSTALLATION_PATH*\ Tools\C\Samples \ Bin (versioni a 32 bit) *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (versioni a 64 bit)
File MQSC di esempio	*MQ_INSTALLATION_PATH*\Tools\MQSC\Esempi
Codice sorgente Visual Basic	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET esempi	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Esempi

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Nota: Sono disponibili versioni a 64 - bit di alcuni esempi di file eseguibili C.

Gli esempi necessitano di una serie di code da gestire. È possibile utilizzare le proprie code o eseguire il file MQSC di esempio amqscos0.tst per creare una serie di code. Per eseguire gli esempi, utilizzare le versioni eseguibili fornite o compilare le versioni di origine come si farebbe con qualsiasi altra applicazione IBM MQ for Windows .

Procedura

1. Creare un gestore code e impostare le definizioni predefinite.

   È necessario eseguire questa operazione prima di poter eseguire uno qualsiasi dei programmi di esempio. Per ulteriori informazioni sulla creazione di un gestore code, consultare Amministrazione IBM

MQ. Per informazioni sulla configurazione di un gestore code per accettare in modo sicuro richieste di connessioni in entrata dalle applicazioni in esecuzione in modalità client, consultare “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072.

2. Se non si utilizzano le proprie code, eseguire il file MQSC di esempio amqscos0.tst per creare una serie di code.

   Per eseguire questa operazione sui sistemi Windows , immettere:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   Controllare il file `sampobj.out` per assicurarsi che non vi siano errori. Questo file è nella directory corrente.

3. Se si desidera utilizzare le versioni COBOL degli esempi Inquire, Set ed Echo, modificare le definizioni di processo prima di eseguire questi esempi.

   Per gli esempi Inquire, Set ed Echo, le definizioni di esempio attivano le versioni C di questi esempi. Se si desidera utilizzare le versioni COBOL, è necessario modificare le definizioni del processo:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   Su Windows, eseguire questa operazione modificando il file `amqscos0.tst` e modificando i nomi file eseguibili C nei nomi file eseguibili COBOL prima di utilizzare il comando **runmqsc** per eseguire questi esempi.

4. Eseguire i programmi di esempio.

   Per eseguire un esempio, immetterne il nome seguito da qualsiasi parametro, ad esempio:

   ```
   amqsput myqueue qmanagername
   ```

   dove *myqueue* è il nome della coda in cui verranno inseriti i messaggi e *qmanagername* è il gestore code proprietario di *myqueue*.

   Per ulteriori informazioni sui parametri previsti da ciascun campione, consultare le descrizioni dei singoli campioni.

   Nota: Per i programmi di esempio COBOL, quando si passano i nomi delle code come parametri, è necessario fornire 48 caratteri, se necessario con spazi vuoti. Qualsiasi elemento diverso da 48 caratteri causa il malfunzionamento del programma con codice di errore 2085.

Riferimenti correlati

“Esempi per IBM MQ for Windows” a pagina 1066
Le tecniche dimostrate dai programmi di esempio per IBM MQ for Windows.

“Esempi Visual Basic per IBM MQ for Windows” a pagina 1069
Le tecniche dimostrati dai programmi di esempio per IBM MQ su sistemi Windows .

Il programma di esempio di uscita API

L'uscita API di esempio genera una traccia MQI per un file specificato dall'utente con un prefisso definito nella variabile d'ambiente **MQAPI_TRACE_LOGFILE** .

Per ulteriori informazioni sulle uscite API, consultare “Scrittura e compilazione di uscite API su Multiplatforms” a pagina 955.

Origine
amqsaxe0.c

Binario
amqsaxe

Configurazione per l'uscita di esempio

1. Aggiungere le seguenti informazioni alla stanza ApiExitLocal del file `qm.ini` .

Piattaforme diverse da Windows

```
ApiExitLocal:
Sequence=100
Function=EntryPoint
Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
Name=SampleApiExit
```

dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ .

Windows

```
ApiExitLocal:
Sequence=100
Function=EntryPoint
Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
Name=SampleApiExit
```

dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ .

2. Imposta la variabile d'ambiente **MQAPI_TRACE_LOGFILE**

```
MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
```

3. Esegui la tua applicazione.

   I file di output vengono creati nella directory di `/tmp` con nomi come: `MqiTrace.`*`pid`*`.`*`tid`*`.log`.

Il programma di esempio di consumo asincrono

Il programma di esempio amqscbf dimostra l'utilizzo di MQCB e MQCTL per utilizzare i messaggi da più code in modo asincrono.

amqscbf è fornito come codice sorgente C e un client binario e un server eseguibile su piattaforme AIX, Linux, and Windows .

Il programma viene avviato dalla riga comandi e utilizza i seguenti parametri facoltativi:

```
Usage: [Options] Queue Name {queue_name}
  where Options are:
    -m Queue Manager Name
    -o Open options
    -r Reconnect Type
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

Fornire più di un nome coda per leggere i messaggi da più code (l'esempio supporta un massimo di dieci code).

Nota: Reconnect type è valida solo per programmi client.

Esempio

L'esempio mostra amqscbf eseguito come un programma server che legge un messaggio da QL1 e viene quindi arrestato.

Utilizzare IBM MQ Explorer per inserire un messaggio di verifica su QL1. Arrestare il programma premendo Invio.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1
```

```
Sample AMQSCBF0 end
```

Cosa dimostra amqscbf

L'esempio mostra come leggere i messaggi da più code in ordine di arrivo. Ciò richiederebbe molto più codice utilizzando MQGET sincrono. Nel caso di un utilizzo asincrono, non è richiesto alcun polling e la gestione del thread e della memoria viene eseguita da IBM MQ. Un esempio "reale" dovrebbe occuparsi degli errori; nell'esempio gli errori vengono scritti nella console.

Il codice di esempio contiene la seguente procedura:

1. Definire la singola funzione di callback di consumo del messaggio,

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
                     MQCBC   * pContext)
{ ... }
```

2. Connettersi al gestore code,

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. Aprire le code di immissione e associarle alla funzione di callback `MessageConsumer` ,

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` non deve essere impostato per ciascuna coda; è un campo di sola immissione. Ma è possibile associare una funzione di callback differente a ciascuna coda.

4. Avviare l'utilizzo dei messaggi,

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Attendere che l'utente abbia premuto Invio e quindi interrompere l'utilizzo dei messaggi,

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Infine, disconnettersi dal gestore code,

```
MQDISC(&Hcon,&CompCode,&Reason);
```

Il programma di esempio Put asincrono

Informazioni sull'esecuzione dell'esempio amqsapt e sulla progettazione del programma di esempio Asynchronous Put.

Il programma di esempio di inserimento asincrono inserisce i messaggi su una coda utilizzando la chiamata MQPUT asincrona e quindi richiama le informazioni sullo stato utilizzando la chiamata MQSTAT. Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per il nome di questo programma su piattaforme differenti.

Esecuzione dell'esempio amqsapt

Questo programma impiega fino a 6 parametri:

1. Il nome della coda di destinazione (obbligatorio)

2. Il nome del gestore code (facoltativo)
3. Opzioni di apertura (facoltativo)
4. Opzioni di chiusura (facoltativo)
5. Il nome del gestore code di destinazione (facoltativo)
6. Il nome della coda dinamica (facoltativo)

Se non viene specificato un gestore code, amqsapt si connette al gestore code predefinito.

Progettazione del programma di esempio Put asincrono

Il programma utilizza la chiamata MQOPEN con le opzioni di output fornite o con le opzioni MQOO_OUTPUT e MQOO_FAIL_IF_QUIESCING per aprire la coda di destinazione per l'inserimento dei messaggi.

Se non riesce ad aprire la coda, il programma emette un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN. Per semplificare il programma, in questa e nelle successive chiamate MQI, il programma utilizza i valori predefiniti per molte opzioni.

Per ogni riga di input, il programma legge il testo in un buffer e utilizza la chiamata MQPUT con MQPMO_ASYNC_RESPONSE per creare un messaggio datagramma contenente il testo di tale riga e inserirlo in maniera asincrona nella coda di destinazione. Il programma continua fino a quando non raggiunge la fine dell'input o la chiamata MQPUT ha esito negativo. Se il programma raggiunge la fine dell'input, chiude la coda utilizzando la chiamata MQCLOSE.

Il programma, quindi, emette la chiamata MQSTAT, restituendo una struttura MQSTS e visualizza i messaggi contenenti il numero di messaggi immessi correttamente, il numero di messaggi immessi con un'avvertenza e il numero di errori.

Programmi di esempio Sfoglia

I programmi di esempio Sfoglia consultano i messaggi su una coda utilizzando la chiamata MQGET.

Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

Progettazione del programma di esempio Sfoglia

Il programma apre la coda di destinazione utilizzando la chiamata MQOPEN con l'opzione MQOO_BROWSE. Se non riesce ad aprire la coda, il programma emette un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN.

Per ogni messaggio sulla coda, il programma utilizza la chiamata MQGET per copiare il messaggio dalla coda, quindi visualizza i dati contenuti nel messaggio. La chiamata MQGET utilizza queste opzioni:

MQGMO_BROWSE_SUCCESSIVO
Dopo la chiamata MQOPEN, il cursore di esplorazione viene posizionato logicamente prima del primo messaggio nella coda, quindi questa opzione determina la restituzione del ***primo*** messaggio quando la chiamata viene effettuata per la prima volta.

MQGMO_NO_WAIT
Il programma non attende se non ci sono messaggi nella coda.

MQGMO_ACCEPT_TRUNCATED_MSG
La chiamata MQGET specifica un buffer di dimensione fissa. Se un messaggio è più lungo di questo buffer, il programma visualizza il messaggio troncato, insieme ad un'avvertenza che il messaggio è stato troncato.

Il programma dimostra come è necessario cancellare i dati dei campi *MsgId* e *CorrelId* della struttura MQMD dopo ogni chiamata MQGET, poiché la chiamata imposta questi campi sui valori contenuti nel messaggio richiamato. La cancellazione di questi campi significa che le chiamate MQGET successive richiamano i messaggi nell'ordine in cui sono conservati nella coda.

Il programma continua fino alla fine della coda; la chiamata MQGET restituisce il codice motivo MQRC_NO_MSG_AVAILABLE e il programma visualizza un messaggio di avviso. Se la chiamata MQGET non riesce, il programma visualizza un messaggio di errore che contiene il codice di errore.

Il programma chiude quindi la coda utilizzando la chiamata MQCLOSE.

Programmi di esempio Sfoglia per AIX, Linux, and Windows
Considerare l'utilizzo di questo argomento per informazioni su Sfoglia programmi di esempio su AIX, Linux, and Windows.

La versione C del programma richiede 2 parametri

1. Il nome della coda di origine (necessario)
2. Il nome del gestore code (facoltativo)

Se un gestore code non è specificato, si connette a quello predefinito. Ad esempio, immettere uno dei seguenti:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

dove `myqueue` è il nome della coda da cui verranno visualizzati i messaggi e `qmanagername` è il gestore code proprietario di `myqueue`.

Se si omette `qmanagername`, quando si esegue l'esempio C, si presuppone che il gestore code predefinito sia proprietario della coda.

La versione COBOL non ha alcun parametro. Si connette al gestore code predefinito e quando viene eseguito viene richiesto:

```
Please enter the name of the target queue
```

Vengono visualizzati solo i primi 50 caratteri di ciascun messaggio, seguiti da `- - - truncated` in questo caso.

I programmi di esempio Sfoglia su IBM i
Ogni programma richiama le copie di tutti i messaggi sulla coda specificata quando si richiama il programma; i messaggi rimangono sulla coda.

È possibile utilizzare la coda fornita SYSTEM.SAMPLE.LOCAL; eseguire prima il programma di esempio Put per inserire alcuni messaggi nella coda. È possibile utilizzare la coda SYSTEM.SAMPLE.ALIAS, che è un nome alias per la stessa coda locale. Il programma continua fino a quando non raggiunge la fine della coda o una chiamata MQI non riesce.

Gli esempi C consentono di specificare il nome del gestore code, generalmente come secondo parametro, in modo simile agli esempi dei sistemi Windows . Ad esempio:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Se un gestore code non è specificato, si connette a quello predefinito. Ciò è rilevante anche per gli esempi RPG. Tuttavia, con gli esempi RPG è necessario fornire un nome gestore code invece di consentirgli di utilizzare il valore predefinito.

ALW *Il programma di esempio Browser*

Il programma di esempio Browser legge e scrive sia il descrittore del messaggio che i campi del contenuto del messaggio di tutti i messaggi su una coda.

Il programma di esempio è scritto come un programma di utilità, non solo per dimostrare una tecnica. Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

Questo programma utilizza i seguenti parametri posizionali:

1. Il nome della coda di origine (obbligatorio)
2. Il nome del gestore code (obbligatorio)
3. Un parametro facoltativo per le proprietà (facoltativo)

Utilizzare le seguenti variabili di ambiente per fornire credenziali utilizzate per l'autenticazione con il gestore code:

ID UTENTE MQSAMP
Impostare l'ID utente da utilizzare per l'autenticazione della connessione, se si desidera utilizzare un ID utente e una password per l'autenticazione con il gestore code. Il programma richiede la password per accompagnare l'ID utente.

V 9.3.4 Linux AIX **MQSAMP_TOKEN**
Impostare su un valore non vuoto se si desidera fornire un token di autenticazione per l'autenticazione con il gestore code. Il programma richiede il token di autenticazione. I token di autenticazione possono essere utilizzati solo dall'esempio **amqsbcgc** che utilizza i bind client.

Per eseguire questi programmi, immettere uno dei seguenti comandi:

- `amqsbcg` *`myqueue qmanagername`*
- `amqsbcgc` *`myqueue qmanagername`*

dove *`myqueue`* è il nome della coda in cui verranno esaminati i messaggi e *`qmanagername`* è il gestore code proprietario di *`myqueue`*.

Legge ogni messaggio dalla coda e scrive quanto segue in stdout:

- Campi del descrittore del messaggio formattato
- Dati del messaggio (dump in formato esadecimale e, dove possibile, in formato carattere)

Tabella 162. Valori consentiti per il parametro della proprietà

Valore	Comportamento
0	Comportamento predefinito. Le proprietà che vengono consegnate all'applicazione dipendono dall'attributo coda **`PropertyControl`** da cui viene richiamato il messaggio.

1	Un handle del messaggio viene creato e utilizzato con MQGET. Le proprietà del messaggio, eccetto quelle contenute nel descrittore del messaggio (o estensione), vengono visualizzate in modo simile al descrittore del messaggio. Ad esempio:
	`****Message properties****`
	`  property name: property value`
	Oppure, se non sono disponibili proprietà:
	`****Message properties****`
	`  None`
	I valori numerici vengono visualizzati utilizzando `printf`, i valori stringa vengono racchiusi tra virgolette singole e le stringhe di byte vengono racchiuse tra virgolette singole e X, come per il descrittore del messaggio.

2	Viene specificato MQGMO_NO_PROPERTIES, in modo che vengano restituite solo le proprietà del descrizione del messaggio.
3	Viene specificato MQGMO_PROPERTIES_FORCE_MQRFH2 , in modo che tutte le proprietà vengano restituite nei dati del messaggio.

Tabella 162. Valori consentiti per il parametro della proprietà (Continua)	
Valore	**Comportamento**
4	MQGMO_PROPERTIES_COMPATIBILITY è specificato, in modo che tutte le proprietà possano essere restituite a seconda che sia inclusa una proprietà IBM MQ , altrimenti le proprietà vengono eliminate.

Il programma è limitato alla stampa dei primi 65535 caratteri del messaggio e ha esito negativo con il motivo `truncated msg` se viene letto un messaggio più lungo.

Per un esempio dell'output di questo programma di utilità, consultare Esplorazione delle code.

L'esempio di transazione CICS

Viene fornito un programma di transazioni CICS di esempio, denominato amqscic0.ccs per il codice sorgente e amqscic0 per la versione eseguibile. È possibile creare transazioni utilizzando le funzionalità CICS standard.

Consultare “Creazione di un'applicazione procedurale” a pagina 1002 per i dettagli sui comandi necessari per la propria piattaforma.

La transazione legge i messaggi dalla coda di trasmissione SYSTEM.SAMPLECICS.WORKQUEUE sul gestore code predefinito e le colloca nella coda locale, il nome del quale è contenuto nell'intestazione di trasmissione del messaggio. Eventuali errori vengono inviati alla coda SYSTEM.SAMPLE.CICS.DLQ.

Nota: È possibile utilizzare uno script MQSC di esempio amqscic0.tst per creare queste code e code di input di esempio.

Il programma di esempio Connect

Il programma di esempio Connect consente di esplorare la chiamata MQCONNX e le sue opzioni da un client. L'esempio si connette al gestore code utilizzando la chiamata MQCONNX, richiede il nome del gestore code utilizzando la chiamata MQINQ e lo visualizza. Inoltre, vengono fornite informazioni sull'esecuzione dell'esempio amqscnxc.

Nota: Il programma di esempio Connect è un esempio client. È possibile compilarlo ed eseguirlo su un server, ma la funzione è significativa solo su un client e vengono forniti solo i file eseguibili dal client.

Esecuzione dell'esempio amqscnxc

La sintassi della riga comandi del programma di esempio Connect è:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

I parametri sono facoltativi e il loro ordine non è importante ad eccezione di QMgrName, che, se specificato, deve essere l'ultimo. I parametri sono:

ConnName
Il nome della connessione TCP/IP del gestore code del server

Se non si specifica il nome connessione TCP/IP, MQCONNX viene emesso con *ClientConnPtr* impostato su NULL.

Nome SvrconnChannel
Il nome del canale di connessione server

Se si specifica il nome connessione TCP/IP ma non il canale di connessione server (l'inverso non è consentito), l'esempio utilizza il nome SYSTEM.DEF.SVRCONN.

Utente
Il nome utente da utilizzare per l'autenticazione della connessione

Se si specifica questa opzione, il programma richiederà una parola d'ordine per accompagnare tale ID utente.

QMgrName
Il nome del gestore code di destinazione

Se non si specifica il gestore code di destinazione, l'esempio si connette al gestore code in ascolto sul nome della connessione TCP/IP fornito.

Nota: Se si immette un punto interrogativo come unico parametro o se si immettono parametri non corretti, si riceve un messaggio che spiega come utilizzare il programma.

Se si esegue l'esempio senza alcuna opzione della riga comandi, il contenuto della variabile di ambiente MQSERVER viene utilizzato per determinare le informazioni di connessione. (In questo esempio MQSERVER è impostata su `SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com`.) Viene visualizzato il seguente output:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

Se si esegue l'esempio e si fornisce un nome di connessione TCP/IP e un nome di canale di connessione server, ma nessun nome di gestore code di destinazione, come riportato di seguito:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

viene utilizzato il nome gestore code predefinito e viene visualizzato un output simile al seguente:

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Se si esegue l'esempio e si fornisce un nome connessione TCP/IP e un nome gestore code di destinazione, come segue:

```
amqscnxc -x machine.site.company.com MACHINE
```

viene visualizzato un output simile al seguente:

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Il programma di esempio Conversione dati

Il programma di esempio di conversione dati è una struttura di una routine di uscita di conversione dati. Scopri la progettazione del campione di conversione dati.

Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

Progettazione del campione di conversione dati

Ogni routine di uscita di conversione dati converte un singolo formato di messaggio denominato. Questa struttura è intesa come un wrapper per i frammenti di codice generati dal programma di utilità di generazione dell'uscita di conversione dati.

Il programma di utilità produce un frammento di codice per ogni struttura di dati; molte di queste strutture costituiscono un formato, quindi diversi frammenti di codice vengono aggiunti a questa struttura per produrre una routine per eseguire la conversione dei dati dell'intero formato.

Il programma verifica quindi se la conversione è riuscita o meno e restituisce i valori richiesti al chiamante.

Esempi di coordinamento database

Vengono forniti due esempi che dimostrano come IBM MQ può coordinare sia gli aggiornamenti IBM MQ che gli aggiornamenti del database nella stessa unità di lavoro.

Questi esempi sono:

1. AMQSXAS0 (in C) o AMQ0XAS0 (in COBOL), che aggiorna un singolo database all'interno di un'unità di lavoro IBM MQ .
2. AMQSXAG0 (in C) o AMQ0XAG0 (in COBOL), AMQSXAB0 (in C) o AMQ0XAB0 (in COBOL), e AMQSXAF0 (in C) o AMQ0XAF0 (in COBOL), che insieme aggiornano due database all'interno di un'unità di lavoro IBM MQ , mostrando come è possibile accedere a più database. Questi esempi vengono forniti per mostrare l'utilizzo della chiamata MQBEGIN, le chiamate SQL e IBM MQ miste e dove e quando connettersi a un database.

Figura 128 a pagina 1087 mostra il modo in cui gli esempi forniti vengono utilizzati per aggiornare i database:

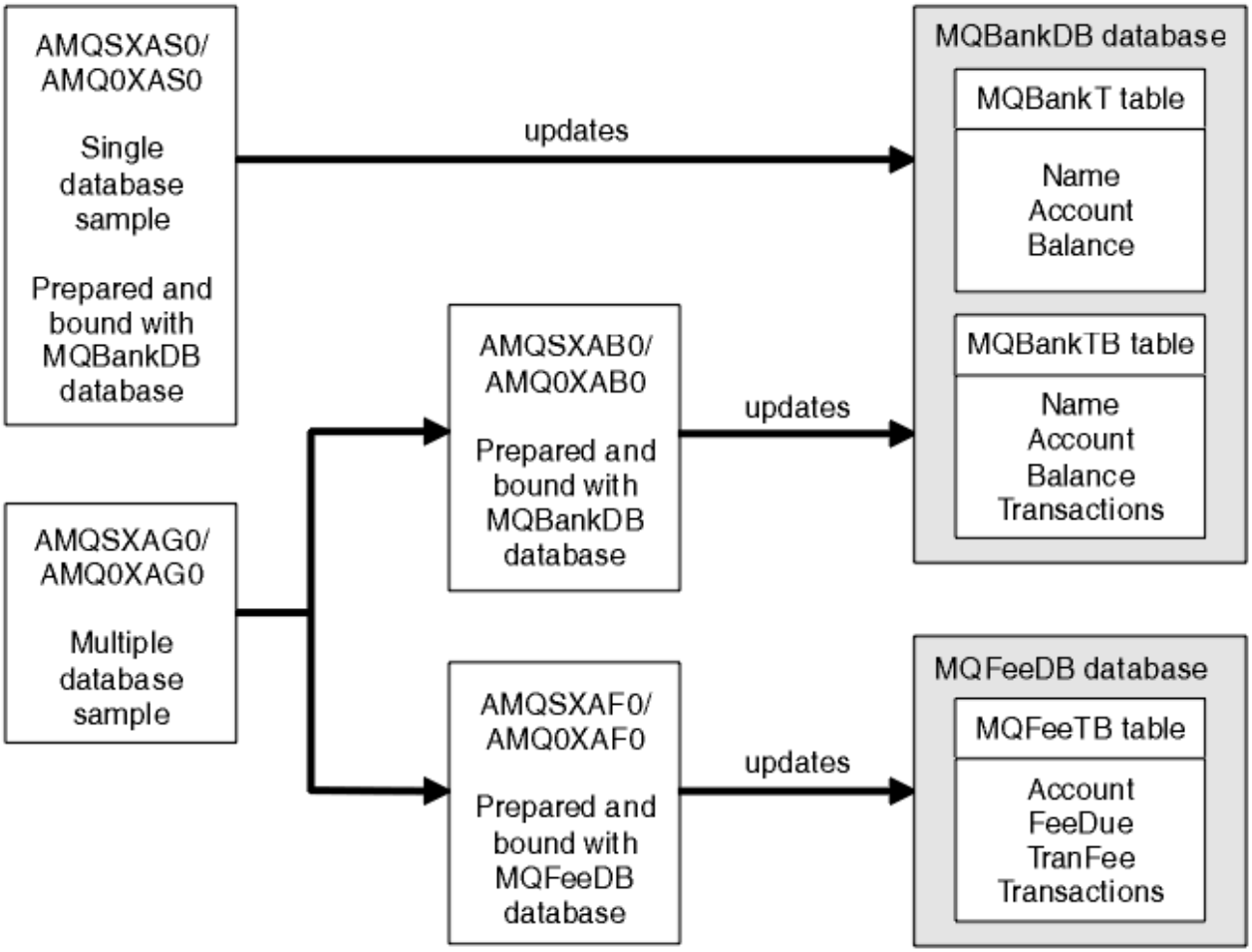


Figura 128. Gli esempi di coordinamento del database

I programmi leggono un messaggio da una coda (nel punto di sincronizzazione), quindi, utilizzando le informazioni nel messaggio, ottengono le informazioni rilevanti dal database e le aggiornano. Il nuovo stato del database viene quindi stampato.

La logica del programma è la seguente:

1. Utilizzare il nome della coda di input dall'argomento del programma
2. Connettersi al gestore code predefinito (o, facoltativamente, al nome fornito in C) utilizzando MQCONN
3. Aprire una coda (utilizzando MQOPEN) per l'input mentre non vi sono errori
4. Avviare un'unità di lavoro utilizzando MQBEGIN
5. Richiama il successivo messaggio (utilizzando MQGET) dalla coda nel punto di sincronizzazione
6. Ottieni informazioni dai database
7. Aggiorna informazioni dai database
8. Esegui il commit delle modifiche utilizzando MQCMIT
9. Stampare le informazioni aggiornate (nessun messaggio disponibile conta come errore e il loop termina)
10. Chiudere la coda utilizzando MQCLOSE
11. Disconnetti dalla coda utilizzando MQDISC

I cursori SQL vengono utilizzati negli esempi, in modo che le letture dai database (ossia, più istanze) vengano bloccate mentre un messaggio viene elaborato, consentendo l'esecuzione simultanea di più

istanze di questi programmi. I cursori sono esplicitamente aperti, ma implicitamente chiusi dalla chiamata MQCMIT.

Il singolo esempio di database (AMQSXAS0 o AMQ0XAS0) non dispone di istruzioni SQL CONNECT e la connessione al database viene effettuata implicitamente da IBM MQ con la chiamata MQBEGIN. L'esempio di database multiplo (AMQSXAG0 o AMQ0XAG0, AMQSXAB0 o AMQ0XAB0e AMQSXAF0 o AMQ0XAF0) ha istruzioni SQL CONNECT, poiché alcuni prodotti database consentono una sola connessione attiva. Se questo non è il caso del prodotto database o se si sta accedendo a un singolo database in più prodotti database, è possibile rimuovere le istruzioni SQL CONNECT.

Gli esempi vengono preparati con il prodotto database IBM Db2 , quindi potrebbe essere necessario modificarli per lavorare con altri prodotti database.

Il controllo errori SQL utilizza le routine in UTIL.C e CHECKERR.CBL fornito da Db2. Questi devono essere compilati o sostituiti prima della compilazione e del collegamento.

Nota: Se si utilizza l'origine Micro Focus COBOL CHECKERR.MFC per il controllo degli errori SQL, è necessario modificare l'ID programma in maiuscolo, ossia CHECKERR, affinché AMQ0XAS0 sia collegato correttamente.

Creazione di database e tabelle
Creare i database e le tabelle prima di compilare gli esempi.

Per creare i database, utilizzare il metodo usuale per il prodotto database, ad esempio:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Creare le tabelle utilizzando le seguenti istruzioni SQL:

In C:

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

In COBOL:

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
```

```
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

Immettere i dati nelle tabelle utilizzando le istruzioni SQL nel modo seguente:

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
⋮
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
⋮
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
⋮
```

Nota: Per COBOL, utilizzare le stesse istruzioni SQL ma aggiungere END_EXEC alla fine di ogni riga.

Precompilazione, compilazione e collegamento degli esempi
Informazioni sulla precompilazione, la compilazione e il collegamento di esempi in C e COBOL.

Precompilare i file `.SQC` (in C) e `.SQB` (in COBOL) e collegarli al database appropriato per produrre i file `.C` o `.CBL` . A tale scopo, utilizzare il metodo tipico per il prodotto database.

Precompilazione in C

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

Precompilazione in COBOL

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

Compilazione e collegamento

I seguenti comandi di esempio utilizzano i simboli *DB2TOP* e *MQ_INSTALLATION_PATH*. *DB2TOP* rappresenta la directory di installazione per il prodotto Db2 . *MQ_INSTALLATION_PATH* rappresenta la directory di livello superiore in cui è installato IBM MQ .

- AIX Su AIX, il percorso della directory è:

```
/usr/lpp/db2_05_00
```

- Windows Sui sistemi Windows , il percorso della directory dipende dal percorso scelto durante l'installazione del prodotto. Se sono state scelte le impostazioni predefinite, il percorso è il seguente:

```
c:\sqllib
```

Nota: Prima di immettere il comando di collegamento su sistemi Windows , assicurarsi che la variabile di ambiente LIB contenga i percorsi delle librerie Db2 e IBM MQ .

Copiare i seguenti file in una directory temporanea:

- Il file `amqsxag0.c` dalla propria installazione IBM MQ

  Nota: Questo file può essere trovato nelle seguenti directory:

  – Linux AIX Su sistemi AIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows Su sistemi Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- I file `.c` ottenuti precompilando i file di origine `.sqc` , `amqsxas0.sqc`, `amqsxaf0.sqce` `amqsxab0.sqc`.
- I file `util.c` e `util.h` dall'installazione Db2 .

  Nota: Questi file si trovano nella directory:

  ```
  DB2TOP/samples/c
  ```

Creare i file di oggetto per ogni file `.c` utilizzando il seguente comando del compilatore per la piattaforma che si sta utilizzando:

- AIX AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Windows Sistemi Windows

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

Creare il file eseguibile `amqsxag0` utilizzando il seguente comando di collegamento per la piattaforma che si sta utilizzando:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows Sistemi Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

Creare il file eseguibile di amqsxas0 utilizzando i seguenti comandi di compilazione e collegamento per la piattaforma che si sta utilizzando:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Sistemi Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Ulteriori informazioni

AIX Se si sta lavorando su AIX e si desidera accedere a Oracle, utilizzare il compilatore xlc_r e il link a libmqm_r.a.

Esecuzione degli esempi
Utilizzare queste informazioni per informazioni su come configurare il gestore code prima di eseguire gli esempi di coordinamento del database su C e COBOL.

Prima di eseguire gli esempi, configurare il gestore code con il prodotto database che si sta utilizzando. Per informazioni su come eseguire questa operazione, consultare Scenario 1: il gestore code esegue il coordinamento.

I seguenti titoli forniscono informazioni su come eseguire gli esempi in C e COBOL:

- “Esempi C” a pagina 1091
- “Esempi COBOL” a pagina 1092

Esempi C

I messaggi devono essere nel seguente formato per essere letti da una coda:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT può essere utilizzato per inserire i messaggi nella coda.

Gli esempi di coordinamento del database richiedono due parametri:

1. Nome coda (obbligatorio)
2. Nome gestore code (facoltativo)

Supponendo che sia stato creato e configurato un gestore code per il singolo database di esempio denominato singDBQM, con una coda denominata singDBQ, si incrementa l'account di Mr Fred Bloggs di 50 come segue:

```
AMQSPUT singDBQ singDBQM
```

Quindi, immettere il seguente messaggio:

```
UPDATE Balance change=50 WHERE Account=1
```

È possibile inserire più messaggi nella coda.

```
AMQSXAS0 singDBQ singDBQM
```

Viene quindi stampato lo stato aggiornato del conto di Fred Bloggs.

Supponendo di aver creato e configurato un gestore code per l'esempio a più database denominato multDBQM, con una coda denominata multDBQ, si decrementa l'account di Mary Brown di 75 come segue:

```
AMQSPUT multDBQ multDBQM
```

Quindi, immettere il seguente messaggio:

```
UPDATE Balance change=-75 WHERE Account=3
```

È possibile inserire più messaggi nella coda.

```
AMQSXAG0 multDBQ multDBQM
```

Viene quindi stampato lo stato aggiornato del conto di Mary Brown.

Esempi COBOL

I messaggi devono essere nel seguente formato per essere letti da una coda:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

Per semplicità, il `Balance change` deve essere un numero di otto caratteri con segno e il `Account` deve essere un numero di otto caratteri.

L'esempio AMQSPUT può essere utilizzato per inserire i messaggi nella coda.

Gli esempi non utilizzano parametri e utilizzano il gestore code predefinito. Può essere configurato per eseguire solo uno degli esempi alla volta. Supponendo che sia stato configurato il gestore code predefinito per il singolo esempio di database, con una coda denominata singDBQ, si incrementa l'account di Fred Bloggs di 50 nel modo seguente:

```
AMQSPUT singDBQ
```

Quindi, immettere il seguente messaggio:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

È possibile inserire più messaggi nella coda:

```
AMQ0XAS0
```

Immettere il nome della coda:

```
singDBQ
```

Viene quindi stampato lo stato aggiornato del conto di Fred Bloggs.

Supponendo che sia stato configurato il gestore code predefinito per l'esempio di più database, con una coda denominata multDBQ, si decrementa l'account di Mary Brown di 75 come segue:

```
AMQSPUT multDBQ
```

Quindi, immettere il seguente messaggio:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

È possibile inserire più messaggi nella coda:

```
AMQ0XAG0
```

Immettere il nome della coda:

```
multDBQ
```

Viene quindi stampato lo stato aggiornato del conto di Mary Brown.

Esempio di gestore code di messaggi non instradabili

Viene fornito un gestore code di messaggi non instradabili di esempio, il nome della versione eseguibile è `amqsdlq`. Se si desidera un gestore code di messaggi non recapitabili diverso da **RUNMQDLQ**, l'origine dell'esempio è disponibile per l'utilizzo come base.

L'esempio è simile al gestore di lettere non recapitate fornito all'interno del prodotto, ma la traccia e la notifica degli errori sono differenti. Sono disponibili due variabili di ambiente:

TRACCIA_ODQ
: Impostare su YES o `yes` per attivare la traccia.

ODQ_MSG
: Impostare sul nome del file contenente i messaggi di errore e informativi. Il file fornito è denominato `amqsdlq.msg`.

È necessario rendere note queste variabili all'ambiente utilizzando i comandi **`export`** o **`set`** , a seconda della propria piattaforma; la funzione di traccia è disattivata utilizzando il comando **`unset`** .

È possibile modificare il file dei messaggi di errore, `amqsdlq.msg`, per adattarlo ai propri requisiti. L'esempio inserisce i messaggi in stdout, **non** nel file di log degli errori IBM MQ .

Per ulteriori informazioni sul funzionamento del gestore dei messaggi non instradabili e su come eseguirlo, consultare Elaborazione dei messaggi su una coda di messaggi non instradabili IBM MQ o *Guida alla gestione del sistema* per la propria piattaforma.

Programma di esempio Elenco di distribuzione

L'esempio Elenco di distribuzione amqsptl0 fornisce un esempio di inserimento di un messaggio in più code di messaggi. Si basa sull'esempio MQPUT amqsput0.

Esecuzione dell'esempio Elenco di distribuzioni, amqsptl0

L'esempio Elenco di distribuzione viene eseguito in modo simile agli esempi Put.

Prende i parametri seguenti:

- I nomi delle code
- I nomi dei gestori code

Questi valori vengono immessi come coppie. Ad esempio:

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

Le code vengono aperte utilizzando MQOPEN e i messaggi vengono inseriti nelle code utilizzando MQPUT. I codici di errore vengono restituiti se uno dei nomi della coda o del gestore code non viene riconosciuto.

Ricordarsi di definire i canali tra i gestori code in modo che i messaggi possano fluire tra di essi. Il programma di esempio non lo fa per voi.

Progettazione dell'esempio Elenco di distribuzione

I record di inserimento messaggi (MQPMR) specificano gli attributi dei messaggi per ciascuna destinazione. L'esempio fornisce valori per *MsgId* e *CorrelId*, e questi sovrascrivono i valori specificati nella struttura MQMD.

Il campo *PutMsgRecFields* nella struttura MQPMO indica quali campi sono presenti nei MQPMR:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

Successivamente, l'esempio assegna i record di risposta e i record oggetto. I record oggetto (MQORs) richiedono almeno una coppia di nomi e un numero pari di nomi, ovvero *ObjectName* e *ObjectQMgrName*.

La fase successiva prevede la connessione ai gestori code utilizzando MQCONN. L'esempio tenta di connettersi al gestore code associato alla prima coda in MQOR; se non riesce, passa attraverso i record dell'oggetto a turno. Si viene informati se non è possibile connettersi ad alcun gestore code e il programma si chiude.

Le code di destinazione vengono aperte tramite MQOPEN e il messaggio viene inserito in queste code mediante MQPUT. Eventuali problemi ed errori vengono riportati nei record di risposta (MQRR).

Infine, le code di destinazione vengono chiuse utilizzando MQCLOSE e il programma si disconnette dal gestore code utilizzando MQDISC. Gli stessi record di risposta vengono utilizzati per ogni chiamata che indica *CompCode* e *Reason*.

I programmi di esempio Echo

I programmi di esempio Echo riecheggiano un messaggio da una coda messaggi alla coda di risposta.

Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

I programmi devono essere eseguiti come programmi attivati.

Su sistemi di IBM i, AIX, Linux, and Windows , il loro unico input è una struttura MQTMC2 (messaggio trigger) che contiene il nome di una coda di destinazione e il gestore code. La versione COBOL utilizza il gestore code predefinito.

IBM i Su IBM i, affinché il processo di attivazione funzioni, assicurarsi che il programma di esempio Echo che si desidera utilizzare sia attivato dai messaggi in arrivo sulla coda SYSTEM.SAMPLE.ECHO. A tale scopo, specificare il nome del programma di esempio Echo che si desidera utilizzare nel campo *ApplId* della definizione del processo SYSTEM.SAMPLE.ECHOPROCESS. (Per tale operazione, è possibile utilizzare il comando CHGMQMPRC; per i dettagli, consultare Modifica processo MQ (CHGMQMPRC).) La coda di esempio ha un tipo di trigger FIRST, quindi, se ci sono già dei messaggi nella coda prima di eseguire l'esempio Richiesta, l'esempio Echo non viene attivato dai messaggi inviati.

Una volta impostata correttamente la definizione, avviare prima AMQSERV4 in un lavoro, quindi AMQSREQ4 in un'altro. È possibile utilizzare AMQSTRG4 invece di AMQSERV4, ma i potenziali ritardi di inoltro dei lavori potrebbero rendere meno semplice seguire ciò che sta accadendo.

Utilizzare i programmi di esempio Richiesta per inviare messaggi alla coda SYSTEM.SAMPLE.ECHO. I programmi di esempio Echo inviano un messaggio di risposta contenente i dati nel messaggio di richiesta alla coda di risposta specificata nel messaggio di richiesta.

Progettazione dei programmi di esempio Echo

Il programma apre la coda denominata nella struttura del messaggio trigger che è stata passata quando è stata avviata. (Per chiarezza, si fa riferimento a questa coda di richieste.) Il programma utilizza la chiamata MQOPEN per aprire questa coda per l'input condiviso.

Il programma utilizza la chiamata MQGET per rimuovere i messaggi da questa coda. Questa chiamata utilizza l'opzione MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT e MQGMO_WAIT, con intervallo di attesa di 5 secondi. Il programma verifica il descrittore di ciascun messaggio per verificare se si tratta di un messaggio di richiesta; in caso contrario, il programma elimina il messaggio e visualizza un messaggio di avvertenza.

Per ogni riga di input, il programma legge il testo in un buffer e utilizza la chiamata MQPUT1 per inserire un messaggio di richiesta, contenente il testo di quella riga, nella coda di risposta.

Se la chiamata MQGET ha esito negativo, il programma inserisce un messaggio di report nella coda di risposta, impostando il campo *Feedback* del descrittore del messaggio sul codice di errore restituito da MQGET.

Quando non ci sono messaggi rimanenti sulla coda di richiesta, il programma chiude tale coda e si disconnette dal gestore code.

IBM i Su IBM i, il programma può anche rispondere ai messaggi inviati alla coda da piattaforme diverse da IBM MQ for IBM i, sebbene non venga fornito alcun esempio per questa situazione. Per far funzionare il programma ECHO:

- Scrivere un programma, specificando correttamente i parametri **Format**, **Encoding**e **CCSID** , per inviare messaggi di richiesta di testo.

  Il programma ECHO richiede al gestore code di eseguire la conversione dei dati del messaggio, se necessario.
- Specificare CONVERT (*YES) sul canale di invio IBM MQ for IBM i , se il programma scritto non fornisce una conversione simile per la risposta.

I programmi di esempio Get

I programmi di esempio Get ricevono i messaggi da una coda utilizzando la chiamata MQGET.

Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

Progettazione del programma di esempio Get

Il programma apre la coda di destinazione utilizzando la chiamata MQOPEN con l'opzione MQOO_INPUT_AS_Q_DEF. Se non è in grado di aprire la coda, il programma visualizza un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN.

Per ogni messaggio sulla coda, il programma utilizza la chiamata MQGET per rimuovere il messaggio dalla coda, quindi visualizza i dati contenuti nel messaggio. La chiamata MQGET utilizza l'opzione MQGMO_WAIT, specificando un *WaitInterval* di 15 secondi, in modo che il programma attenda per questo periodo se non è presente alcun messaggio nella coda. Se non arriva alcun messaggio prima della scadenza di questo intervallo, la chiamata ha esito negativo e restituisce il codice di errore MQRC_NO_MSG_AVAILABLE.

Il programma dimostra in che modo è necessario cancellare i campi *MsgId* e *CorrelId* della struttura MQMD dopo ogni chiamata MQGET poiché la chiamata imposta questi campi sui valori contenuti nel messaggio richiamato. La cancellazione di questi campi significa che le chiamate MQGET successive richiamano i messaggi nell'ordine in cui sono conservati nella coda.

La chiamata MQGET specifica un buffer di dimensione fissa. Se un messaggio è più lungo di questo buffer, la chiamata non riesce e il programma si arresta.

Il programma continua fino a quando la chiamata MQGET non restituisce il codice motivo MQRC_NO_MSG_AVAILABLE o la chiamata MQGET non riesce. Se la chiamata ha esito negativo, il programma visualizza un messaggio di errore che contiene il codice di errore.

Il programma chiude quindi la coda utilizzando la chiamata MQCLOSE.

Esecuzione degli esempi amqsget e amqsgetc

Ognuno di questi programmi prende i parametri posizionali seguenti:

1. Il nome della coda di origine (obbligatorio)
2. Il nome del gestore code (facoltativo)

   Se non viene specificato un gestore code, **amqsget** si connette al gestore code predefinito e **amqsgetc** si connette al gestore code identificato dalla variabile di ambiente *MQSERVER* o dal file di definizione del canale del client.
3. Le opzioni di apertura (facoltativo)

   Se non vengono specificate opzioni di apertura, l'esempio utilizza un valore di 8193 che è la combinazione di queste due opzioni:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. Le opzioni di chiusura (facoltativo)

   Se non vengono specificate le opzioni di chiusura, l'esempio utilizza un valore 0 che è MQCO_NONE.

Utilizzare le seguenti variabili di ambiente per fornire credenziali utilizzate per l'autenticazione con il gestore code:

ID UTENTE MQSAMP
Impostare l'ID utente da utilizzare per l'autenticazione della connessione, se si desidera utilizzare un ID utente e una password per l'autenticazione con il gestore code. Il programma richiede la password per accompagnare l'ID utente.

V 9.3.4 Linux AIX **MQSAMP_TOKEN**
Impostare su un valore non vuoto se si desidera fornire un token di autenticazione per l'autenticazione con il gestore code. Il programma richiede il token di autenticazione. I token di autenticazione possono essere utilizzati solo dall'esempio **amqsgetc** che utilizza i bind client.

Per eseguire questi programmi, immettere uno dei seguenti:

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

dove *myqueue* è il nome della coda da cui il programma riceve i messaggi e *qmanagername* è il gestore code proprietario di *myqueue*.

Utilizzo di amqsget e amqsgetc

Si noti che **amqsget** esegue una connessione locale al gestore code, utilizzando la memoria condivisa per collegarsi al gestore code e, come tale, può essere eseguito solo sul sistema in cui risiede il gestore code, mentre **amqsgetc** esegue una connessione in stile client (anche se ci si connette a un gestore code sullo stesso sistema).

Quando si utilizza **amqsgetc** , è necessario fornire i dettagli dell'applicazione su come raggiungere effettivamente il gestore code, in termini di host del gestore code o indirizzo IP e porta del listener del gestore code.

Di solito, questa operazione viene eseguita utilizzando la variabile di ambiente **MQSERVER** o definendo i dettagli di connessione utilizzando una tabella di definizione del canale client, che può essere fornita anche a **amqsgetc** utilizzando le variabili di ambiente; ad esempio, consultare MQCCDTURL.

Un esempio che utilizza **MQSERVER**, che si connette a un gestore code localmente, che dispone di un listener in esecuzione sulla porta 1414 e che utilizza il canale di connessione server predefinito è:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

Programmi di esempio HA (High Availability)

I programmi di esempio ad alta disponibilità **amqsghac**, **amqsphac**e **amqsmhac** utilizzano la riconnessione client automatizzata per dimostrare il recupero in caso di errore di un gestore code. **amqsfhac** verifica che un gestore code che utilizza la memoria di rete mantenga l'integrità dei dati in seguito a un errore.

I programmi **amqsghac**, **amqsphac**e **amqsmhac** vengono avviati dalla riga comandi e possono essere utilizzati in combinazione per dimostrare la riconnessione dopo l'errore di un'istanza di gestore code a più istanze.

In alternativa, è anche possibile utilizzare gli esempi **amqsghac**, **amqsphac**e **amqsmhac** per dimostrare la riconnessione del client ai gestori code a istanza singola, generalmente configurati in un gruppo di gestori code.

Per semplificare l'esempio, in modo che sia facile da configurare, vengono mostrati i programmi di esempio che si riconnettono a una singola istanza del gestore code che viene avviata, arrestata e riavviata; consultare “Impostare e controllare il gestore code” a pagina 1099.

Utilizzare **amqsfhac** in parallelo con **amqmfsck** per verificare l'integrità del file system. Per ulteriori informazioni, consultare **amqmfsck** (controllo file system) e Verifica del funzionamento del file system condiviso .

amqsphac *queueName* [*qMgrNome*]

- **amqsphac** è un'applicazione IBM MQ MQI client . Inserisce una sequenza di messaggi in una coda con un ritardo di due secondi tra ciascun messaggio e visualizza gli eventi inviati al relativo gestore eventi.
- Nessun punto di sincronizzazione viene utilizzato per inserire i messaggi nella coda.
- La riconnessione può essere effettuata a qualsiasi gestore code nello stesso gruppo di gestori code.

amqsghac *queueName* [*qMgrNome*]

- **amqsghac** è un'applicazione IBM MQ MQI client . Richiama i messaggi da una coda e visualizza gli eventi inviati al gestore eventi.
- Nessun punto di sincronizzazione viene utilizzato per richiamare i messaggi dalla coda.
- La riconnessione può essere effettuata a qualsiasi gestore code nello stesso gruppo di gestori code.

amqsmhac -s *sourceQueueNome* -t *targetQueueNome* [-m *qMgrNome*] [-w *waitInterval*]

- **amqsmhac** è un'applicazione IBM MQ MQI client . Copia i messaggi da una coda a un'altra con un intervallo di attesa predefinito di 15 minuti dopo l'ultimo messaggio ricevuto prima del termine del programma.
- I messaggi vengono copiati nel punto di sincronizzazione.
- La riconnessione può essere effettuata solo allo stesso gestore code.

amqsfhac *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* (0 | 1 | 2)

- **amqsfhac** è un'applicazione IBM MQ MQI client . Verifica che un gestore code a più istanze IBM MQ che utilizza la memoria di rete, come un NAS o un file system cluster, mantenga l'integrità dei dati. Seguire i passi per eseguire **amqsfhac** in Verifica del funzionamento del file system condiviso.
- Utilizza l'opzione MQCNO_RECONNECT_Q_MGR durante la connessione a *QueueManagerName*. Si riconnette automaticamente quando si verifica il failover del gestore code.
- Inserisce i messaggi persistenti *InTransactionCount***RepeatCount* in *QueueName* durante il quale si verifica il failover del gestore code per un numero di volte. **amqsfhac** si riconnette al gestore code ogni volta e continua. Il test è per assicurarsi che nessun messaggio venga perso.

- *InTransactionConteggio* messaggi vengono inseriti all'interno di ciascuna transazione. La transazione viene ripetuta *RepeatCount* numero di volte. Se si verifica un errore in una transazione, **amqsfhac** esegue il rollback e reinoltra la transazione quando **amqsfhac** si riconnette al gestore code.
- Inserisce inoltre i messaggi in *SideQueueNome*. Utilizza il nome *SideQueue* per controllare se è stato eseguito correttamente il commit o il rollback di tutti i messaggi da *QueueName* . Se rileva un'incongruenza, scrive un messaggio di errore.
- Modificare la quantità di traccia di output da **amqsfhac** impostando l'ultimo parametro su `(0|1|2)`.

  0
  Output minimo.

  1
  Output intermedio.

  2
  La maggior parte dell'output.

Configurazione di una connessione client

È necessario configurare un canale di connessione client e server per eseguire gli esempi. La procedura di verifica client spiega come impostare un ambiente di test client.

In alternativa, utilizzare la configurazione fornita nel seguente esempio.

Esempio di utilizzo di amqsghac, amqsphace amqsmhac

L'esempio illustra i client ricollegabili che utilizzano un gestore code a istanza singola.

I messaggi vengono collocati nella coda SOURCE da **amqsphac**, trasferiti a TARGET da **amqsmhac**e richiamati da TARGET da **amqsghac** ; consultare Figura 129 a pagina 1098.

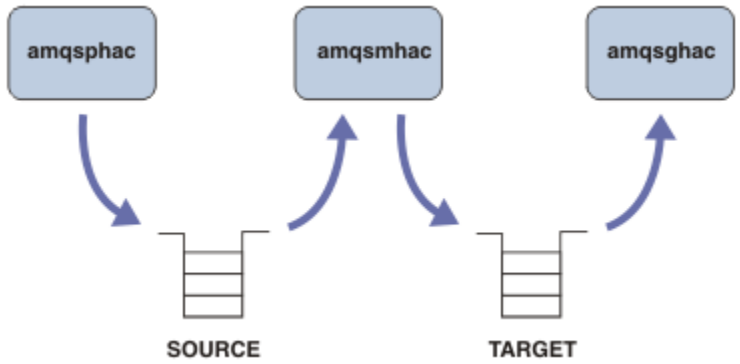


Figura 129. Esempi di client ricollegabili

Seguire questa procedura per eseguire gli esempi.

1. Creare un file `hasamples.tst` contenente i comandi:

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. Immettere i seguenti comandi in una richiesta comandi:

   a. `crtmqm QM1`

b. `strmqm QM1`

c. `runmqsc QM1 < hasamples.tst`

3. Impostare il valore della variabile d'ambiente **MQCHLLIB** sul percorso del file di definizione del canale client AMQCLCHL.TAB , ad esempio `SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Aprire tre nuove finestre con **MQCHLLIB** impostato; ad esempio in Windows, immettere **start** tre volte al prompt dei comandi precedente, avviando ciascun programma in una delle finestre. Fare riferimento al punto “5” a pagina 1100 in “Impostare e controllare il gestore code” a pagina 1099.)
5. Immettere il comando `endmqm -r -p QM1` per arrestare il gestore code e consentire ai client di riconnettersi.
6. Immettere il comando `strmqm QM1` per riavviare il gestore code.

I risultati dell'esecuzione degli esempi **amqsghac**, **amqsphac**e **amqsmhac** su Windows sono riportati nei seguenti esempi.

Impostare e controllare il gestore code

1. Creare il gestore code.

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Ricordare la directory dei dati per impostare la variabile **MQCHLLIB** in un secondo momento.

2. Avviare il gestore code.

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. Creare le code e i canali, modificare la porta del listener e avviare il listener e il canale.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. Rendere nota la tabella del canale client ai client.

Utilizzare la directory di dati restituita dal comando **crtmqm** nel passo “1” a pagina 1099e aggiungere la directory `@ipcc` per impostare la variabile **MQCHLLIB** .

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. Avviare i programmi di esempio nelle altre finestre

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. Terminare il gestore code e riavviarlo.

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

Attività correlate

Verifica del funzionamento del file system condiviso

Riferimenti correlati

amqmfsck (controllo file system)

I programmi di esempio Inquire

I programmi di esempio di interrogazione analizzano alcuni attributi di una coda che utilizza la chiamata MQINQ.

Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

Questi programmi sono concepiti per essere eseguiti come programmi attivati, quindi il loro unico input è una struttura MQTMC2 (messaggio trigger) per IBM MQ for Multiplatforms. Questa struttura contiene il nome di una coda di destinazione con attributi su cui eseguire l'interrogazione. La versione C utilizza anche il nome del gestore code. La versione COBOL utilizza il gestore code predefinito.

Affinché il processo di attivazione funzioni, assicurarsi che il programma di esempio Inquire che si desidera utilizzare sia attivato dai messaggi che arrivano sulla coda SYSTEM.SAMPLE.INQ. A tale scopo, specificare il nome del programma di esempio di interrogazione che si desidera utilizzare nel campo *ApplicId* della definizione del processo SYSTEM.SAMPLE.INQPROCESS. IBM i Per IBM i, è possibile utilizzare il comando CHGMQMPRC per questo; per i dettagli, vedere Modifica del processo MQ (CHGMQMPRC). La coda di esempio ha un tipo di trigger FIRST; se ci sono già dei messaggi nella coda prima di eseguire l'esempio di richiesta, l'esempio inquire non viene attivato dai messaggi inviati.

Una volta impostata correttamente la definizione:

- ALW Per AIX, Linux, and Windows, avviare il programma **runmqtrm** in una sessione, quindi avviare il programma `amqsreq` in un altro.
- IBM i Per IBM i, avviare il programma AMQSERV4 in una sessione, quindi avviare il programma AMQSREQ4 in un altro. È possibile utilizzare AMQSTRG4 invece di AMQSERV4, ma i potenziali ritardi di inoltro dei lavori potrebbero rendere meno semplice seguire ciò che sta accadendo.

Utilizzare i programmi di esempio Richiesta per inviare messaggi di richiesta, ciascuno contenente solo un nome coda, alla coda SYSTEM.SAMPLE.INQ. Per ogni messaggio di richiesta, i programmi di esempio Inquire inviano un messaggio di risposta contenente informazioni sulla coda specificata nel messaggio di richiesta. Le risposte vengono inviate alla coda di risposta specificata nel messaggio di richiesta.

IBM i Su IBM i, se il membro del file di input di esempio QMQMSAMP.AMQSDATA(INQ), l'ultimo nome della coda non esiste, quindi l'esempio restituirà un messaggio di report con un codice motivo per l'errore.

Progettazione del programma di esempio Inquire

Il programma apre la coda denominata nella struttura del messaggio trigger che è stata passata quando è stata avviata. (Per chiarezza, chiameremo questa *coda di richiesta*.) Il programma utilizza la chiamata MQOPEN per aprire questa coda per l'input condiviso.

Il programma utilizza la chiamata MQGET per rimuovere i messaggi da questa coda. Questa chiamata utilizza le opzioni MQGMO_ACCEPT_TRUNCATED_MSG e MQGMO_WAIT, con un intervallo di attesa di 5 secondi. Il programma verifica il descrittore di ciascun messaggio per verificare se si tratta di un messaggio di richiesta; in caso contrario, il programma elimina il messaggio e visualizza un messaggio di avvertenza.

Per ogni messaggio di richiesta rimosso da una coda di richiesta, il programma legge il nome della coda (che chiameremo *coda di destinazione*) e apre la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_INQ. Il programma utilizza quindi la chiamata MQINQ per richiedere informazioni sui valori degli attributi *InhibitGet*, **CurrentQDepth**e **OpenInputCount** della coda di destinazione.

Se la chiamata MQINQ ha esito negativo, il programma utilizza la chiamata MQPUT1 per inserire un messaggio di risposta nella coda di risposta. Questo messaggio contiene i valori dei tre attributi.

Se la chiamata MQOPEN o MQINQ ha esito negativo, il programma utilizza la chiamata MQPUT1 per inserire un messaggio di report nella coda di risposta. Nel campo *Feedback* del descrittore del messaggio di questo messaggio di report è presente il codice motivo restituito dalla chiamata MQOPEN o MQINQ, a seconda di quale non è riuscito.

Dopo la chiamata MQINQ, il programma chiude la coda di destinazione utilizzando la chiamata MQCLOSE.

Quando non ci sono messaggi rimanenti sulla coda di richiesta, il programma chiude tale coda e si disconnette dal gestore code.

Il programma di esempio Inquire Properties of a Message Handle

AMQSIQMA è un programma C di esempio per analizzare le proprietà di un handle del messaggio da una coda messaggi ed è un esempio dell'utilizzo della chiamata API MQINQMP.

Questo esempio crea un handle del messaggio e lo inserisce nel campo MsgHandle della struttura MQGMO. L'esempio ottiene quindi un messaggio e interroga e stampa tutte le proprietà con cui è stato popolato l'handle del messaggio.

```
C:\Program Files\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

Programmi di esempio di pubblicazione / sottoscrizione

I programmi di esempio di pubblicazione / sottoscrizione dimostrano l'utilizzo delle funzioni di pubblicazione e sottoscrizione in IBM MQ.

Esistono tre programmi di esempio in linguaggio C che illustrano come programmare l'interfaccia di pubblicazione / sottoscrizione IBM MQ . Ci sono alcuni esempi C che utilizzano interfacce più vecchie e ci sono esempi Java . Gli esempi Java utilizzano l'interfaccia di pubblicazione / sottoscrizione IBM MQ in com.ibm.mq.jar e l'interfaccia di pubblicazione / sottoscrizione JMS in com.ibm.mqjms. Gli esempi JMS non sono trattati in questo argomento.

C

Trovare l'esempio del publisher `amqspub` nella cartella degli esempi C . Eseguirlo con qualsiasi nome di argomento che si desidera come primo parametro, seguito da un nome di gestore code facoltativo. Ad esempio, `amqspub mytopic QM3` . Esiste anche una versione client denominata `amqspubc`. Se si sceglie di eseguire la versione del client, consultare prima “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072 per i dettagli.

Il publisher si connette al gestore code predefinito e risponde con l'output, `target topic is mytopic` . Ogni riga immessa in questa finestra da questo momento in poi viene pubblicata in `mytopic` .

Aprire un'altra finestra di comandi nella stessa directory ed eseguire il programma sottoscrittore (subscriber), `amqssub`, fornendogli lo stesso nome argomento e un nome gestore code facoltativo. Ad esempio, `amqssub mytopic QM3` .

Il sottoscrittore risponde con l'output, `Calling MQGET : 30 seconds wait time` . Da ora in poi, le righe immessa nel publisher vengono visualizzate nell'output del sottoscrittore.

Avviare un altro sottoscrittore (subscriber) in un'altra finestra di comandi e controllare che entrambi i sottoscrittori ricevano le pubblicazioni.

Per la documentazione completa dei parametri, incluse le opzioni di impostazione, fare riferimento al codice sorgente di esempio. I valori per il campo delle opzioni del sottoscrittore sono descritti nel seguente argomento: Opzioni (MQLONG).

Esiste un altro esempio di sottoscrittore (subscriber) `amqssbx`, che offre ulteriori opzioni di sottoscrizione come switch della riga comandi.

Immettere `amqssbx -d mysub -t mytopic -k` per richiamare il sottoscrittore utilizzando sottoscrizioni durevoli che vengono conservate dopo la chiusura del sottoscrittore.

Verificare la sottoscrizione pubblicando un altro elemento utilizzando il publisher. Attendere per 30 secondi la chiusura del sottoscrittore. Pubblicare altri elementi nello stesso argomento. Riavviare il sottoscrittore. L'ultimo elemento pubblicato mentre il sottoscrittore non era in esecuzione viene visualizzato dal sottoscrittore immediatamente riavviato.

Legacy C

Esiste una ulteriore serie di esempi C che mostrano i comandi accodati. Alcuni di questi esempi sono stati originariamente forniti come parte del supporto MQ0C . Le funzioni dimostrate negli esempi sono completamente supportate, per motivi di compatibilità.

Si sconsiglia di utilizzare l'interfaccia comandi in coda. È molto più complessa dell'API di pubblicazione / sottoscrizione e non vi è alcun motivo funzionale convincente per programmare comandi accodati complessi. Tuttavia, si potrebbe trovare l'approccio accodato più adatto, forse perché si sta già utilizzando l'interfaccia o perché l'ambiente di programmazione semplifica la creazione di un messaggio complesso e la chiamata di un MQPUT generico, piuttosto che la creazione di chiamate differenti a MQSUB.

Gli esempi aggiuntivi si trovano nella sottodirectory `pubsub` nella cartella `samples` .

Esistono sei tipi di esempio elencati in Tabella 163 a pagina 1103.

Tabella 163. Categorie di programmi di esempio C di pubblicazione / sottoscrizione legacy

Categoria	Programmi	Commenti
RFH1	`amqssr1a.c` `amqspr1a.c`	Esempio di pubblicazione / sottoscrizione semplice creato utilizzando i messaggi in formato RFH1 .
RFH2	`amqssr2a.c` `amqspr2a.c`	Esempio di pubblicazione / sottoscrizione semplice creato utilizzando i messaggi in formato RFH2 .
Esempi MQAI	`amqsppca.c` `amqsspca.c`	Esempio di pubblicazione / sottoscrizione semplice creato utilizzando i comandi PCF e l'interfaccia comandi MQAI.
MA0C Servizio risultati utilizzando RFH1	`amqsgama.c` `amqsresa.c`	Servizio dei risultati creato utilizzando intestazioni RFH1 1. Richiede le code definite in `amqsgama.tst` e `amqsresa.tst` 2. `amqsresa` deve essere avviato prima `amqsgama`
MA0C Servizio risultati utilizzando RFH2	`amqsgr2a.c` `amqsrr2a.c`	Servizio dei risultati creato utilizzando intestazioni RFH2 1. Richiede le code definite in `amqsgama.tst` e `amqsresa.tst` 2. `amqsresa` deve essere avviato prima `amqsgama`
Esempio di pubblicazione / sottoscrizione dell'uscita di instradamento	`amqspsra.c`	Dimostra come modificare la destinazione della coda o del gestore code per un messaggio di pubblicazione / sottoscrizione in un'uscita di instradamento.

Programma di esempio per Java

L' Java esempio `MQPubSubApiSample.java` combina publisher e sottoscrittori in un singolo programma. I relativi file di origine e di classe compilati si trovano nella cartella degli esempi `wmqjava` .

Se si sceglie di eseguire in modalità client, consultare prima “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072 per i dettagli.

Eseguire l'esempio dalla riga comandi utilizzando il comando Java , se è stato configurato un ambiente Java . È anche possibile eseguire l'esempio dallo spazio di lavoro IBM MQ Explorer Eclipse che ha un workbench di programmazione Java già configurato.

Potrebbe essere necessario modificare alcune delle proprietà del programma di esempio per eseguirlo. A tale scopo, fornire i parametri alla JVM o modificare l'origine.

Le istruzioni in “Esecuzione dell'esempio MQPubSubApiSample Java” a pagina 1104 mostrano come eseguire l'esempio dallo spazio di lavoro Eclipse .

Esecuzione dell'esempio MQPubSubApiSample Java
Come eseguire MQPubSubApiSample utilizzando Java Development Tools dalla piattaforma Eclipse .

Prima di iniziare

Aprire il workbench Eclipse . Creare una nuova directory dello spazio di lavoro e selezionarla. Chiudere la finestra di benvenuto.

Seguire la procedura descritta in “Configurazione di un gestore code per accettare connessioni client su Multiplatforms” a pagina 1072 prima di eseguire come client.

Informazioni su questa attività

Il programma di esempio di pubblicazione / sottoscrizione Java è un programma IBM MQ MQI client Java . L'esempio viene eseguito senza modifiche utilizzando un gestore code predefinito in ascolto sulla porta 1414. L'attività descrive questo caso semplice e indica in termini generali come fornire parametri e modificare l'esempio per adattarlo alle diverse configurazioni IBM MQ . L'esempio è illustrato in esecuzione su Windows. I percorsi dei file differiranno su altre piattaforme.

Procedura

1. Importazione dei programmi di esempio Java
   a) Nel workbench, selezionare **Finestra** > **Apri prospettiva** > **Altro** > **Java** e selezionare **OK**.
   b) Passare alla vista **Esplora package** .
   c) Fare clic con il tasto destro del mouse nello spazio vuoto nella vista **Esplora package** . Fare clic su **Nuovo** > **Progetto Java**.
   d) Nel campo **`Project name`** immettere `MQ Java Samples`. Fare clic su **Avanti**.
   e) Nel pannello **`Java Settings`** , passare alla scheda **Librerie** .
   f) Fare clic su **Aggiungi JAR esterni**.
   g) Passare a *MQ_INSTALLATION_PATH* `\java\lib` dove *MQ_INSTALLATION_PATH* è la cartella di installazione di IBM MQ e selezionare `com.ibm.mq.jar` e `com.ibm.mq.jmqi.jar`
   h) Fare clic su **Apri** > **Fine.**
   i) Fare clic con il pulsante destro del mouse su `src` nella vista **Esplora package** .
   j) Selezionare **Importa ...** > **Generale** > **File System** > **Avanti** > **Sfoglia**... e passare al percorso *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` dove *MQ_INSTALLATION_PATH* è la directory di installazione di IBM MQ .
   k) Nel riquadro **Importa** , Figura 130 a pagina 1105, fare clic su `samples` (senza selezionare la check box).
   l) Selezionare `MQPubSubApiSample.java`. Il campo **`Into folder`** deve contenere `MQ Java Samples/src`. Fare clic su **Fine**.

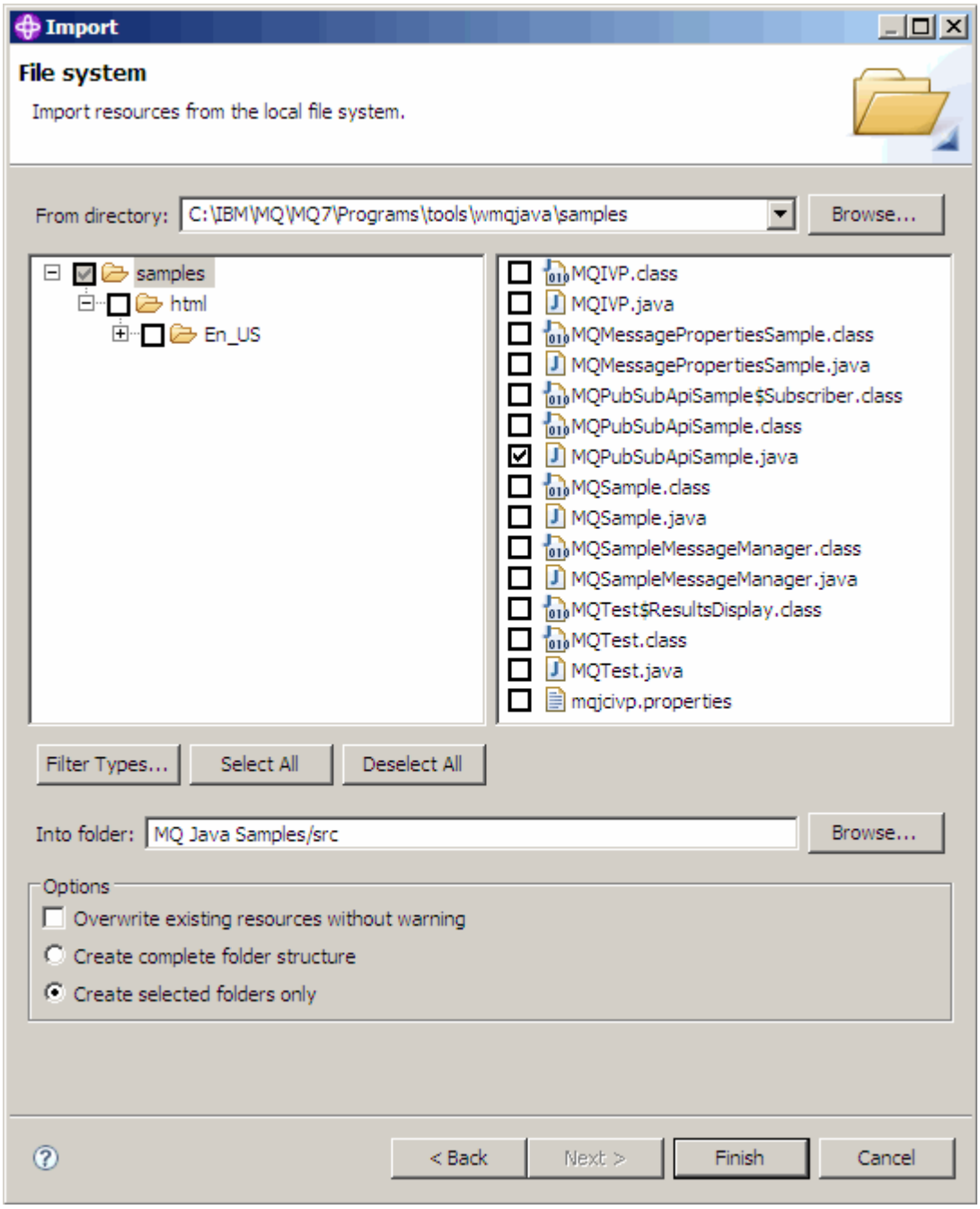


Figura 130. Importazione file system

2. Eseguire il programma di esempio di pubblicazione / sottoscrizione.

   Esistono due modi per eseguire il programma, a seconda che sia necessario modificare i parametri predefiniti.

   - La prima scelta esegue il programma senza apportare alcuna modifica:
     - Nel menu principale dello spazio di lavoro, espandere la cartella `src` . Fare clic con il tasto destro del mouse su **MQPubSubApiSample.java Run - as** > **1. Java Applicazione**
   - La seconda scelta esegue il programma con parametri o con codice sorgente modificato per il proprio ambiente:
     - Aprire `MQPubSubApiSample.java` e studiare il costruttore `MQPubSubApiSample` .

– Modificare gli attributi del programma.

  Questi attributi sono modificabili utilizzando lo switch JVM -D oppure fornendo un valore predefinito per la proprietà System pmodificando il codice sorgente.

  - topicObject
  - queueManagerName
  - subscriberCount

  Questi attributi sono modificabili solo modificando il codice sorgente nel costruttore.

  - nome host
  - porta
  - canale

  Per impostare le proprietà System p, codificare un valore predefinito nell'accessor, ad esempio:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

  Oppure fornire il parametro alla JVM utilizzando l'opzione `-D` , come mostrato nei seguenti passi:

a. Copiare il nome completo di System.Property che si desidera impostare, ad esempio: `com.ibm.mq.pubSubSample.queueManagerName`.
b. Nello spazio di lavoro, fare clic con il tasto destro del mouse su **Esegui** > **Apri finestra Esegui**. Fare doppio clic su Java Applicazione in **Crea, Gestisci ed Esegui applicazioni** e fare clic sulla scheda **(x) = Argomenti** .
c. Nel riquadro **Argomenti VM:** , immettere `-D` e incollare il nome System.property , `com.ibm.mq.pubSubSample.queueManagerName`, seguito da `=QM3`. Fare clic su **Applica** > **Esegui**.
d. Aggiungere ulteriori argomenti come un elenco separato da virgole o come righe aggiuntive nel riquadro, senza separatori di virgole.

   Ad esempio: `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3,`
   `-Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

Il programma di esempio Pubblica uscita

AMQSPSE0 è un programma C di esempio di un'uscita per intercettare una pubblicazione prima di consegnarla a un sottoscrittore. L'uscita può quindi, ad esempio, modificare le intestazioni del messaggio, il payload o la destinazione oppure impedire che il messaggio venga pubblicato in un sottoscrittore.

Per eseguire l'esempio, eseguire le seguenti attività:

1. Configurare il gestore code:
   - Linux AIX Sui sistemi AIX and Linux aggiungere una sezione come questa al file `qm.ini` :

```
PublishSubscribe:
PublishExitPath=Module
PublishExitFunction=EntryPoint
```

     dove il modulo è *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`. *MQ_INSTALLATION_PATH* rappresenta la directory di livello superiore in cui è installato IBM MQ .
   - Windows Su Windows , impostare gli attributi equivalenti nel registro.
2. Assicurarsi che il modulo sia accessibile a IBM MQ.
3. Riavviare il gestore code per selezionare la configurazione.

4. Nel processo dell'applicazione da tracciare, descrivere dove devono essere scritti i file di traccia. Ad esempio:
   - Linux AIX Su sistemi AIX and Linux , assicurarsi che la directory `/var/mqm/trace` esista ed esportare la variabile di ambiente **MQPSE_TRACE_LOGFILE** :

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows Su Windows, assicurarsi che la directory `C:\temp` esista e impostare la variabile di ambiente **MQPSE_TRACE_LOGFILE** :

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

Programmi di esempio Put

I programmi di esempio Put inserano i messaggi su una coda utilizzando la chiamata MQPUT.

Consultare "Funzioni dimostrate nei programmi di esempio su Multiplatforms" a pagina 1063 per i nomi di questi programmi.

Progettazione del programma di esempio Put

Il programma utilizza la chiamata MQOPEN con l'opzione MQOO_OUTPUT per aprire la coda di destinazione per inserire i messaggi.

Se non riesce ad aprire la coda, il programma emette un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN. Per semplificare il programma, in questa e nelle successive chiamate MQI, il programma utilizza i valori predefiniti per molte opzioni.

Per ogni riga di input, il programma legge il testo in un buffer e utilizza la chiamata MQPUT per creare un messaggio datagramma contenente il testo di tale riga. Il programma continua fino a quando non raggiunge la fine dell'input o la chiamata MQPUT ha esito negativo. Se il programma raggiunge la fine dell'input, chiude la coda utilizzando la chiamata MQCLOSE.

Esecuzione dei programmi di esempio Put

Esecuzione degli esempi amqsput e amqsputc

ALW

L'esempio **amqqsput** è il programma per inserire i messaggi utilizzando i collegamenti locali e l'esempio **amqsputc** è il programma per inserire i messaggi utilizzando i collegamenti client. Ognuno di questi programmi prende i parametri posizionali seguenti:

1. Il nome della coda di destinazione (obbligatorio)
2. Il nome del gestore code (facoltativo)

   Se non viene specificato un gestore code, **amqsput** si connette al gestore code predefinito e **amqsputc** si connette al gestore code identificato dalla variabile di ambiente *MQSERVER* o dal file di definizione del canale del client.
3. Le opzioni di apertura (facoltativo)

   Se non vengono specificate le opzioni di apertura, l'esempio utilizza un valore di 8208 che è la combinazione di queste due opzioni:
   - OUTPUT MQOO
   - MQOO_FAIL_IF_QUIESCING
4. Le opzioni di chiusura (facoltativo)

   Se non vengono specificate le opzioni di chiusura, l'esempio utilizza un valore 0 che è MQCO_NONE.

5. Il nome del gestore code di destinazione (facoltativo)

   Se non viene specificato un gestore code di destinazione, il campo `ObjectQMgrName` nell'MQOD verrà lasciato vuoto.

6. Il nome della coda dinamica (facoltativo)

   Se non viene specificato un nome coda dinamica, il campo `DynamicQName` in MQOD verrà lasciato vuoto.

Utilizzare le seguenti variabili di ambiente per fornire credenziali utilizzate per l'autenticazione con il gestore code:

ID UTENTE MQSAMP
Impostare l'ID utente da utilizzare per l'autenticazione della connessione, se si desidera utilizzare un ID utente e una password per l'autenticazione con il gestore code. Il programma richiede la password per accompagnare l'ID utente.

MQSAMP_TOKEN
Impostare su un valore non vuoto se si desidera fornire un token di autenticazione per l'autenticazione con il gestore code. Il programma richiede il token di autenticazione. I token di autenticazione possono essere utilizzati solo dall'esempio **amqsputc** che utilizza i bind client.

Per eseguire questi programmi, immettere uno dei seguenti comandi:

- `amqsput` *`myqueue qmanagername`*
- `amqsputc` *`myqueue qmanagername`*

dove *myqueue* è il nome della coda in cui verranno inseriti i messaggi e *qmanagername* è il gestore code che possiede *myqueue*.

Esecuzione dell'esempio amq0put

La versione COBOL non ha alcun parametro. Si connette al gestore code predefinito e quando viene eseguito viene richiesto:

```
Please enter the name of the target queue
```

Prende l'input da StdIn e aggiunge ogni riga di input alla coda di destinazione. Una riga vuota indica che non ci sono più dati.

Esecuzione dell'esempio AMQSPUT4 C (IBM i)

Il programma C AMQSPUT4, disponibile solo per la piattaforma IBM i , crea i messaggi leggendo i dati da un membro di un file origine.

È necessario specificare il nome del file come parametro quando si avvia il programma. La struttura del file deve essere:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

Un esempio di input per gli esempi di inserimento viene fornito nella libreria PUT del membro AMQSDATA del file QMQMSAMP.

Nota: Tenere presente che i nomi delle code sono sensibili al maiuscolo / minuscolo. Tutte le code create dal programma di creazione del file di esempio AMQSAMP4 hanno nomi creati in caratteri maiuscoli.

Il programma C inserisce i messaggi nella coda denominata nella prima linea del file; è possibile utilizzare la coda fornita SYSTEM.SAMPLE.LOCAL. Il programma inserisce il testo di ciascuna delle seguenti righe del file in messaggi datagramma separati e si arresta quando legge una riga vuota alla fine del file.

Utilizzando il file di dati di esempio il comando è:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

Esecuzione dell'esempio COBOL AMQ0PUT4 (IBM i)

IBM i

Il programma COBOL AMQ0PUT4, disponibile solo sulla piattaforma IBM i , crea messaggi accettando i dati dalla tastiera.

Per avviare il programma, richiamare il programma e fornire il nome della coda di destinazione come parametro del programma. Il programma accetta input dalla tastiera in un buffer e crea un messaggio datagramma per ogni riga di testo. Il programma si arresta quando si immette una riga vuota sulla tastiera.

Programmi di esempio del messaggio di riferimento

Gli esempi di messaggi di riferimento consentono di trasferire un oggetto di grandi dimensioni da un nodo a un altro (di solito su sistemi differenti) senza la necessità di memorizzare l'oggetto nelle code IBM MQ nei nodi di origine o di destinazione.

Viene fornita una serie di programmi di esempio per dimostrare come i messaggi di riferimento possono essere inseriti in una coda, ricevuti dalle uscite dei messaggi e presi da una coda. I programmi di esempio utilizzano i messaggi di riferimento per spostare i file. Se si desidera spostare altri oggetti come i database o se si desidera eseguire controlli di sicurezza, definire la propria uscita, in base all'esempio, amqsxrm.

La versione del programma di esempio di uscita del messaggio di riferimento da utilizzare dipende dalla piattaforma su cui è in esecuzione il canale:

- Su tutte le piattaforme, utilizzare amqsxrma all'estremità di invio.
- Utilizzare amqsxrma all'estremità di ricezione se il ricevitore è in esecuzione su qualsiasi piattaforma tranne IBM i.
- IBM i Se il ricevitore è in esecuzione in IBM i, utilizzare amqsxrm4.

.

IBM i *Note per gli utenti IBM i*

Per ricevere un messaggio di riferimento utilizzando l'uscita del messaggio di esempio, specificare un file nel file system root di IFS o in qualsiasi sottoindirizzario in modo che sia possibile creare un file di flusso.

L'uscita del messaggio di esempio su IBM i crea il file, converte i dati in EBCDIC e imposta la codepage sulla codepage del sistema. È quindi possibile copiare questo file in QSYS.LIB LIB utilizzando il comando CPYFRMSTMF. Ad esempio:

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

Il comando CPYFRMSTMF non crea il file. È necessario crearla prima di eseguire questo comando.

Se si invia un file da QSYS.LIB, non è richiesta alcuna modifica agli esempi. Per qualsiasi altro file system, assicurarsi che il CCSID specificato nel campo CodedCharSetId nella struttura MQRMH corrisponda ai dati di massa che si stanno inviando.

Quando si utilizza l'IFS (integrated file system), creare moduli di programma con l'opzione SYSIFCOPT (*IFSIO) impostata. Se si desidera spostare il database o i file di record a lunghezza fissa, definire la propria uscita in base all'esempio fornito AMQSXRM4.

Il metodo consigliato per trasferire un file di database consiste nel convertirlo in struttura IFS, utilizzando il comando CPYTOSTMF, quindi inviare il messaggio di riferimento allegando il file IFS. Se si sceglie di trasferire un file di database facendo riferimento ad esso dall'interno di IFS, ma non lo si converte nella struttura IFS, è necessario specificare il nome del membro. L'integrità dei dati non è garantita se si sceglie questo metodo.

Multi *Esecuzione degli esempi del messaggio di riferimento*

Utilizzare questo esempio per informazioni su come eseguire l'applicazione di esempio del messaggio di riferimento AMQSPRM su AIX, Linux, and Windowso AMQSPRMA su IBM i. L'esempio mostra come i messaggi di riferimento possono essere inseriti in una coda, ricevuti dalle uscite dei messaggi e presi da una coda.

Gli esempi del messaggio di riferimento vengono eseguiti come segue:

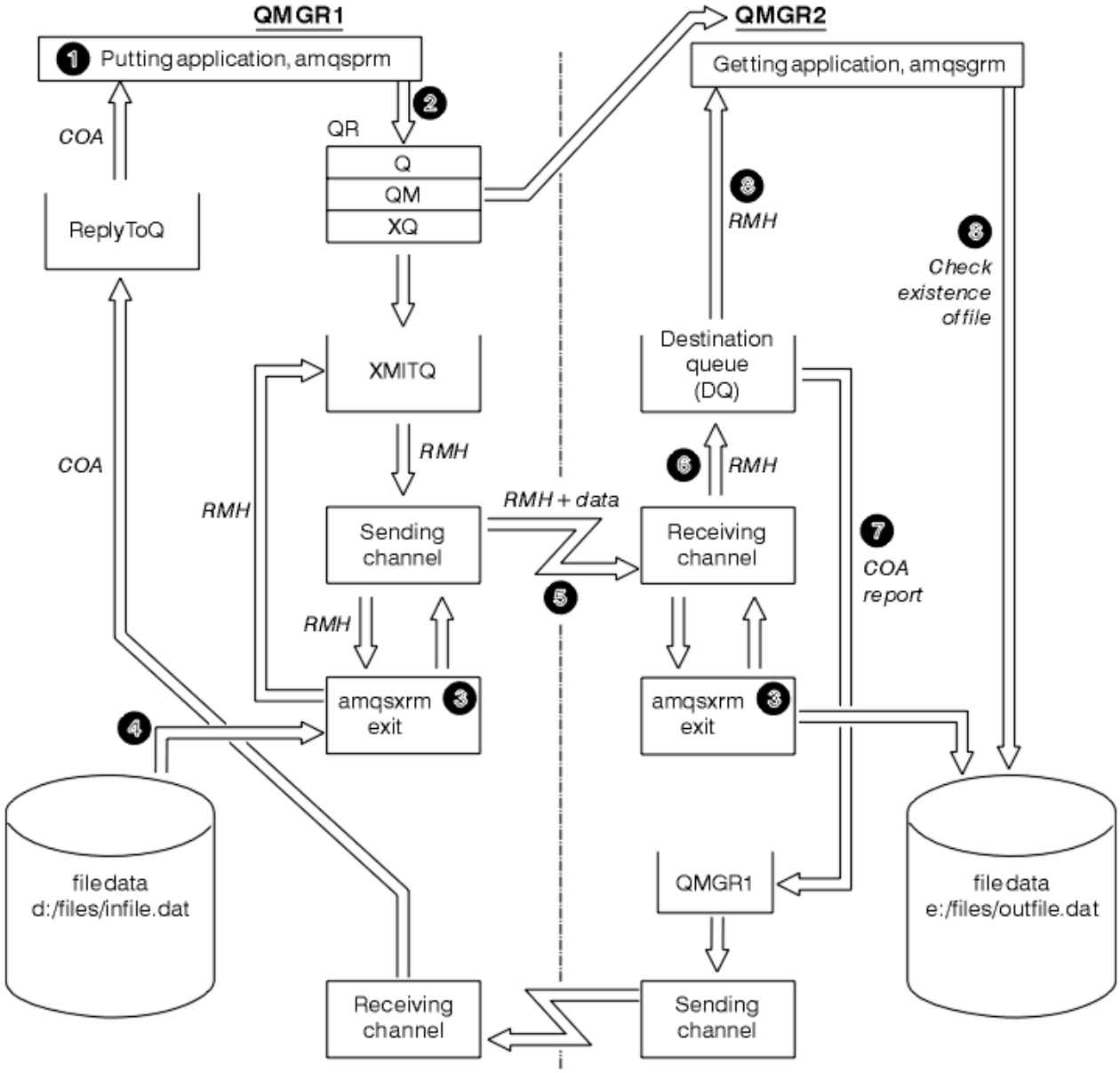


Figura 131. Esecuzione degli esempi del messaggio di riferimento

1. Impostare l'ambiente per avviare i listener, i canali e i controlli dei trigger e definire i canali e le code.

   Per descrivere come impostare il messaggio di riferimento, questo esempio fa riferimento alla macchina mittente come MACHINE1 con un gestore code denominato QMGR1 e la macchina ricevente come MACHINE2 con un gestore code denominato QMGR2.

Nota: Le seguenti definizioni consentono di creare un messaggio di riferimento per inviare un file con un tipo di oggetto FLATFILE dal gestore code QMGR1 a QMGR2 e per creare nuovamente il file come definito nella chiamata ad AMQSPRM (o AMQSPRMA su IBM i). Il messaggio di riferimento (inclusi i dati del file) viene inviato utilizzando il canale CHL1 e la coda di trasmissione XMITQ e inserito nella coda DQ. I report di eccezione e COA vengono inviati nuovamente a QMGR1 utilizzando il canale REPORT e la coda di trasmissione QMGR1.

L'applicazione che riceve il messaggio di riferimento (AMQSGRM o AMQSGRMA su IBM i) viene attivata utilizzando la coda di avvio INITQ e elaborando PROC. Assicurarsi che i campi CONNAME siano impostati correttamente e che il campo MSGEXIT rifletta la struttura di directory, in base al tipo di macchina e alla posizione di installazione del prodotto IBM MQ .

IBM i Le definizioni MQSC hanno utilizzato uno stile AIX per definire le uscite, quindi se si utilizza MQSC su IBM i, è necessario modificarle di conseguenza. È importante notare che i dati del messaggio FLATFILE sono sensibili al maiuscolo / minuscolo e l'esempio non funzionerà a meno che non siano in maiuscolo.

Sulla macchina MACHINE1, gestore code QMGR1

Sintassi MQSC

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **IBM i sintassi dei comandi**

Nota: Se non si specifica un nome di gestore code, il sistema utilizza il gestore code predefinito.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
            REPLACE(*YES) TRPTYPE(*TCP) +
            CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
            MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ   QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
            REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
            MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ   QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
            REPLACE(*YES) RMTQNAME(DQ) +
            RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

Sulla macchina MACHINE2, gestore code QMGR2

Sintassi MQSC

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i sintassi dei comandi

Nota: Su IBM i, se non si specifica un nome gestore code, il sistema utilizza il gestore code predefinito.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. Una volta creati gli oggetti IBM MQ :
   a. Dove applicabile alla piattaforma, avviare il listener per i gestori code di invio e ricezione
   b. Avviare i canali CHL1 e REPORT
   c. Sul gestore code di ricezione avviare il controllo trigger per la coda di iniziazione INITQ
3. Richiamare il programma di esempio AMQSPRM (AMQSPRMA on IBM i) dalla riga comandi utilizzando i seguenti parametri:

   -m
   Nome del gestore code locale; il valore predefinito è il gestore code predefinito

   -i
   Nome e ubicazione del file di origine

   -o
   Nome e ubicazione del file di destinazione

   -q
   Il nome della coda

   -g
   Il nome del gestore code in cui è presente la coda, definita nel parametro -q. Il valore predefinito è il gestore code specificato nel parametro -m.

   -t
   Tipo oggetto

   -w
   Intervallo di attesa, ossia il tempo di attesa per i report di eccezione e COA dal gestore code di ricezione

   Ad esempio, per utilizzare l'esempio con gli oggetti definiti precedentemente, utilizzare i seguenti parametri:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   L'aumento del tempo di attesa consente l'invio di un file di grandi dimensioni attraverso una rete prima che il programma metta in timeout i messaggi.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

IBM i: IBM i Su IBM i, completare la seguente procedura:

a. Utilizzare il seguente comando:

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

Ciò presuppone che il file originale `rmsg1` si trovi nella directory IFS `/refmsgs` e che si desidera che il file di destinazione sia `rmsgx` nella directory IFS `/refmsgs` sul sistema di destinazione.

b. Creare il proprio indirizzario utilizzando il comando CRTDIR invece di utilizzare l'indirizzario root.

c. Quando si richiama il programma che inserisce i dati, tenere presente che il nome del file di output deve riflettere la convenzione di denominazione IFS; ad esempio /TEST/FILENAME crea un file denominato FILENAME nella directory TEST.

Nota:

IBM i Su IBM i, è possibile utilizzare una barra (/) o un trattino (-) quando si specificano i parametri. Ad esempio:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX Per le piattaforme AIX and Linux , è necessario utilizzare due barre rovesciate (\\) invece di una per indicare la directory del file di destinazione. Pertanto, il comando **amqsprm** è simile al seguente:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

L'esecuzione del programma put Reference Message effettua le seguenti operazioni:

- Il messaggio di riferimento viene inserito nella coda QR sul gestore code QMGR1.
- Il file di origine e il percorso sono `d:\files\infile.dat` ed esistono sul sistema in cui viene immesso il comando di esempio.
- Se la coda QR è una coda remota, il messaggio di riferimento viene inviato a un altro gestore code, su un sistema differente, dove viene creato un file con il nome e il percorso `e:\files\outfile.dat`. Il contenuto di questo file è lo stesso del file di origine.
- amqsprm attende per 30 secondi un report COA dal gestore code di destinazione.
- Il tipo di oggetto è `flatfile`, quindi il canale utilizzato per spostare i messaggi dalla coda QR deve specificarlo nel campo *MsgData* .

4. Quando si definiscono i canali, selezionare l'uscita del messaggio sia all'estremità di invio che a quella di ricezione per essere amqsxrm.

Windows Ciò è definito in Windows come segue:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX Ciò è definito in AIX and Linux come segue:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Se si specifica un nome percorso, specificare il nome completo. Se si omette il nome del percorso, si presume che il programma si trovi nel percorso specificato nel file `qm.ini` (o, su IBM MQ for Windows, nel percorso specificato nel registro).

5. L'uscita del canale legge l'intestazione del messaggio di riferimento e trova il file a cui fa riferimento.
6. L'uscita del canale può quindi segmentare il file prima di inviarlo lungo il canale insieme all'intestazione.

   Linux AIX Su AIX and Linux, modificare il proprietario del gruppo della directory di destinazione in 'mqm' in modo che l'uscita del messaggio di esempio possa creare il file in tale directory. Inoltre, modificare le autorizzazioni della directory di destinazione per consentire ai membri del gruppo mqm di scrivervi. I dati del file non vengono memorizzati sulle code IBM MQ .
7. Quando l'ultimo segmento del file viene elaborato dall'uscita del messaggio ricevente, il messaggio di riferimento viene inserito nella coda di destinazione specificata da amqsprm. Se questa coda viene attivata (ovvero, la definizione specifica gli attributi della coda **Trigger**, **InitQ**e **Process**), il programma specificato dal parametro PROC della coda di destinazione viene attivato. Il programma da attivare deve essere definito nel campo ApplId dell'attributo **Process** .
8. Quando il messaggio di riferimento raggiunge la coda di destinazione (DQ), viene inviato un report COA all'applicazione di inserimento (amqsprm).
9. L'esempio Richiama messaggio di riferimento, amqsgrm, richiama i messaggi dalla coda specificata nel messaggio del trigger di input e verifica l'esistenza del file.

Progettazione dell'esempio Inserisci messaggio di riferimento (amqsprma.c, AMQSPRM4)
Questo argomento fornisce una descrizione dettagliata di un esempio di messaggio Put Reference.

Questo esempio crea un messaggio di riferimento che fa riferimento a un file e lo inserisce in una coda specificata:

1. L'esempio si connette a un gestore code locale utilizzando MQCONN.
2. Apre quindi (MQOPEN) una coda modello utilizzata per ricevere i messaggi di report.
3. L'esempio crea un messaggio di riferimento contenente i valori richiesti per spostare il file, ad esempio, i nomi dei file di origine e di destinazione e il tipo di oggetto. Come esempio, l'esempio fornito con IBM MQ crea un messaggio di riferimento per l'invio del file d:\x\file.in da QMGR1 a QMGR2 e per ricreare il file come d:\y\file.out utilizzando i seguenti parametri:

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   Dove QR è una definizione di coda remota che fa riferimento a una coda di destinazione su QMGR2.

   Nota: Per le piattaforme AIX and Linux , utilizzare due barre rovesciate (\\) invece di una per indicare la directory del file di destinazione. Pertanto, il comando **amqsprm** è simile al seguente:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```
4. Il messaggio di riferimento viene inserito (senza dati di file) nella coda specificata dal parametro /q. Se si tratta di una coda remota, il messaggio viene inserito nella coda di trasmissione corrispondente.
5. L'esempio attende, per il periodo di tempo specificato nel parametro /w (che per impostazione predefinita è di 15 secondi), i report COA, che, insieme ai report di eccezioni, vengono inviati alla coda dinamica creata sul gestore code locale (QMGR1).

Progettazione dell'esempio Uscita messaggio di riferimento (amqsxrma.c, AMQSXRM4)
Questo esempio riconosce i messaggi di riferimento con un tipo di oggetto che corrisponde al tipo di oggetto nel campo dati utente dell'uscita messaggi della definizione del canale.

Per questi messaggi, si verifica quanto segue:

- Sul canale mittente o server, la lunghezza specificata dei dati viene copiata dall'offset specificato del file specificato nello spazio rimanente nel buffer dell'agente dopo il messaggio di riferimento. Se la fine del file non viene raggiunta, il messaggio di riferimento viene reinserito nella coda di trasmissione dopo l'aggiornamento del campo *DataLogicalOffset* .

- Sul canale richiedente o destinatario, se il campo *DataLogicalOffset* è zero e il file specificato non esiste, viene creato. I dati che seguono il messaggio di riferimento vengono aggiunti alla fine del file specificato. Se il messaggio di riferimento non è l'ultimo per il file specificato, viene eliminato. Altrimenti, viene restituito all'uscita del canale, senza i dati accodati, da inserire nella coda di destinazione.

Per i canali mittente e server, se il campo *DataLogicalLength* nel messaggio di riferimento di input è zero, la parte rimanente del file, da *DataLogicalOffset* alla fine del file, deve essere inviata lungo il canale. Se non è zero, viene inviata solo la lunghezza specificata.

Se si verifica un errore (ad esempio, se l'esempio non può aprire un file), MQCXP. *ExitResponse* è impostata su MQXCC_SUPPRESS_FUNCTION in modo che il messaggio in fase di elaborazione venga inserito nella coda di messaggi non recapitabili invece di continuare con la coda di destinazione. Un codice di feedback viene restituito in MQCXP. *Feedback* e restituito all'applicazione che inserisce il messaggio nel campo *Feedback* del descrittore del messaggio di un messaggio di report. Ciò è dovuto al fatto che l'eccezione richiesta dall'applicazione di inserimento viene segnalata impostando MQRO_EXCEPTION nel campo *Report* di MQMD.

Se la codifica o il CCSID (*CodedCharacterSetId*) del messaggio di riferimento è diverso da quello del gestore code, il messaggio di riferimento viene convertito nella codifica locale e nel CCSID. Nel nostro esempio, amqsprm, il formato dell'oggetto è MQFMT_STRING, per cui amqsxrm converte i dati dell'oggetto nel CCSID locale all'estremità di ricezione prima che i dati vengano scritti nel file.

Non specificare il formato del file trasferito come MQFMT_STRING se il file contiene caratteri multibyte (ad esempio, DBCS o Unicode). Ciò è dovuto al fatto che un carattere multibyte potrebbe essere suddiviso quando il file è segmentato all'estremità di invio. Per trasferire e convertire tale file, specificare il formato diverso da MQFMT_STRING in modo che l'uscita del messaggio di riferimento non lo converta e converte il file all'estremità di ricezione quando il trasferimento è completo.

Compilazione dell'esempio di uscita del messaggio di riferimento
Per compilare l'esempio Uscita messaggio di riferimento, utilizzare il comando per la piattaforma su cui è installato IBM MQ .

MQ_INSTALLATION_PATH rappresenta la directory di livello superiore in cui è installato IBM MQ .

Per compilare amqsxrma, utilizzare i comandi riportati di seguito:

AttivoAIX

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

AttivoIBM i

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

Nota:

1. Per creare il modulo in modo che utilizzi il filesystem IFS, aggiungere l'opzione SYSIFCOPT(*IFSIO)
2. Per creare il programma da utilizzare con canali non sottoposti a thread, utilizzare il seguente comando: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)
3. Per creare il programma da utilizzare con i canali con thread, utilizzare il seguente comando: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)

AttivoLinux

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

AttivoWindows

Windows IBM MQ ora fornisce la libreria mqm con i pacchetti client e i pacchetti server, quindi il seguente esempio utilizza `mqm.lib` invece di `mqmvx.lib`:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Concetti correlati

“Scrittura di programmi di uscita canale” a pagina 966
È possibile utilizzare le seguenti informazioni per scrivere programmi di uscita canale.

Progettazione dell'esempio Richiama messaggio di riferimento (amqsgrma.c, AMQSGRM4)
Questo argomento illustra la progettazione dell'esempio Richiama messaggio di riferimento.

La logica del programma è la seguente:

1. L'esempio viene attivato ed estrae il nome della coda e del gestore code dal messaggio del trigger di input.
2. Si connette quindi al gestore code specificato utilizzando MQCONN e apre la coda specificata utilizzando MQOPEN.
3. L'esempio emette MQGET con un intervallo di attesa di 15 secondi in un loop per richiamare i messaggi dalla coda.
4. Se un messaggio è un messaggio di riferimento, l'esempio verifica l'esistenza del file che è stato trasferito.
5. Chiude la coda e si disconnette dal gestore code.

I programmi di esempio Richiesta

I programmi di esempio Richiesta dimostrano l'elaborazione client/server. Gli esempi sono i client che inseriscono i messaggi di richiesta in una coda del server di destinazione elaborata da un programma server. Essi attendono che il programma server inserisci un messaggio di risposta in una coda di risposta.

Gli esempi di richiesta inserano una serie di messaggi di richiesta sulla coda server di destinazione utilizzando la chiamata MQPUT. Questi messaggi specificano la coda locale, SYSTEM.SAMPLE.REPLY come coda di risposta, che può essere una coda locale o remota. I programmi attendono i messaggi di risposta, quindi li visualizzano. Le risposte vengono inviate solo se la coda del server di destinazione viene elaborata da un'applicazione server o se un'applicazione viene attivata a tale scopo (i programmi di esempio Inquire, Set ed Echo sono progettati per essere attivati). L'esempio C attende 1 minuto (l'esempio COBOL attende 5 minuti), per l'arrivo della prima risposta (per consentire l'attivazione di un'applicazione server) e 15 secondi per le risposte successive, ma entrambi gli esempi possono terminare senza ottenere alcuna risposta. Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi dei programmi di esempio Richiesta.

Esecuzione dei programmi di esempio Richiesta

Esecuzione degli esempi amqsreq0.c, amqsreq e amqsreqc

La versione C del programma prende tre parametri:

1. Il nome della coda del server di destinazione (necessario)
2. Il nome del gestore code (facoltativo)

3. La coda di risposta (facoltativo)

Ad esempio, immettere uno dei seguenti:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

dove `myqueue` è il nome della coda del server di destinazione, `qmanagername` è il nome del gestore code proprietario di `myqueuee` `replyqueue` è il nome della coda di risposta.

Se si omette il nome del gestore code, si presume che il gestore code predefinito sia proprietario della coda. Se si omette il nome della coda di risposta, viene fornita la coda di risposta predefinita.

Esecuzione dell'esempio amq0req0.cbl

La versione COBOL non ha alcun parametro. Si connette al gestore code predefinito e quando viene eseguito viene richiesto:

```
Please enter the name of the target server queue
```

Il programma prende il suo input da StdIn e aggiunge ogni linea alla coda del server di destinazione, prendendo ogni riga di testo come contenuto di un messaggio di richiesta. Il programma termina quando viene letta una riga nulla.

Esecuzione dell'esempio AMQSREQ4

Il programma C crea messaggi prendendo i dati da stdin (la tastiera) con un tempo vuoto che termina l'immissione. Il programma utilizza un massimo di tre parametri: il nome della coda di destinazione (obbligatorio), il nome del gestore code (facoltativo) e il nome della coda di risposta (facoltativo). Se non viene specificato alcun nome di gestore code, viene utilizzato il gestore code predefinito. Se non viene specificata alcuna coda di risposta, SYSTEM.SAMPLE.REPLY REPLY.

Di seguito viene riportato un esempio di come richiamare il programma di esempio C, specificando la coda di risposta, ma lasciando al gestore code il valore predefinito:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

Nota: Tenere presente che i nomi delle code sono sensibili al maiuscolo / minuscolo. Tutte le code create dal programma di creazione del file di esempio AMQSAMP4 hanno nomi creati in caratteri maiuscoli.

Esecuzione dell'esempio AMQ0REQ4

Il programma COBOL crea messaggi accettando i dati dalla tastiera. Per avviare il programma, richiamare il programma e specificare il nome della coda di destinazione come parametro. Il programma accetta l'immissione dalla tastiera in un buffer e crea un messaggio di richiesta per ogni riga di testo. Il programma si arresta quando si immette una riga vuota sulla tastiera.

Esecuzione dell'esempio Richiesta utilizzando il trigger

Se l'esempio viene utilizzato con il trigger e uno dei programmi di esempio Inquire, Set o Echo, la riga di input deve essere il nome della coda a cui si desidera che il programma attivato acceda.

ALW *Esecuzione dell'esempio Richiesta utilizzando il trigger su AIX, Linux, and Windows*

Su AIX, Linux, and Windows, avviare il programma di controllo dei trigger RUNMQTRM in una sessione, quindi avviare il programma amq in un'altra sessione.

Per eseguire gli esempi utilizzando il trigger:

1. Avviare il programma di controllo trigger RUNMQTRM in una sessione (la coda di iniziazione SYSTEM.SAMPLE.TRIGGER è disponibile per l'uso).
2. Avviare il programma amqsreq in un'altra sessione.
3. Assicurarsi di aver definito una coda del server di destinazione.

   Le code di esempio disponibili da utilizzare come coda del server di destinazione per l'esempio di richiesta per inserire i messaggi sono:

   - SYSTEM.SAMPLE.INQ - per il programma di esempio Inquire
   - SYSTEM.SAMPLE.SET - per il programma di esempio Set
   - SYSTEM.SAMPLE.ECHO - per il programma di esempio Echo

   Queste code hanno un tipo di trigger FIRST, quindi se ci sono già messaggi nelle code prima di eseguire l'esempio Richiesta, le applicazioni server non vengono attivate dai messaggi inviati.
4. Assicurarsi di aver definito una coda per il programma di esempio Inquire, Set o Echo da utilizzare.

Ciò significa che il controllo trigger è pronto quando l'esempio di richiesta invia un messaggio.

Nota: Le definizioni del processo di esempio create utilizzando RUNMQSC e il file amqscos0.tst attivano gli esempi C. Modificare le definizioni del processo in amqscos0.tst e utilizzare RUNMQSC con questo file aggiornato per utilizzare versioni COBOL.

Figura 132 a pagina 1118 dimostra come utilizzare insieme gli esempi Richiesta e Interroga.

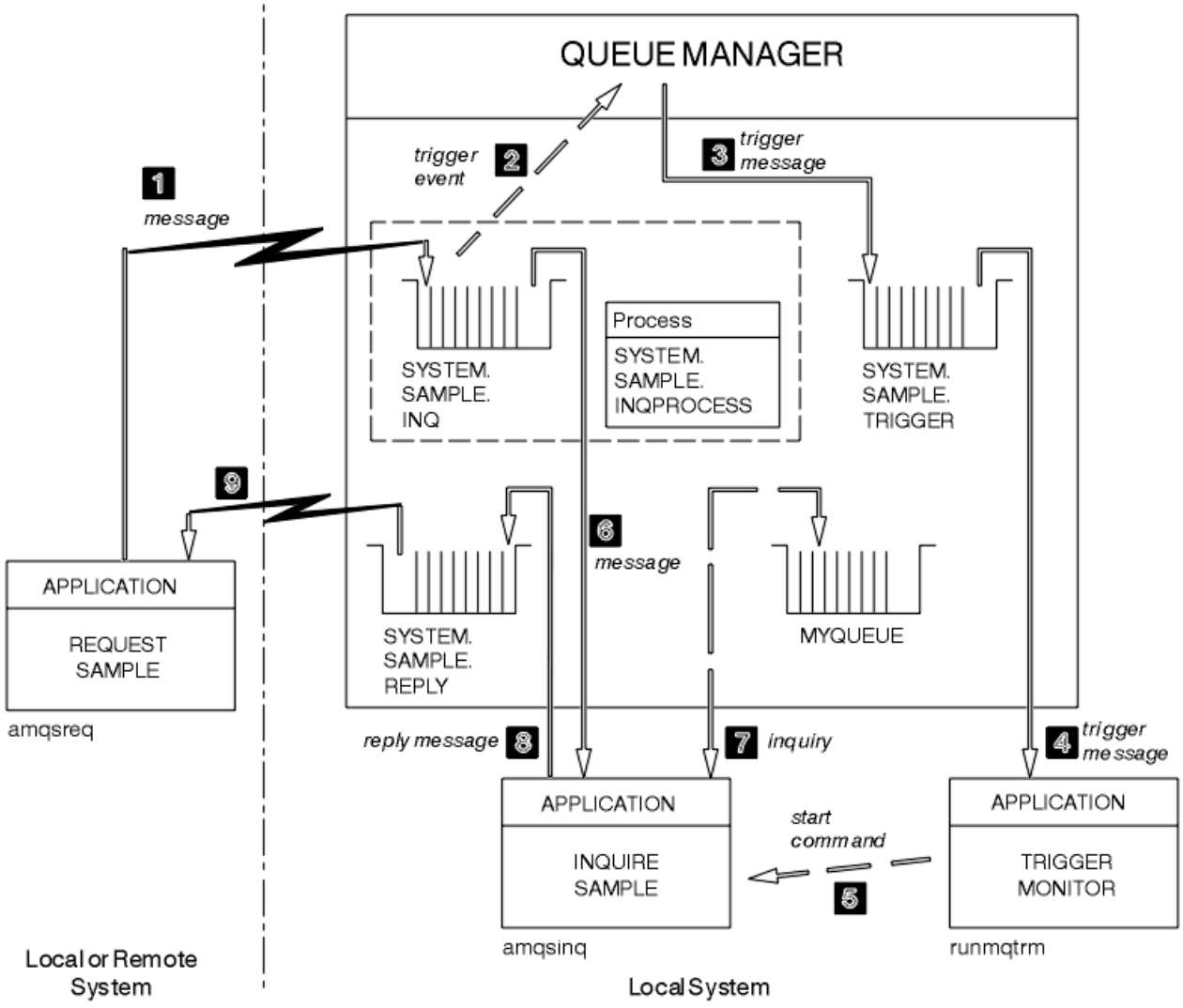


Figura 132. Richiedi e richiedi esempi utilizzando il trigger

In Figura 132 a pagina 1118 l'esempio Richiesta inserisce i messaggi nella coda del server di destinazione, SYSTEM.SAMPLE.INQe l'esempio Inquire interroga la coda MYQUEUE. In alternativa, è possibile utilizzare una delle code di esempio definite quando è stato eseguito amqscos0.tsto qualsiasi altra coda definita, per l'esempio Inquire.

Nota: I numeri in Figura 132 a pagina 1118 mostrano la sequenza di eventi.

Per eseguire gli esempi Richiesta e Interroga, utilizzando il trigger:

1. Verificare che le code che si desidera utilizzare siano definite. Eseguire amqscos0.tstper definire le code di esempio e definire una coda MYQUEUE.
2. Eseguire il comando di controllo trigger RUNMQTRM:

   ```
   RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
   ```

3. Esegui l'esempio di richiesta

   ```
   amqsreq SYSTEM.SAMPLE.INQ
   ```

   Nota: L'oggetto processo definisce cosa deve essere attivato. Se il client e il server non sono in esecuzione sulla stessa piattaforma, qualsiasi processo avviato dal controllo trigger deve definire *ApplType*, altrimenti il server utilizza le definizioni predefinite (ovvero, il tipo di applicazione normalmente associato alla macchina server) e causa un errore.

   Per un elenco di tipi di applicazione, vedere ApplType.
4. Immettere il nome della coda che si desidera venga utilizzata dall'esempio di interrogazione:

   ```
   MYQUEUE
   ```

5. Immettere una riga vuota (per terminare il programma Richiesta).
6. L'esempio di richiesta visualizzerà quindi un messaggio, contenente i dati del programma Inquire ottenuti da MYQUEUE.

È possibile utilizzare più di una coda; in questo caso, immettere i nomi delle altre code al passo “4” a pagina 1119.

Per ulteriori informazioni sull'attivazione, consultare “Avvio delle applicazioni IBM MQ utilizzando i trigger” a pagina 866.

IBM i *Esecuzione dell'esempio Richiesta utilizzando il trigger su IBM i*

Su IBM i, avviare il server trigger di esempio, AMQSERV4, in un lavoro, quindi avviare AMQSREQ4 in un altro. Ciò significa che il server trigger è pronto quando il programma di esempio Richiesta invia un messaggio.

Nota:

1. Le definizioni di esempio create da AMQSAMP4 attivano le versioni C degli esempi. Se si desidera attivare le versioni COBOL, modificare le definizioni di processo SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESSe SYSTEM.SAMPLE.SETPROCESS. È possibile utilizzare il comando CHGMQMPRC (per dettagli, consultare Modifica processo MQ (CHGMQMPRC)) per eseguire questa operazione o modificare ed eseguire la propria versione di AMQSAMP4.
2. Il codice sorgente per AMQSERV4 viene fornito solo per il linguaggio C. Tuttavia, una versione compilata (che è possibile utilizzare con gli esempi COBOL) viene fornita nella libreria QMQM.

È possibile inserire i messaggi di richiesta su queste code del server di esempio:

- SYSTEM.SAMPLE.ECHO (per i programmi di esempio Echo)
- SYSTEM.SAMPLE.INQ (per i programmi di esempio Inquire)
- SYSTEM.SAMPLE.SET (per i programmi di esempio Set)

Un grafico di flusso per SYSTEM.SAMPLE.ECHO è mostrato in Figura 133 a pagina 1121. Utilizzando il file di dati di esempio, il comando per emettere la richiesta di programma C a questo server è:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

Nota: Questa coda di esempio ha un tipo di trigger FIRST, quindi se ci sono già messaggi nella coda prima di eseguire l'esempio Richiesta, le applicazioni server non vengono attivate dai messaggi inviati.

Se si desidera provare ulteriori esempi, è possibile provare le seguenti variazioni:

- Utilizzare AMQSTRG4 (o l'equivalente della riga comandi STRMQMTRM, per i dettagli, consultare Start MQ Trigger Monitor (STRMQMTRM)) invece di AMQSERV4 per inoltrare il lavoro, ma i potenziali ritardi di inoltro del lavoro potrebbero rendere meno semplice seguire ciò che sta accadendo.
- Eseguire SYSTEM.SAMPLE.INQUIRE e SYSTEM.SAMPLE.SET . Utilizzando il file di dati di esempio, i comandi per emettere le richieste di programma C a questi server sono, rispettivamente:

  ```
  CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
  CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
  ```

  Queste code di esempio hanno anche un tipo di trigger FIRST.

Progettazione del programma di esempio Richiesta
Il programma apre la coda del server di destinazione in modo che possa inserire i messaggi. Utilizza la chiamata MQOPEN con l'opzione MQOO_OUTPUT. Se non è in grado di aprire la coda, il programma visualizza un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN.

Il programma apre quindi la coda di risposta denominata SYSTEM.SAMPLE.REPLY in modo che possa ricevere i messaggi di risposta. Per questo, il programma usa la chiamata MQOPEN con l'opzione MQOO_INPUT_EXCLUSIVE. Se non è in grado di aprire la coda, il programma visualizza un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN.

Per ogni riga di input, il programma legge il testo in un buffer e utilizza la chiamata MQPUT per creare un messaggio di richiesta contenente il testo di quella riga. In questa chiamata il programma utilizza l'opzione di report MQRO_EXCEPTION_WITH_DATA per richiedere che qualsiasi messaggio di report inviato sul messaggio di richiesta includa i primi 100 byte dei dati del messaggio. Il programma continua fino a quando non raggiunge la fine dell'input o la chiamata MQPUT ha esito negativo.

Il programma utilizza quindi la chiamata MQGET per rimuovere i messaggi di risposta dalla coda e visualizza i dati contenuti nelle risposte. La chiamata MQGET utilizza le opzioni MQGMO_WAIT, MQGMO_CONVERT e MQGMO_ACCEPT_TRUNCATED. Il *WaitInterval* è 5 minuti nella versione COBOL e 1 minuto nella versione C, per la prima risposta (per consentire l'attivazione di un'applicazione server) e 15 secondi per le risposte successive. Il programma attende questi periodi se non è presente alcun messaggio sulla coda. Se non arriva alcun messaggio prima della scadenza di questo intervallo, la chiamata ha esito negativo e restituisce il codice di errore MQRC_NO_MSG_AVAILABLE. La chiamata utilizza anche l'opzione MQGMO_ACCEPT_TRUNCATED_MSG, in modo che i messaggi più lunghi della dimensione del buffer dichiarata vengano troncati.

Il programma dimostra come cancellare i dati dei campi *MsgId* e *CorrelId* della struttura MQMD dopo ogni chiamata MQGET perché la chiamata imposta questi campi sui valori contenuti nel messaggio richiamato. La cancellazione di questi campi significa che le chiamate MQGET successive richiamano i messaggi nell'ordine in cui sono conservati nella coda.

Il programma continua fino a quando la chiamata MQGET non restituisce il codice motivo MQRC_NO_MSG_AVAILABLE o la chiamata MQGET non riesce. Se la chiamata ha esito negativo, il programma visualizza un messaggio di errore che contiene il codice di errore.

Il programma chiude quindi la coda del server di destinazione e la coda di risposta utilizzando la chiamata MQCLOSE.

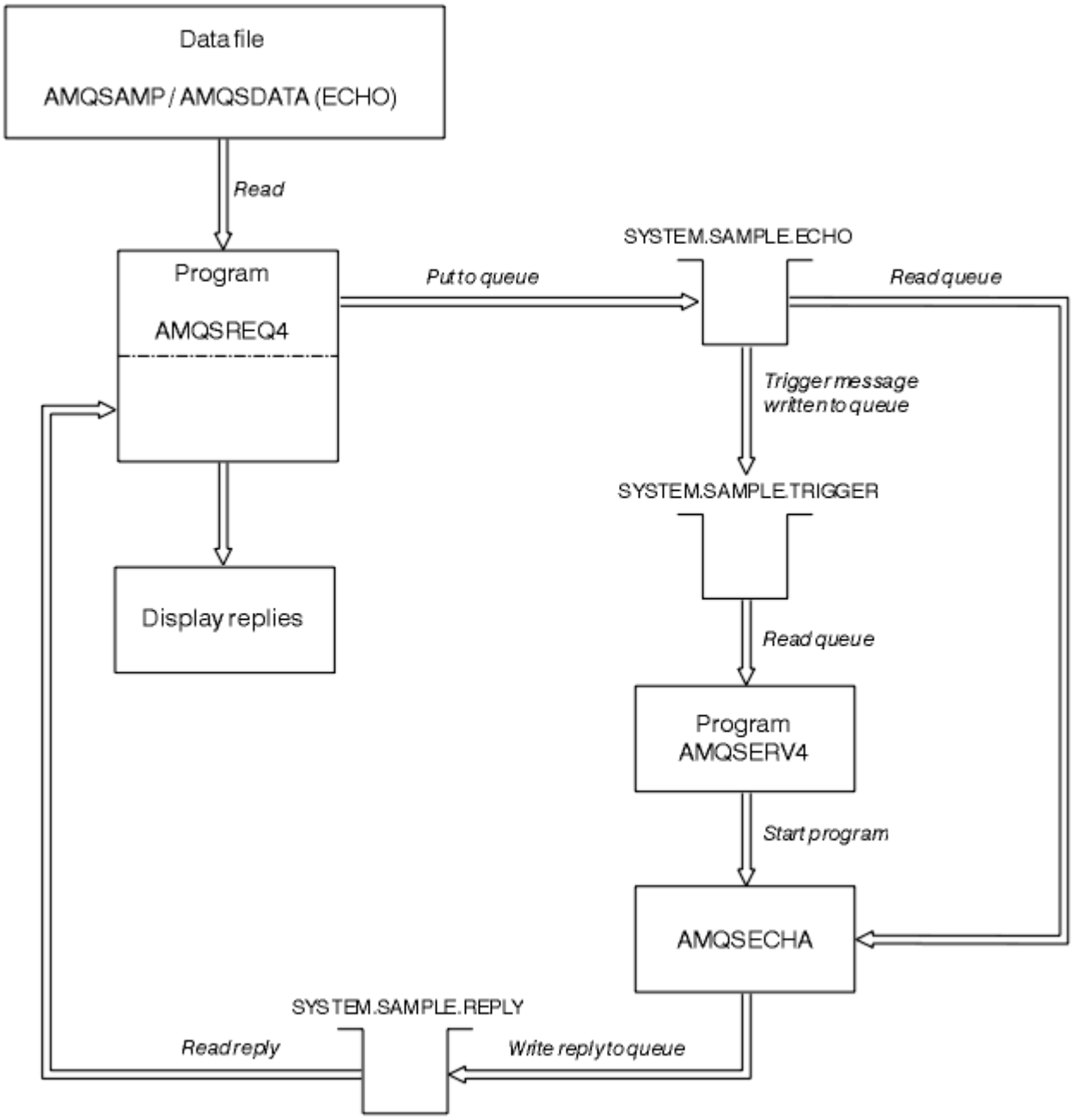


Figura 133. Diagramma di flusso del programma IBM i Client / Server (Echo) di esempio

I programmi di esempio Set

I programmi di esempio Set inibiscono le operazioni di inserimento su una coda utilizzando la chiamata MQSET per modificare l'attributo **InhibitPut** della coda. Inoltre, vengono fornite informazioni sulla progettazione di programmi di esempio Set.

Consultare "Funzioni dimostrate nei programmi di esempio su Multiplatforms" a pagina 1063 per i nomi di questi programmi.

I programmi sono concepiti per essere eseguiti come programmi attivati, quindi il loro unico input è una struttura MQTMC2 (messaggio trigger) che contiene il nome di una coda di destinazione con gli attributi che devono essere interrogati. La versione C utilizza anche il nome del gestore code. La versione COBOL utilizza il gestore code predefinito.

Affinché il processo di attivazione funzioni, assicurarsi che il programma di esempio Set che si desidera utilizzare sia attivato dai messaggi in arrivo sulla coda SYSTEM.SAMPLE.SET. A tale scopo, specificare il

nome del programma di esempio Set che si desidera utilizzare nel campo *ApplicId* della definizione del processo SYSTEM.SAMPLE.SETPROCESS. La coda di esempio ha un tipo di trigger FIRST; se ci sono già dei messaggi sulla coda prima di eseguire l'esempio Richiesta, l'esempio Imposta non viene attivato dai messaggi inviati.

Una volta impostata correttamente la definizione:

- ALW Per sistemi AIX, Linux, and Windows , avviare il programma **runmqtrm** in una sessione, quindi avviare il programma amqsreq in un altro.
- IBM i Per IBM i, avviare il programma AMQSERV4 in una sessione, quindi avviare il programma AMQSREQ4 in un altro. È possibile utilizzare AMQSTRG4 invece di AMQSERV4, ma i potenziali ritardi di inoltro dei lavori potrebbero rendere meno semplice seguire ciò che sta accadendo.

Utilizzare i programmi di esempio Richiesta per inviare messaggi di richiesta, ciascuno contenente solo un nome coda, alla coda SYSTEM.SAMPLE.SET. Per ogni messaggio di richiesta, i programmi di esempio Set inviano un messaggio di risposta contenente una conferma che le operazioni di inserimento sono state inibite sulla coda specificata. Le risposte vengono inviate alla coda di risposta specificata nel messaggio di richiesta.

Progettazione del programma di esempio Set

Il programma apre la coda denominata nella struttura del messaggio trigger che è stata passata quando è stata avviata. (Per chiarezza, chiameremo questa *coda di richiesta*.) Il programma utilizza la chiamata MQOPEN per aprire questa coda per l'input condiviso.

Il programma utilizza la chiamata MQGET per rimuovere i messaggi da questa coda. Questa chiamata utilizza le opzioni MQGMO_ACCEPT_TRUNCATED_MSG e MQGMO_WAIT, con un intervallo di attesa di 5 secondi. Il programma verifica il descrittore di ogni messaggio per verificare se si tratta di un messaggio di richiesta; in caso contrario, il programma elimina il messaggio e visualizza un messaggio di avvertenza.

Per ogni messaggio di richiesta rimosso da una coda di richiesta, il programma legge il nome della coda (che chiameremo *coda di destinazione*) contenuto nei dati e apre la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_SET. Quindi, il programma utilizza la chiamata MQSET per impostare il valore dell'attributo **InhibitPut** della coda di destinazione su MQQA_PUT_INIITED.

Se la chiamata MQSET ha esito positivo, il programma utilizza la chiamata MQPUT1 per inserire un messaggio di risposta nella coda di risposta. Questo messaggio contiene la stringa `PUT inhibited`.

Se la chiamata MQOPEN o MQSET ha esito negativo, il programma utilizza la chiamata MQPUT1 per inserire un messaggio `report` nella coda di risposta. Nel campo *Feedback* del descrittore del messaggio di questo messaggio di report è presente il codice motivo restituito dalla chiamata MQOPEN o MQSET, a seconda di quale ha avuto esito negativo.

Dopo la chiamata MQSET, il programma chiude la coda di destinazione utilizzando la chiamata MQCLOSE.

Quando non ci sono messaggi rimanenti sulla coda di richiesta, il programma chiude tale coda e si disconnette dal gestore code.

Il programma di esempio TLS

AMQSSSLC è un programma C di esempio che illustra come utilizzare le strutture MQCNO e MQSCO per fornire informazioni di connessione client TLS sulla chiamata MQCONNX. Ciò consente a un'applicazione MQI client di fornire la definizione del suo canale di connessione client e le impostazioni TLS al runtime senza una tabella di definizione del canale client (CCDT).

Se viene fornito un nome connessione, il programma crea una definizione di canale di connessione client in una struttura MQCD.

Se viene fornito il nome della radice del file del repository delle chiavi, il programma crea una struttura MQSCO; se viene fornito anche un URL del responder OCSP, il programma crea una struttura MQAIR del record delle informazioni di autenticazione.

Il programma si connette al gestore code utilizzando MQCONNX. Esso interroga e stampa il nome del gestore code a cui è connesso.

Questo programma deve essere collegato come applicazione client MQI. Tuttavia, può essere collegato come applicazione MQI regolare, nel qual caso si connette semplicemente a un gestore code locale e ignora le informazioni di connessione client.

V 9.3.0 V 9.3.0 Se la passphrase per accedere al repository delle chiavi non è nascosta in un file, è necessario fornire la passphrase a **amqsssl c** quando l'applicazione viene eseguita. È possibile fornire la passphrase mediante:

- Richiesta a **amqsssl c** di richiedere la passphrase, oppure
- Utilizzando la variabile di ambiente *MQKEYRPWD* oppure
- Utilizzo dell'attributo **SSLKeyRepositoryPassword** nel file di configurazione client

Per ulteriori informazioni sulla fornitura della password del repository delle chiavi alle applicazioni IBM MQ MQI client , consultare Supplemento della password del repository delle chiavi per un IBM MQ MQI client su AIX, Linux, and Windows.

amqsssl c accetta i seguenti parametri, tutti facoltativi:

-m QmgrName
Nome del gestore code a cui connettersi

-c ChannelName
Nome del canale da utilizzare

-x ConnName
Nome connessione server

Parametri TLS:

V 9.3.0 V 9.3.0 **-k KeyReposFileName**
Il nome del file di repository chiavi. Se l'estensione file non viene fornita, si presuppone che sia .kdb. Ad esempio:

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
La stringa CipherSpec del canale TLS corrispondente a **SSLCIPH** nella definizione del canale SVRCONN sul gestore code.

-f
Specifica che devono essere utilizzati solo algoritmi certificati FIPS 140-2.

-b VALUE1[,VALUE2...]
Specifica che devono essere utilizzati solo algoritmi compatibili con Suite B. Questo parametro è un elenco separato da virgole di uno o più dei seguenti valori: NONE,128_BIT,192_BIT. Questi valori hanno lo stesso significato di quelli per la variabile di ambiente **MQSUITEB** e l'impostazione **EncryptionPolicySuiteB** equivalente nella stanza SSL del file di configurazione client.

-p Politica
Specifica la politica di convalida del certificato da utilizzare. Può essere uno dei seguenti valori:

ANY
Applicare ciascuna delle politiche di convalida del certificato supportate dalla libreria dei socket protetti e accettare la catena di certificati se una delle politiche considera valida la catena di certificati. Questa impostazione può essere utilizzata per la massima retrocompatibilità con i vecchi certificati digitali che non sono conformi ai moderni standard di certificazione.

RFC5280
Applicare solo la politica di convalida del certificato conforme RFC 5280. Questa impostazione fornisce una convalida più rigorosa rispetto all'impostazione ANY, ma rifiuta alcuni certificati digitali meno recenti.

Il valore predefinito è qualsiasi.

-l CertLabel
L'etichetta del certificato da utilizzare per la connessione protetta.

Nota: È necessario specificare il valore utilizzando caratteri minuscoli.

V 9.3.0 V 9.3.0 **-w**
Specifica che **amqssslc** richiede la passphrase del repository delle chiavi da fornire.

V 9.3.0 V 9.3.0 **-i**
Specifica che **amqssslc** richiede la chiave iniziale utilizzata per codificare la passphrase del repository delle chiavi da fornire.

Specificare questa opzione se è stato specificato un file di chiavi iniziale quando la passphrase del repository delle chiavi è stata codificata utilizzando il programma di utilità **runmqicred** .

Parametro di revoca certificato OCSP:

-o URL
URL del responder OCSP

È anche possibile impostare una delle seguenti variabili di ambiente per fornire credenziali utilizzate per l'autenticazione con il gestore code:

ID UTENTE MQSAMP
Impostare l'ID utente da utilizzare per l'autenticazione della connessione, se si desidera utilizzare un ID utente e una password per l'autenticazione con il gestore code. Il programma richiede la password per accompagnare l'ID utente.

V 9.3.4 Linux AIX **MQSAMP_TOKEN**
Impostare su un valore non vuoto se si desidera fornire un token di autenticazione per l'autenticazione con il gestore code. Il programma richiede il token di autenticazione.

Esecuzione del programma di esempio TLS
Per eseguire il programma di esempio TLS, è necessario prima configurare l'ambiente TLS. Si esegue quindi l'esempio dalla riga comandi, fornendo un numero di parametri.

Informazioni su questa attività

Le seguenti istruzioni eseguono il programma di esempio utilizzando i certificati personali. Modificando il comando è possibile, ad esempio, utilizzare i certificati CA e verificarne lo stato utilizzando un responder OCSP. Consultare le istruzioni contenute nell'esempio.

Procedura

1. Creare un gestore code denominato QM1. Per ulteriori informazioni, consultare crtmqm.
2. Creare un repository delle chiavi per il gestore code. Per ulteriori informazioni, vedi Impostazione di un repository delle chiavi su AIX, Linux, and Windows.
3. Creare un archivio chiavi per il client. Chiamalo *clientkey.kdb*.

   Eseguire lo stash della password del repository chiavi in un file quando si crea il repository chiavi.
4. Creare un certificato personale per il gestore code. Per ulteriori informazioni, consultare Creazione di un certificato personale autofirmato su AIX, Linux, and Windows.
5. Creare un certificato personale per il client.
6. Estrarre il certificato personale dal repository delle chiavi del server e aggiungerlo al repository del client. Per ulteriori informazioni, consultare Estrazione della parte pubblica di un certificato autofirmato da un repository delle chiavi su AIX, Linux, and Windowse Aggiunta di un certificato CA (o

della parte pubblica di un'autocertificazione) in un repository delle chiavi, su sistemi AIX, Linux, and Windows.

7. Estrarre il certificato personale dal repository delle chiavi del client e aggiungerlo al repository delle chiavi del server.
8. Creare un canale di connessione server utilizzando il comando MQSC:

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

   Per ulteriori informazioni, consultare Canale di connessione server
9. Definire e avviare un listener del canale sul gestore code. Per ulteriori informazioni, vedere DEFINE LISTENER e START LISTENER.
10. Eseguire il programma di esempio utilizzando il comando seguente:

```
V 9.3.0 V 9.3.0
AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
-k "C:\Program Files\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
-o http://dummy.OCSP.responder
```

Risultati

Il programma di esempio esegue le seguenti operazioni:

1. Si connette a qualsiasi gestore code specificato o al gestore code predefinito, utilizzando le opzioni specificate.
2. Apre il gestore code e ne interroga il nome.
3. Chiude il gestore code.
4. Si disconnette dal gestore code.

Se il programma di esempio viene eseguito correttamente, viene visualizzato un output simile al seguente esempio:

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Program Files\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

Se il programma di esempio rileva un problema, visualizza un messaggio di errore appropriato, ad esempio se si specifica un URLdel responder OCSP non valido, si riceve il seguente messaggio:

```
MQCONNX ended with reason code 2553
```

Per un elenco di codici motivo, consultare Codici di completamento API e codici motivo.

I programmi di esempio Trigger

La funzione fornita nell'esempio di trigger è una serie secondaria di quella fornita nel controllo trigger nel programma **runmqtrm** .

Consultare “Funzioni dimostrate nei programmi di esempio su Multiplatforms” a pagina 1063 per i nomi di questi programmi.

Progettazione del campione di attivazione

Il programma di esempio di trigger apre la coda di avvio utilizzando la chiamata MQOPEN con l'opzione MQOO_INPUT_AS_Q_DEF. Richiama i messaggi dalla coda di iniziazione utilizzando la chiamata MQGET con le opzioni MQGMO_ACCEPT_TRUNCATED_MSG e MQGMO_WAIT, specificando un intervallo di attesa illimitato. Il programma elimina i campi *MsgId* e *CorrelId* prima di ogni chiamata MQGET per ottenere i messaggi in sequenza.

Una volta richiamato un messaggio dalla coda di iniziazione, il programma verifica il messaggio controllando la dimensione del messaggio per assicurarsi che sia della stessa dimensione di una struttura MQTM. Se questo test ha esito negativo, il programma visualizza un'avvertenza.

Per messaggi trigger validi, l'esempio di trigger copia i dati da questi campi: *ApplicId*, *EnvrData*, *Versione* *ApplType*. Gli ultimi due di questi campi sono numerici, quindi il programma crea sostituzioni di caratteri da utilizzare in una struttura MQTMC2 per sistemi IBM i, AIX, Linux, and Windows .

L'esempio di trigger emette un comando di avvio per l'applicazione specificata nel campo *ApplicId* del messaggio di trigger e passa una struttura MQTMC2 o MQTMC (una versione di caratteri del messaggio di trigger).

- ALW Nei sistemi AIX, Linux, and Windows , il campo *EnvData* viene utilizzato come estensione della stringa di comandi di richiamo.
- IBM i In IBM i, viene utilizzato come parametri di inoltro del lavoro, ad esempio, la priorità del lavoro o la descrizione del lavoro.

Infine, il programma chiude la coda di iniziazione.

Chiusura dei programmi di esempio di attivazione su IBM i

IBM i

Un programma di controllo trigger può essere terminato dall'opzione 2 di sysrequest (ENDRQS) o inibendo le ricezioni dalla coda trigger.

Se viene utilizzata la coda di trigger di esempio, il comando è:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

Importante: Prima di avviare di nuovo l'attivazione su questa coda, è necessario immettere il comando seguente:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Esecuzione dei programmi di esempio Trigger
Questo argomento contiene informazioni sull'esecuzione dei programmi di esempio Triggering.

Esecuzione degli esempi amqstrg0.c, amqstrg e amqstrgc

Il programma accetta 2 parametri:

1. Il nome della coda di iniziazione (necessario)
2. Il nome del gestore code (facoltativo)

Se un gestore code non è specificato, si connette a quello predefinito. Una coda di iniziazione di esempio sarà stata definita quando è stato eseguito amqscos0.tst; il nome di tale coda è SYSTEM.SAMPLE.TRIGGERed è possibile utilizzarlo quando si esegue questo programma.

Nota: La funzione in questo campione è un sottoinsieme della funzione di attivazione completa fornita nel programma runmqtrm.

Esecuzione dell'esempio AMQSTRG4

IBM i

Questo è un controllo trigger per l'ambiente IBM i . Inoltra un lavoro IBM i per ciascuna applicazione da avviare. Ciò significa che esiste un'ulteriore elaborazione associata a ciascun messaggio trigger.

AMQSTRG4 (in QCSRC) prende due parametri: il nome della coda di iniziazione che deve servire e il nome del gestore code (facoltativo). AMQSAMP4 (in QCLSRC) definisce una coda di avvio di esempio, SYSTEM.SAMPLE.TRIGGER, che è possibile utilizzare quando si provano i programmi di esempio.

Utilizzando la coda di trigger di esempio, il comando da immettere è:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

In alternativa, è possibile utilizzare l'equivalente CL STRMQMTRM; per i dettagli, consultare Start MQ Trigger Monitor (STRMQMTRM).

Esecuzione dell'esempio AMQSERV4

IBM i

Questo è un server di trigger per l'ambiente IBM i . Per ogni messaggio di trigger, questo server esegue il comando di avvio nel proprio lavoro per avviare l'applicazione specificata. Il server trigger può richiamare le transazioni CICS .

AMQSERV4 utilizza due parametri: il nome della coda di iniziazione che deve servire e il nome del gestore code (facoltativo). AMQSAMP4 definisce una coda di avvio di esempio, SYSTEM.SAMPLE.TRIGGER, che è possibile utilizzare quando si provano i programmi di esempio.

Utilizzando la coda di trigger di esempio, il comando da emettere è:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Progettazione del server trigger
La progettazione del server trigger è simile a quella del controllo trigger, con alcune eccezioni

La progettazione del server trigger è simile a quella del controllo trigger, tranne che per il fatto che il server trigger:

- Consente applicazioni MQAT_CICS e MQAT_OS400 .
- IBM i Richiama le applicazioni IBM i nel proprio lavoro (o utilizza STRCICSUSR per avviare le applicazioni CICS) invece di inoltrare un lavoro IBM i .
- Per le applicazioni CICS , sostituisce *EnvData*, ad esempio, per specificare la regione CICS , dal messaggio trigger nel comando STRCICSUSR.
- Apre la coda di avvio per l'input condiviso, in modo che molti server trigger possano essere eseguiti contemporaneamente.

Nota: I programmi avviati da AMQSERV4 non devono utilizzare la chiamata MQDISC perché questo arresta il server trigger. Se i programmi avviati da AMQSERV4 utilizzano la chiamata MQCONN, ricevono il codice di errore MQRC_ALREADY_CONNECTED.

ALW *Utilizzo degli esempi di TUXEDO su AIX, Linux, and Windows*

Informazioni sui programmi di esempio Put e Get per TUXEDO e sulla creazione dell'ambiente server in TUXEDO.

Prima di iniziare

Prima di eseguire questi esempi, è necessario creare l'ambiente server.

Informazioni su questa attività

Nota: In questa sezione, il carattere barra rovesciata (\) viene utilizzato per suddividere i comandi lunghi su più di una riga. Non immettere questo carattere. Immettere ciascun comando come riga singola.

ALW *Creazione di un ambiente server*
Informazioni sulla creazione dell'ambiente server per IBM MQ per diverse piattaforme.

Prima di iniziare

Si presume che l'utente disponga di un ambiente TUXEDO funzionante.

AIX *Creazione di un ambiente server per AIX (32 bit)*
Come creare l'ambiente del server per IBM MQ for AIX (32 bit).

Procedura

1. Creare una directory (ad esempio, APPDIR) in cui viene creato l'ambiente del server ed eseguire tutti i comandi in questa directory.
2. Esportare le seguenti variabili di ambiente, dove TUXDIR è la directory root per TUXEDO e *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui è installato IBM MQ :

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. Aggiungere la seguente riga al file TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. Eseguire i comandi:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. Modificare ubbstxcx.cfg e aggiungere i dettagli del nome della macchina, delle directory di lavoro e del gestore code come necessario:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Creare il TLOGDEVICE:

```
$tmadmin -c
```

   Viene visualizzata una richiesta. A questo prompt, immettere:

```
> crdl -z /APPDIR/TLOG1
```

7. Avviare il gestore code:

```
$ strmqm
```

8. Avviare Tuxedo:

```
$ tmboot -y
```

Operazioni successive

È ora possibile utilizzare i programmi doputs e dogets per inserire i messaggi in una coda e richiamarli da una coda.

AIX *Creazione dell'ambiente del server per AIX (64 bit)*
Come creare l'ambiente del server per IBM MQ for AIX (64 bit).

Procedura

1. Creare una directory (ad esempio, APPDIR) in cui viene creato l'ambiente del server ed eseguire tutti i comandi in questa directory.
2. Esportare le seguenti variabili di ambiente, dove TUXDIR rappresenta la directory root per TUXEDO e *MQ_INSTALLATION_PATH* rappresenta la directory di alto livello in cui IBM MQ è installed.:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Aggiungere la seguente riga al file TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. Eseguire i comandi:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. Modificare ubbstxcx.cfg e aggiungere i dettagli del nome della macchina, delle directory di lavoro e del gestore code come necessario:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Creare il TLOGDEVICE:

```
$tmadmin -c
```

Viene visualizzata una richiesta. A questo prompt, immettere:

```
> crdl -z /APPDIR/TLOG1
```

7. Avviare il gestore code:

```
$ strmqm
```

8. Avviare Tuxedo:

```
$ tmboot -y
```

Operazioni successive

È ora possibile utilizzare i programmi doputs e dogets per inserire i messaggi in una coda e richiamarli da una coda.

Windows *Creazione di un ambiente server per Windows (32 bit)*
Creazione dell'ambiente server per IBM MQ for Windows (32 bit).

Informazioni su questa attività

Nota: Modificare i campi identificati come *VARIABLES* nei seguenti percorsi di directory:

Tabella 164. Campi da modificare in percorsi di directory

Campo	**Percorso directory**
MQMDIR	Il percorso di directory specificato quando è installato IBM MQ , ad esempio `g:\Program Files\IBM\MQ`.
TUXDIR	Il percorso di directory specificato quando è stato installato TUXEDO, ad esempio `f:\tuxedo`.
DIRAPP	Il percorso di directory da utilizzare per l'applicazione di esempio, ad esempio `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figura 134. Esempio di file ubbstxcn.cfg per IBM MQ for Windows

Nota: Modificare il nome macchina *MachineName*e i percorsi delle directory in modo che corrispondano all'installazione. Inoltre, modificare il nome del gestore code *MYQUEUEMANAGER* con il nome del gestore code a cui si desidera connettersi.

Il file `ubbconfig` di esempio per IBM MQ for Windows è elencato in Figura 134 a pagina 1131. Viene fornito come `ubbstxcn.cfg` nella directory degli esempi IBM MQ .

Il makefile di esempio (vedere Figura 135 a pagina 1132) fornito per IBM MQ for Windows è denominato `ubbstxmn.make` si trova nella directory degli esempi IBM MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figura 135. Makefile TUXEDO di esempio per IBM MQ for Windows

Per creare l'ambiente server e gli esempi, completare la seguente procedura.

Procedura

1. Creare una directory dell'applicazione in cui creare l'applicazione di esempio, ad esempio:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copiare i seguenti file di esempio dalla directory di esempio IBM MQ alla directory dell'applicazione:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Modificare ciascuno di tali file per impostare i nomi di directory e i percorsi di directory utilizzati sull'installazione.
4. Modificare `ubbstxcn.cfg` (consultare Figura 134 a pagina 1131) per aggiungere i dettagli del nome macchina e del gestore code a cui si desidera connettersi.
5. Aggiungere la seguente riga al file TUXEDO *TUXDIR*`udataobj\rm`:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   La nuova voce deve essere una riga nel file.
6. Impostare le seguenti variabili di ambiente:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Creare una periferica TLOG per TUXEDO.

   A tale scopo, richiamare `tmadmin -c` e immettere il seguente comando:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Impostare la directory corrente su *APPDIR*e richiamare il makefile di esempio `amqstxmn.mak` come makefile di progetto esterno. Ad esempio, con Microsoft Visual C++, immettere il seguente comando:

```
msvc amqstxmn.mak
```

   Selezionare **build** per creare tutti i programmi di esempio.

Creazione dell'ambiente del server per Windows (64 bit)
Come creare l'ambiente del server per IBM MQ for Windows (64 bit).

Informazioni su questa attività

Nota: Modificare i campi identificati come *VARIABLES* nei seguenti percorsi di directory:

Tabella 165. Campi da modificare in percorsi di directory

Campo	**Percorso directory**
MQMDIR	Il percorso di directory specificato quando è installato IBM MQ , ad esempio `g:\Program Files\IBM\MQ`.
TUXDIR	Il percorso di directory specificato quando è stato installato TUXEDO, ad esempio `f:\tuxedo`.
DIRAPP	Il percorso di directory da utilizzare per l'applicazione di esempio, ad esempio `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figura 136. Esempio di file ubbstxcn.cfg per IBM MQ for Windows

Nota: Modificare il nome macchina *MachineName*e i percorsi delle directory in modo che corrispondano all'installazione. Inoltre, modificare il nome del gestore code *MYQUEUEMANAGER* con il nome del gestore code a cui si desidera connettersi.

Il file ubbconfig di esempio per IBM MQ for Windows viene elencato in Figura 136 a pagina 1134. Viene fornito come `ubbstxcn.cfg` nella directory degli esempi IBM MQ .

Il makefile di esempio (consultare Figura 137 a pagina 1135) fornito per IBM MQ for Windows è denominato `ubbstxmn.make` si trova nella directory degli esempi IBM MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figura 137. Makefile TUXEDO di esempio per IBM MQ for Windows

Per creare l'ambiente server e gli esempi, completare la seguente procedura.

Procedura

1. Creare una directory dell'applicazione in cui creare l'applicazione di esempio, ad esempio:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copiare i seguenti file di esempio dalla directory di esempio IBM MQ alla directory dell'applicazione:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Modificare ciascuno di tali file per impostare i nomi di directory e i percorsi di directory utilizzati sull'installazione.
4. Modificare `ubbstxcn.cfg` (consultare Figura 136 a pagina 1134) per aggiungere i dettagli del nome macchina e del gestore code a cui si desidera connettersi.
5. Aggiungere la seguente riga nel file TUXEDO *TUXDIR*`udataobj\rm`

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   La nuova voce deve essere una riga nel file.
6. Impostare le seguenti variabili di ambiente:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Creare una periferica TLOG per TUXEDO. A tale scopo, richiamare `tmadmin -c` e immettere il comando:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Impostare la directory corrente su *APPDIR*e richiamare il makefile di esempio `amqstxmn.mak` come makefile di progetto esterno. Ad esempio, con Microsoft Visual C++, immettere il seguente comando:

```
msvc amqstxmn.mak
```

   Selezionare **build** per creare tutti i programmi di esempio.

ALW *Programma server di esempio per TUXEDO*

Il programma del server di esempio (amqstxsx) è progettato per essere eseguito con i programmi di esempio Put (amqstxpx.c) e Get (amqstxgx.c). Il programma server di esempio viene eseguito automaticamente quando TUXEDO viene avviato.

Nota: È necessario avviare il gestore code prima di avviare TUXEDO.

Il server di esempio fornisce due servizi TUXEDO, MPUT1 e MGET1:

- Il servizio MPUT1 è guidato dall'esempio PUT e utilizza MQPUT1 nel punto di sincronizzazione per inserire un messaggio in un'unità di lavoro controllata da TUXEDO. Prende i parametri QName e Message Text, che sono forniti dall'esempio PUT.
- Il servizio MGET1 si apre e chiude la coda ogni volta che riceve un messaggio. Acquisisce i parametri QName e Message Text, forniti dall'esempio GET.

Tutti i messaggi di errore, i codici di errore e i messaggi di stato vengono scritti nel file di log TUXEDO.

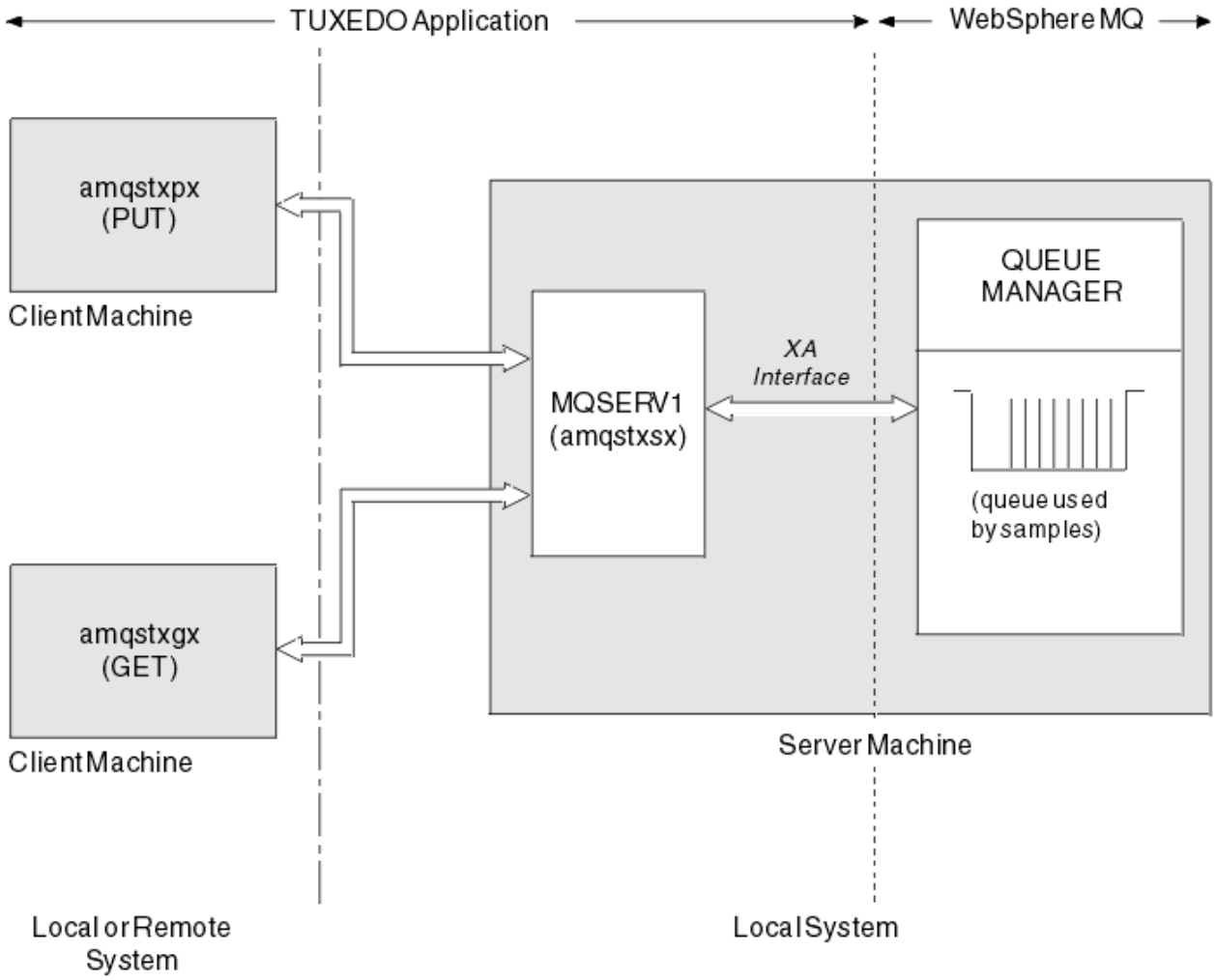


Figura 138. Funzionamento degli esempi TUXEDO

ALW *Programma di esempio di inserimento per TUXEDO*

Questo esempio consente di inserire un messaggio in una coda più volte, in batch, dimostrando il punto di sincronizzazione utilizzando TUXEDO come gestore risorse.

Il programma del server di esempio `amqstxsx` deve essere in esecuzione affinché l'esempio di inserimento abbia successo; il programma di esempio del server si connette al gestore code e utilizza l'interfaccia XA. Per eseguire l'esempio, immettere:

- `doputs -n queuename -b batchsize -c trancount -t message`

Ad esempio:

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

Ciò inserisce 30 messaggi nella coda denominata `myqueue`, in sei batch, ciascuno con cinque messaggi. In caso di problemi, esegue il backup di un batch di messaggi, altrimenti ne esegue il commit.

Tutti i messaggi di errore vengono scritti nel file di log TUXEDO e in stderr. Tutti i codici di errore vengono scritti in stderr.

ALW *Ottieni esempio per TUXEDO*

Questo esempio consente di richiamare i messaggi da una coda in batch.

Il programma del server di esempio `amqstxsx` deve essere in esecuzione affinché l'esempio Get abbia esito positivo; il programma del server di esempio si connette al gestore code e utilizza l'interfaccia XA. Per eseguire l'esempio, immettere il seguente comando:

- `dogets -n queuename -b batchsize -c trancount`

Ad esempio:

- `dogets -n myqueue -b 6 -c 4`

Ciò toglie 24 messaggi dalla coda denominata `myqueue`, in sei batch, ognuno con quattro messaggi. Se si esegue questa operazione dopo l'esempio di inserimento, che inserisce 30 messaggi su `myqueue`, si hanno solo sei messaggi su `myqueue`. Il numero di batch e la relativa dimensione possono variare tra l'inserimento e il richiamo dei messaggi.

Tutti i messaggi di errore vengono scritti nel file di log TUXEDO e in stderr. Tutti i codici di errore vengono scritti in stderr.

Windows

Utilizzo dell'uscita di sicurezza SSPI su Windows

Questo argomento descrive come utilizzare i programmi di uscita canale SSPI sui sistemi Windows . Il codice di uscita fornito è in due formati: oggetto e origine.

Codice oggetto

Il file del codice oggetto è denominato `amqrspin.dll`. Per client e server, viene installato come parte standard di IBM MQ for Windows nella cartella *MQ_INSTALLATION_PATH*`\exits\`*INSTALLATION_NAME* . Ad esempio, `C:\Program Files\IBM\MQ\exits\installation2`. Viene caricato come uscita utente standard. È possibile eseguire l'uscita del canale di sicurezza fornita e utilizzare i servizi di autenticazione nella definizione di canale.

A tale scopo, specificare una delle seguenti opzioni:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

Per fornire il supporto per un canale limitato, specificare quanto segue sul canale SVRCONN:

```
SCYDATA('remote_principal_name')
```

dove *nome_principale_remoto* è nel formato *DOMINIO**utente*. Il canale sicuro viene stabilito solo se il nome del principal remoto corrisponde a *nome_principale_remoto*.

Per utilizzare i programmi di uscita canale forniti tra i sistemi che operano all'interno di un dominio di sicurezza Kerberos , creare un **servicePrincipalName** per il gestore code.

Codice sorgente

Il file del codice sorgente di uscita è denominato `amqsspin.c`. Si trova in `C:\Program Files\IBM\MQ\Tools\c\Samples`.

Se si modifica il codice sorgente, è necessario ricompilare l'origine modificata.

È possibile compilarlo e collegarlo allo stesso modo di qualsiasi altra uscita del canale per la piattaforma pertinente, ad eccezione del fatto che è necessario accedere alle intestazioni SSPI al momento della compilazione e che le librerie di sicurezza SSPI, insieme alle librerie associate consigliate, devono essere accessibili al momento del collegamento.

Prima di eseguire questo comando, verificare che `cl.exe`, la libreria Visual C++ e la cartella `include` siano disponibili nel percorso. Ad esempio:

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

Nota: Il codice di origine non include alcuna disposizione per la traccia o la gestione degli errori. Se si modifica e si utilizza il codice sorgente, aggiungere le proprie routine di traccia e di gestione degli errori.

Esecuzione degli esempi utilizzando le code remote

È possibile dimostrare l'accodamento remoto eseguendo gli esempi sui gestori code connessi.

Il programma amqscos0.tst fornisce una definizione locale di una coda remota (SYSTEM.SAMPLE.REMOTE) che utilizza un gestore code remoto denominato OTHER. Per utilizzare questa definizione di esempio, modificare OTHER con il nome del secondo gestore code che si desidera utilizzare. Inoltre, è necessario impostare un canale di messaggi tra i due gestori code; per informazioni su come eseguire questa operazione, vedere Definizione dei canali.

I programmi di esempio di richiesta inseriscono il loro nome gestore code locale nel campo *ReplyToQMgr* dei messaggi che inviano. Gli esempi Inquire e Set inviano i messaggi di risposta alla coda e al gestore code dei messaggi denominati nei campi *ReplyToQ* e *ReplyToQMgr* dei messaggi di richiesta elaborati.

Programma di esempio Monitoraggio coda cluster (AMQSCLM)

Questo esempio utilizza le funzioni di bilanciamento del carico di lavoro del cluster IBM MQ integrate per indirizzare i messaggi alle istanze delle code a cui sono collegate le applicazioni che utilizzano. Questa direzione automatica impedisce la creazione di messaggi su un'istanza di una coda cluster a cui non è collegata alcuna applicazione di consumo.

Panoramica

È possibile impostare un cluster che abbia più di una definizione per la stessa coda su gestori code differenti. Questa configurazione fornisce il vantaggio di una maggiore disponibilità e bilanciamento del carico di lavoro. Tuttavia, non esiste alcuna funzionalità integrata in IBM MQ per modificare dinamicamente la distribuzione dei messaggi in un cluster in base allo stato delle applicazioni collegate. Per questo motivo, un'applicazione di consumo deve essere sempre collegata a ogni istanza di una coda per garantire che i messaggi vengano elaborati.

Il programma di esempio di monitoraggio della coda cluster monitora lo stato delle applicazioni collegate. Il programma regola in modo dinamico la configurazione di bilanciamento del carico di lavoro integrato per indirizzare i messaggi alle istanze di una coda cluster con applicazioni di consumo collegate. In alcune situazioni questo programma può essere utilizzato per allentare la necessità che un'applicazione di

consumo sia sempre connessa a ogni istanza di una coda. Reinvia anche i messaggi che vengono accodati su un'istanza di una coda senza applicazioni collegate. Il nuovo invio dei messaggi consente di instradare i messaggi verso un'applicazione che utilizza e che viene temporaneamente chiusa.

Il programma è progettato per essere utilizzato dove le applicazioni di consumo sono applicazioni di lunga durata, piuttosto che collegare e scollegare frequentemente le applicazioni.

Il programma di esempio di controllo della coda cluster è il programma eseguibile compilato del file di esempio C `amqsclma.c`.

Ulteriori informazioni sui cluster e sul carico di lavoro sono disponibili in Utilizzo dei cluster per la gestione del carico di lavoro

AMQSCLM: Progettazione e pianificazione dell'utilizzo dell'esempio
Le informazioni relative al funzionamento del programma di esempio di monitoraggio della coda cluster, puntano a considerare quando si imposta un sistema su cui eseguire il programma di esempio e le modifiche che possono essere apportate al codice sorgente di esempio.

Progetta

Il programma di esempio di monitoraggio della coda del cluster monitora le code cluster locali che dispongono di applicazioni collegate. Il programma controlla le code specificate dall'utente. Il nome della coda potrebbe essere specifico, ad esempio APP.TEST01o generico. I nomi generici devono avere un formato conforme a PCF (Programmable Command Format). Esempi di nomi generici sono APP.TEST*o APP*.

Ogni gestore code in un cluster che possiede un'istanza di una coda locale da monitorare, richiede la connessione di un'istanza del programma di esempio di monitoraggio della coda del cluster.

Instradamento messaggio dinamico

Il programma di esempio di monitoraggio della coda cluster utilizza il valore **IPPROCS** (aperto per il conteggio del processo di input) di una coda per determinare se tale coda ha dei consumer. Un valore maggiore di 0 indica che la coda ha almeno un'applicazione di consumo collegata. Tali code sono attive. Il valore 0 indica che la coda non ha programmi di consumo collegati. Tali code sono inattive.

Per una coda con cluster con più istanza in un cluster, IBM MQ utilizza la proprietà di priorità del carico di lavoro del cluster **CLWLPRTY** di ogni istanza della coda per determinare a quali istanze inviare messaggi. IBM MQ invia i messaggi alle istanze disponibili di una coda con il valore **CLWLPRTY** più elevato.

Il programma di esempio di controllo della coda cluster attiva una coda cluster impostando il valore **CLWLPRTY** locale su 1. Il programma disattiva una coda cluster impostando il valore **CLWLPRTY** su 0.

La tecnologia di cluster IBM MQ propaga la proprietà **CLWLPRTY** aggiornata di una coda cluster a tutti i gestori code pertinenti nel cluster. Ad esempio:

- Un gestore code con un'applicazione collegata che inserisce messaggi nella coda.
- Un gestore code che possiede una coda locale con lo stesso nome nello stesso cluster.

La propagazione viene eseguita utilizzando i gestori code del repository completo del cluster. I nuovi messaggi per la coda del cluster vengono indirizzati alle istanze con il valore **CLWLPRTY** più alto all'interno del cluster.

Trasferimento messaggio accodato

La modifica dinamica del valore di **CLWLPRTY** influenza l'instradamento dei nuovi messaggi. Questa modifica dinamica non influisce sui messaggi già accodati su un'istanza della coda senza utenti collegati o sui messaggi che hanno attraversato il meccanismo di bilanciamento del carico di lavoro prima che un valore **CLWLPRTY** modificato venisse propagato nel cluster. Di conseguenza, i messaggi rimangono su qualsiasi coda inattiva e non vengono elaborati da un'applicazione che li utilizza. Per risolvere questo problema, il programma di esempio di controllo della coda del cluster è in grado di richiamare i messaggi

da una coda locale senza consumer e di inviare tali messaggi alle istanze remote della stessa coda in cui sono collegati i consumer.

Il programma di esempio di monitoraggio della coda cluster trasferisce i messaggi da una coda locale inattiva a una o più code remote attive richiamando i messaggi (utilizzando **MQGET**) e l'inserimento di messaggi (utilizzando **MQPUT**) alla stessa coda cluster. Questo trasferimento fa sì che la gestione del carico di lavoro del cluster IBM MQ selezioni un'altra istanza di destinazione, basata su un valore CLWLPRTY superiore a quello dell'istanza della coda locale. La persistenza e il contesto del messaggio vengono conservati durante il trasferimento del messaggio. L'ordine dei messaggi e le opzioni di bind non vengono conservate.

Pianificazione

Il programma di esempio di monitoraggio della coda del cluster modifica la configurazione del cluster quando si verifica una modifica nella connettività delle applicazioni utilizzate. Le modifiche vengono trasmesse dai gestori code in cui il programma di esempio di monitoraggio della coda del cluster sta monitorando le code, ai gestori code del repository completo nel cluster. I gestori code del repository completo elaborano gli aggiornamenti della configurazione e li inviano nuovamente a tutti i gestori code pertinenti nel cluster. I gestori code rilevanti includono i gestori code che possiedono code cluster con lo stesso nome (dove è in esecuzione un'istanza del programma di esempio di monitoraggio della coda del cluster) e qualsiasi gestore code in cui un'applicazione ha aperto la coda del cluster per inserire i messaggi negli ultimi 30 giorni.

Le modifiche vengono elaborate asincrono nel cluster. Pertanto, dopo ogni modifica, diversi gestori code nel cluster potrebbero avere viste diverse della configurazione per un periodo di tempo.

Il programma di esempio di monitoraggio della coda cluster è adatto solo per i sistemi in cui le applicazioni che utilizzano raramente si collegano o si scollegano; ad esempio, le applicazioni che utilizzano da lungo tempo. Quando viene utilizzato per monitorare i sistemi in cui le applicazioni che utilizzano sono collegate solo per brevi periodi, la latenza che si verifica durante la distribuzione degli aggiornamenti della configurazione potrebbe far sì che i gestori code nel cluster abbiano una vista non corretta delle code in cui sono collegati i consumer. Questa latenza potrebbe causare messaggi instradati in modo non corretto.

Quando si esegue il monitoraggio di molte code, una frequenza relativamente bassa di modifica nei consumer collegati in tutte le code potrebbe aumentare il traffico di configurazione del cluster nel cluster. L'aumento del traffico di configurazione cluster può causare un carico eccessivo su uno o più dei seguenti gestori code.

- I gestori code in cui è in esecuzione il programma di esempio di monitoraggio della coda cluster
- I gestori code del repository completo
- Un gestore code con un'applicazione connessa che inserisce i messaggi nella coda
- Un gestore code che possiede una coda locale con lo stesso nome nello stesso cluster

È necessario valutare l'utilizzo del processore sui gestori code del repository completo. Un ulteriore utilizzo del processore è visibile come traffico di messaggi sulla coda del repository completo SYSTEM.CLUSTER.COMMAND.QUEUE. Se i messaggi si accumulano su tale coda, ciò indica che i gestori code del repository completo non sono in grado di tenere il passo con la frequenza di modifica della configurazione del cluster nel sistema.

Quando molte code vengono monitorate dal programma di esempio di monitoraggio della coda cluster, il programma di esempio e il gestore code eseguono una quantità di lavoro. Questo lavoro viene eseguito, anche quando non vi sono modifiche ai consumatori collegati. L'argomento **-i** può essere modificato per diminuire l'utilizzo del processore del programma di esempio sul sistema locale, diminuendo la frequenza del ciclo di monitoraggio.

Per facilitare il rilevamento di un'attività eccessiva, il programma di esempio di monitoraggio della coda del cluster riporta il tempo di elaborazione medio per l'intervallo di polling, il tempo di elaborazione trascorso e il numero di modifiche alla configurazione. I report vengono consegnati in un messaggio informativo, **CLM0045I**, ogni 30 minuti o ogni 600 intervalli di polling, a seconda di quale dei due si verifica prima.

Requisiti di utilizzo del monitoraggio della coda cluster

Il programma di esempio di monitoraggio della coda del cluster ha requisiti e limitazioni. È possibile modificare il codice sorgente di esempio fornito per modificare alcune di tali limitazioni in modo che possa essere utilizzato. Gli esempi elencati in questa sezione descrivono le modifiche che è possibile apportare.

- Il programma di esempio di monitoraggio della coda cluster è progettato per essere utilizzato per monitorare le code in cui le applicazioni che utilizzano sono collegate o non collegate. Se il sistema utilizza applicazioni che si collegano e scollegano frequentemente, il programma di esempio potrebbe generare un'eccessiva attività di configurazione del cluster nell'intero cluster. Ciò potrebbe avere un impatto sulle prestazioni dei gestori code nel cluster.
- Il programma di esempio di monitoraggio della coda cluster dipende dalla tecnologia cluster e dal sistema IBM MQ sottostante. Il numero di code monitorate, la frequenza di monitoraggio e la frequenza di modifica dello stato di ciascuna coda influiscono sul carico sul sistema globale. Questi fattori devono essere considerati quando si selezionano le code da monitorare e l'intervallo di polling del monitoraggio.
- Un'istanza del programma di esempio di monitoraggio della coda del cluster deve essere connessa a ogni gestore code nel cluster che possiede un'istanza di coda da monitorare. Non è necessario connettere il programma di esempio ai gestori code nel cluster che non possiedono le code.
- Il programma di esempio di monitoraggio della coda cluster deve essere eseguito con l'autorizzazione appropriata per accedere a tutte le risorse IBM MQ richieste. Ad esempio:
  - Il gestore code a cui connettersi
  - Il SISTEMA SYSTEM.ADMIN.COMMAND.QUEUE
  - Tutte le code da monitorare quando viene effettuato il trasferimento del messaggio
- Il server dei comandi deve essere in esecuzione per ogni gestore code con il programma di esempio di monitoraggio della coda cluster connesso.
- Ogni istanza del programma di esempio di monitoraggio della coda cluster richiede l'uso esclusivo di una coda locale (non cluster) sul gestore code a cui è connessa. Questa coda locale viene utilizzata per controllare il programma di esempio e ricevere messaggi di risposta da richieste effettuate al server dei comandi del gestore code.
- Tutte le code che devono essere monitorate da una singola istanza del programma di esempio di monitoraggio della coda cluster devono trovarsi nello stesso cluster. Se un gestore code ha code in più cluster che richiedono il controllo, sono richieste più istanze del programma di esempio. Per ogni istanza è necessaria una coda locale per i messaggi di controllo e di risposta.
- Tutte le code da monitorare devono trovarsi in un singolo cluster. Le code configurate per utilizzare un elenco nomi cluster non vengono monitorate.
- L'abilitazione del trasferimento dei messaggi dalle code inattive è facoltativa. Si applica a tutte le code controllate dall'istanza del programma di esempio di monitoraggio della coda cluster. Se solo un sottoinsieme delle code monitorate richiede il trasferimento del messaggio abilitato, sono necessarie due istanze del programma di esempio di monitoraggio della coda cluster. Un programma di esempio ha il trasferimento messaggi abilitato e l'altro ha il trasferimento messaggi disabilitato. Ciascuna istanza del programma di esempio necessita di una coda locale per i messaggi di controllo e di risposta.
- Il bilanciamento del carico di lavoro del cluster IBM MQ , per impostazione predefinita, invierà messaggi alle istanze delle code cluster che risiedono sullo stesso gestore code a cui è connessa un'applicazione di inserimento. Questo deve essere disabilitato mentre la coda locale è inattiva nelle seguenti circostanze:
  - Le applicazioni di inserimento si connettono ai gestori code che possiedono le istanze di una coda inattiva monitorata
  - I messaggi in coda vengono trasferiti dalle code inattive alle code attive.

  La preferenza di bilanciamento del carico di lavoro locale sulla coda può essere disabilitata staticamente, tramite l'impostazione del valore `CLWLUSEQ` su `ANY`. In questa configurazione i messaggi inseriti nelle code locali vengono distribuiti alle istanze della coda locale e remota per bilanciare il carico di lavoro, anche quando sono presenti applicazioni che utilizzano il sistema locale. In alternativa, è possibile configurare il programma di esempio di monitoraggio della coda del cluster per impostare

temporaneamente il valore **CLWLUSEQ** su ANY mentre la coda non ha consumer collegati, il che fa sì che solo i messaggi locali vengano inviati alle istanze locali di una coda mentre tale coda è attiva.

- Il sistema IBM MQ e le applicazioni non devono utilizzare **CLWLPRTY** per le code da monitorare o per i canali da utilizzare. Altrimenti, le azioni del programma di esempio di monitoraggio della coda cluster sugli attributi della coda **CLWLPRTY** potrebbero avere effetti indesiderati.
- Il programma di esempio di monitoraggio della coda del cluster registra le informazioni di runtime in una serie di file di report. È necessaria una directory per memorizzare questi report e il programma di esempio di monitoraggio della coda cluster deve disporre dell'autorizzazione per scrivervi.

AMQSCLM: Preparazione ed esecuzione dell'esempio
L'esempio di monitoraggio della coda cluster può essere eseguito localmente connesso a un gestore code o come client connesso su un canale. L'esempio deve essere in esecuzione ogni volta che il gestore code è in esecuzione, quando è in esecuzione localmente può essere configurato come servizio del gestore code per avviare e arrestare automaticamente l'esempio con il gestore code.

Prima di iniziare

I seguenti passi devono essere completati prima di eseguire l'esempio di monitoraggio della coda cluster.

1. Creare una coda di lavoro su ciascun gestore code per l'utilizzo interno dell'esempio.

   Ogni istanza dell'esempio richiede una coda non cluster locale per uso interno esclusivo. È possibile scegliere il nome della coda. L'esempio utilizza il nome AMQSCLM.CONTROL.QUEUE. Ad esempio, in Windows, è possibile creare questa coda utilizzando il seguente comando **MQSC** :

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   È possibile lasciare i valori di **MAXDEPTH** e **MAXMSGL** come predefiniti.

2. Creare una directory per i log dei messaggi di errore e informativi.

   L'esempio scrive i messaggi diagnostici nei file di report. È necessario scegliere una directory in cui memorizzare i file. Ad esempio, su Windows, è possibile creare una directory utilizzando il seguente comando:

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   I file di report creati dall'esempio hanno la seguente convenzione di denominazione:

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```

3. (Facoltativo) Definire l'esempio di monitoraggio della coda del cluster come servizio IBM MQ .

   Per monitorare le code, l'esempio deve essere sempre in esecuzione. Per accertarsi che l'esempio di controllo della coda del cluster sia sempre in esecuzione, è possibile definire l'esempio come un servizio gestore code. La definizione dell'esempio come servizio significa che AMQSCLM viene avviato all'avvio del gestore code. Puoi utilizzare il seguente esempio per definire l'esempio di monitoraggio della coda del cluster come servizio IBM MQ .

   ```
   define service(AMQSCLM) +
         descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
         control(qmgr) +
         servtype(server) +
         startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
         startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
   c:\AMQSCLM\rpts') +
         stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
         stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
   ```

Definizione	Descrizione
service	Specifica il nome del servizio. È possibile scegliere il nome servizio.
descr	Specifica una descrizione testuale del servizio.
control	Indica che il servizio viene avviato e arrestato contemporaneamente al gestore code.
servtype	Indica un oggetto servizio del server, che indica che è possibile eseguire una sola istanza alla volta per questo gestore code.
startcmd	Specifica l'ubicazione e nome del programma.
startarg	Specifica gli argomenti dell'esempio. Notare l'utilizzo di *+ QMNAME +*. Il nome del gestore code viene sostituito automaticamente.
stdout	Il nome file completo a cui viene reindirizzato l'output standard. L'esempio scrive in questo file solo i messaggi che confermano che l'esempio è terminato. L'esempio fa ciò perché il file di errore standard è già stato chiuso in una fase precedente del processo di terminazione dell'esempio.
stderr	Il nome file completo a cui viene reindirizzato l'output di errore standard. L'esempio scrive nel file di errore standard tutti i messaggi di errore prima della chiusura dell'esempio.

Informazioni su questa attività

Questa attività consente di avviare e arrestare l'esempio di monitoraggio della coda cluster in modi diversi. Inoltre, consente di eseguire l'esempio in una modalità che genera file di report contenenti informazioni statistiche sulle code monitorate.

Il programma di esempio può essere eseguito utilizzando il seguente comando.

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

La tabella elenca gli argomenti che è possibile utilizzare con l'esempio di controllo della coda cluster, insieme ad ulteriori informazioni su ciascuno di essi.

Argomento	Variabile	Ulteriori informazioni
-m	**QMgrName**	Il gestore code da controllare.
-c	**ClusterName**	Il cluster contenente le code da monitorare.
-q	**QNameMask**	La coda o le code da monitorare. Un * finale monitora tutte le code con nomi che corrispondono a zero o più caratteri finali.
-f	**QListFile**	Il percorso completo e il nome file di un file contenente un elenco di nomi coda o maschere nome coda da monitorare. Il file deve contenere un nome coda / maschera per riga. È possibile specificare **-q** o **-f**, ma non entrambi.
-r	**MonitorQName**	La coda locale utilizzata esclusivamente dall'esempio.
-l	**ReportDir**	Il percorso di directory in cui memorizzare i messaggi di informazioni registrati in una serie di wrapping[9]File di report.
-t		(Facoltativo) Abilita il trasferimento dei messaggi in coda dalle code locali inattive alle code attive. Se non abilitato, solo i nuovi messaggi che entrano nel cluster vengono instradati dinamicamente alle istanze attive di una coda.
-u	**ActiveVal**	(Facoltativo) Passa automaticamente la proprietà **CLWLUSEQ** di una istanza della coda monitorata a ANY quando è inattiva e al valore **ActiveVal** quando è attiva. **ActiveVal** può essere LOCAL o QMGR. Se questo argomento non è impostato in un sistema in cui le applicazioni di inserimento si connettono allo stesso gestore code o in cui è abilitato il trasferimento dei messaggi, le code monitorate devono avere un **CLWLUSEQ** valore ANYo QMGR con il gestore code con un valore ANY.
-i	**Interval**	(Facoltativo) L'intervallo di tempo, in secondi, con cui il monitoraggio controlla le code. Il valore predefinito è 300 secondi (5 minuti).
-d		(Facoltativo) Abilita ulteriori output di diagnostica. L'output di debug potrebbe essere utile quando si configura inizialmente il sistema o quando si utilizza il codice di esempio.
-s		(Facoltativo) Abilita l'output statistico minimo per intervallo.
-v		(Facoltativo) Registrare le informazioni del report in `standard out`, oltre ai file di report.

Esempi di elenco argomenti:

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

File di elenco code di esempio:

```
Q1
QUEUE.*
ABC
ABD
```

[9] Per ogni combinazione di gestore code e coda viene generato un file di log a dimensione fissa che, quando è pieno, viene sovrascritto. Il programma di registrazione scrive sempre nello stesso file e conserva anche le due precedenti versioni del file.

Procedura

1. Avviare l'esempio di monitoraggio coda cluster. È possibile avviare l'esempio in uno dei seguenti modi:
   - Utilizzare un prompt dei comandi con le autorizzazioni utente appropriate.
   - Utilizzare il comando MQSC **START SERVICE** , se l'esempio è configurato come un servizio IBM MQ .

   L'elenco degli argomenti è lo stesso in entrambi i casi.

   L'esempio non avvia il controllo delle code per 10 secondi dopo l'inizializzazione del programma. Questo ritardo consente alle applicazioni di connettersi prima alle code monitorate, evitando modifiche non necessarie allo stato attivo della coda.
2. Arrestare l'esempio di monitoraggio della coda cluster. L'esempio si arresta automaticamente quando il gestore code viene arrestato, arrestato, sospeso o se la connessione al gestore code è interrotta. Esistono diversi modi per arrestare l'esempio senza arrestare il gestore code:
   - Configurare la coda locale utilizzata esclusivamente dall'esempio per disabilitare la funzione Get.
   - Inviare un messaggio con un **CorrelId** di `"STOP CLUSTER MONITOR\0\0\0\0"`, alla coda locale utilizzata esclusivamente dall'esempio.
   - Terminare il processo di esempio. Ciò potrebbe causare la perdita di messaggi non persistenti trasferiti alle code attive. Potrebbe anche risultare nella coda locale utilizzata dall'esempio tenuta aperta per un numero di secondi dopo la terminazione. Questa situazione impedisce l'avvio immediato di una nuova istanza dell'esempio di monitoraggio della coda cluster.

   Se l'esempio è stato avviato come servizio IBM MQ , **STOP SERVICE** non ha alcun effetto. È possibile utilizzare uno dei metodi di terminazione descritti come meccanismo **STOP SERVICE** configurato nel gestore code.

Operazioni successive

Controllare lo stato dell'esempio.

Se la creazione report è abilitata, è possibile esaminare i file di report per lo stato. Utilizzare il seguente comando per esaminare il file di report più recente:

```
QMgrName.ClusterName.RPT01.LOG
```

Per esaminare i vecchi file di report, utilizzare i comandi riportati di seguito:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

I file di report raggiungono una dimensione massima di circa 1 MB. Quando il file RPT01 si riempie, viene creato un nuovo file RPT01 . Il vecchio file RPT01 viene ridenominato in RPT02. RPT02 viene ridenominato in RPT03. Il vecchio RPT03 viene eliminato.

L'esempio crea messaggi informativi nelle situazioni seguenti:

- all'avvio
- al termine
- quando contrassegna una coda **ACTIVE** o **INACTIVE**
- quando riaccoda i messaggi da una coda inattiva a un'istanza o a istanze attive

L'esempio crea un messaggio di errore *CLMnnnnE* per segnalare un problema che richiede attenzione.

Ogni 30 minuti, il campione riporta il tempo di elaborazione medio per intervallo di polling e il tempo di elaborazione trascorso. Queste informazioni sono contenute nel messaggio CLM0045I.

Quando i messaggi statistici sono abilitati **-s**, l'esempio riporta le seguenti informazioni statistiche su ciascun controllo coda:

- Tempo impiegato per elaborare le code (in millisecondi)
- Numero di code controllate
- Numero di modifiche attive / inattive effettuate
- Numero di messaggi trasferiti

Queste informazioni sono riportate nel messaggio CLM0048I.

I file di report potrebbero crescere rapidamente in modalità di debug e impacchettarsi rapidamente. In questa situazione, il limite di dimensione di 1 MB per i singoli file potrebbe essere superato.

AMQSCLM: Risoluzione dei problemi
Le seguenti sezioni contengono informazioni sugli scenari che potrebbero essere rilevati durante l'utilizzo dell'esempio. Vengono fornite informazioni sulle potenziali spiegazioni per uno scenario e le opzioni su come risolverlo.

Scenario: AMQSCLM non è in fase di avvio

Spiegazione potenziale: sintassi non corretta.

Azione: controllare l'output dell'errore standard per la sintassi corretta

Spiegazione potenziale: il gestore code non è disponibile.

Azione: controllare l'ID messaggio nel file di report CLM0010E.

Spiegazione potenziale: impossibile aprire o creare file o file di report.

Azione: controllare l'output di errore standard per i messaggi di errore durante l'inizializzazione.

Scenario: AMQSCLM non sta modificando una coda in ATTIVO o INATTIVO

Spiegazione potenziale: la coda non si trova nell'elenco di code da monitorare

Azione: controllare i parametri **-q** e **-f** .

Spiegazione potenziale: la coda non è una coda locale nel cluster corretto.

Azione: verificare che la coda sia locale e nel cluster corretto.

Spiegazione potenziale: AMQSCLM non è in esecuzione per questo gestore code e cluster.

Azione : avviare AMQSCLM per il gestore code e il cluster pertinenti.

Spiegazione potenziale: la coda viene lasciata INATTIVO, **CLWLPRTY** = 0, perché non ha consumer. In alternativa, viene lasciato ATTIVO **CLWLPRTY** > =1, perché ha almeno 1 consumer.

Azione: verificare se le applicazioni che utilizzano sono collegate alla coda.

Spiegazione potenziale: il server dei comandi del gestore code non è in esecuzione.

Azione: verificare la presenza di errori nei file di report.

Scenario: i messaggi non vengono instradati intorno alle code INATTIVO

Spiegazione potenziale: i messaggi vengono inseriti direttamente nel gestore code che possiede la coda inattiva e il valore **CLWLUSEQ** della coda non è ANYe l'argomento **-u** non viene utilizzato per AMQSCLM.

Azione: controllare il valore **CLWLUSEQ** del gestore code pertinente oppure verificare che l'argomento **-u** sia utilizzato per AMQSCLM.

Spiegazione potenziale: non sono presenti code attive su alcun gestore code. I messaggi vengono bilanciati in modo uniforme tra tutte le code inattive fino a quando una coda diventa attiva.

Azione: controllare lo stato delle code su tutti i gestori code.

Spiegazione potenziale: i messaggi vengono inseriti in un gestore code differente nel cluster rispetto a quello proprietario della coda inattiva e il valore **CLWLPRTY** aggiornato 0 non viene propagato al gestore code dell'applicazione di inserimento.

Azione: verificare che i canali cluster tra il gestore code monitorato e il gestore code del repository completo siano in esecuzione. Verificare che i canali tra il gestore code di inserimento e il gestore code del repository completo siano in esecuzione. Controllare i log degli errori dei gestori code del repository completo, di inserimento e monitorati.

Spiegazione potenziale: le istanze della coda remota sono attive (CLWLPRTY=1), ma i messaggi non possono essere instradati a queste istanze della coda perché il canale mittente del cluster dal gestore code locale non è in esecuzione.

Azione: controllare lo stato dei canali mittenti del cluster dal gestore code locale al gestore code remoto, o ai gestori, con un'istanza attiva della coda.

Scenario: AMQSCLM non trasferisce messaggi da una coda inattiva

Spiegazione potenziale: il trasferimento del messaggio non è abilitato (**-t**).

Azione: accertarsi che il trasferimento del messaggio sia abilitato (**-t**).

Spiegazione potenziale: la coda non si trova nell'elenco di code da monitorare.

Azione: controllare i parametri **-q** e **-f** .

Spiegazione potenziale: AMQSCLM non è in esecuzione per questo o per altri gestori code nel cluster che possiedono istanze della stessa coda.

Azione: avviare AMQSCLM.

Spiegazione potenziale: la coda ha **CLWLUSEQ** = LOCAL o **CLWLUSEQ** = QMGRe l'argomento **-u** non è impostato.

Azione: impostare il parametro **-u** o modificare la coda o la configurazione del gestore code su ANY.

Spiegazione potenziale: non ci sono istanze attive della coda nel cluster.

Azione: verificare la presenza di istanze della coda con un valore **CLWLPRTY** pari o superiore a 1.

Spiegazione potenziale: le istanze della coda remota hanno consumer (**IPPROCS** > = 1) ma sono inattive su tali gestori code (**CLWLPRTY** = 0) perché AMQSCLM non sta monitorando tali istanze remote.

Azione: verificare che AMQSCLM sia in esecuzione su tali gestori code e / o che la coda si trovi nell'elenco di code da monitorare controllando i valori dei parametri **-q** e **-f** .

Spiegazione potenziale: le istanze della coda remota sono attive (**CLWLPRTY** = 1), ma vengono visualizzate come inattive sul gestore code locale (**CLWLPRTY** = 0). Questa situazione è dovuta al fatto che il valore **CLWLPRTY** aggiornato non viene propagato a questo gestore code.

Azione: verificare che i gestori code remoti siano connessi ad almeno uno dei gestori code del repository completo nel cluster. Assicurarsi che i gestori code del repository completo funzionino correttamente. Verificare che i canali tra i gestori code del repository completo e i gestori code monitorati siano in esecuzione.

Spiegazione potenziale: i messaggi non sono sottoposti a commit, quindi non sono richiamabili.

Azione: verificare il corretto funzionamento dell'applicazione mittente.

Spiegazione potenziale: AMQSCLM non ha accesso alla coda locale in cui sono accodati i messaggi.

Azione: verificare se AMQSCLM è in esecuzione come utente con autorizzazione sufficiente per accedere alla coda.

Spiegazione potenziale: il server dei comandi del gestore code non è in esecuzione.

Azione: avviare il server dei comandi del gestore code.

Spiegazione potenziale: AMQSCLM ha rilevato un errore.

Azione: verificare la presenza di errori nei file di report.

Spiegazione potenziale: le istanze della coda remota sono attive (CLWLPRTY=1), ma i messaggi non possono essere trasferiti a tali istanze della coda perché il canale mittente del cluster dal gestore code locale non è in esecuzione. Questo è spesso accompagnato da un'avvertenza CLM0030W nel log del report amqsclm.

Azione: controllare lo stato dei canali mittenti del cluster dal gestore code locale al gestore code remoto, o ai gestori, con un'istanza attiva della coda.

ALW *Programma di esempio per Connection Endpoint Lookup (CEPL)*

IBM MQ L'esempio Ricerca endpoint connessione fornisce un modulo di uscita semplice ma potente che offre agli utenti IBM MQ un modo per recuperare le definizioni di connessione da un repository LDAP, ad esempio Tivoli Directory Server.

Tivoli Directory Server v6.3 Client deve essere installato per poter utilizzare CEPL.

Per utilizzare questo esempio è necessaria una conoscenza pratica dell'amministrazione di IBM MQ sulle piattaforme supportate.

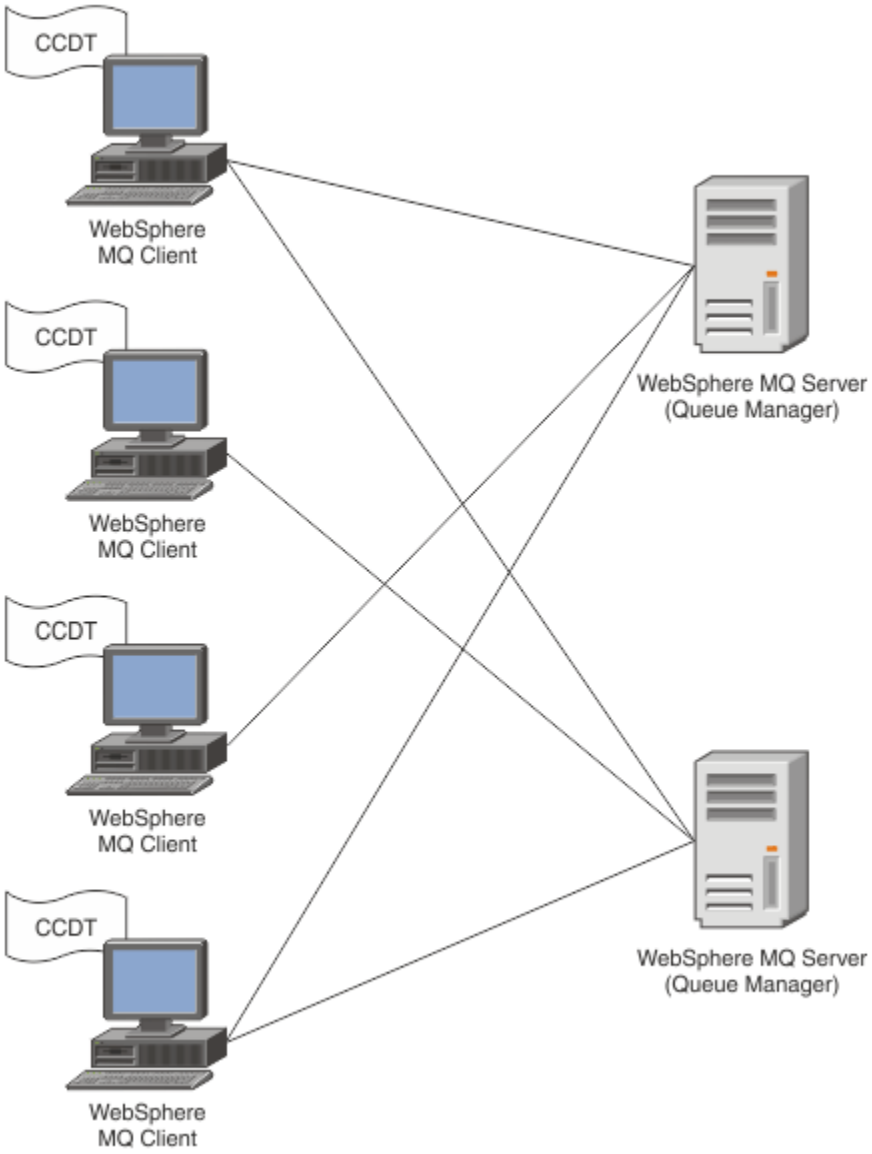
Linux Windows AIX *Introduzione*

Configurare un repository globale, ad esempio una directory LDAP (Lightweight Directory Access Protocol), per memorizzare le definizioni di connessione client per facilitare la gestione e la manutenzione.

Utilizzo di un'applicazione client IBM MQ per stabilire una connessione a un gestore code mediante CCDT (Client Connection Definition Table).

La CCDT viene creata tramite l'interfaccia di gestione MQSC IBM MQ standard. L'utente deve essere connesso a un gestore code per creare definizioni di connessione client, anche se i dati contenuti nella definizione non sono limitati al gestore code. Il file CCDT

generato deve essere distribuito manualmente tra le applicazioni e le macchine client.

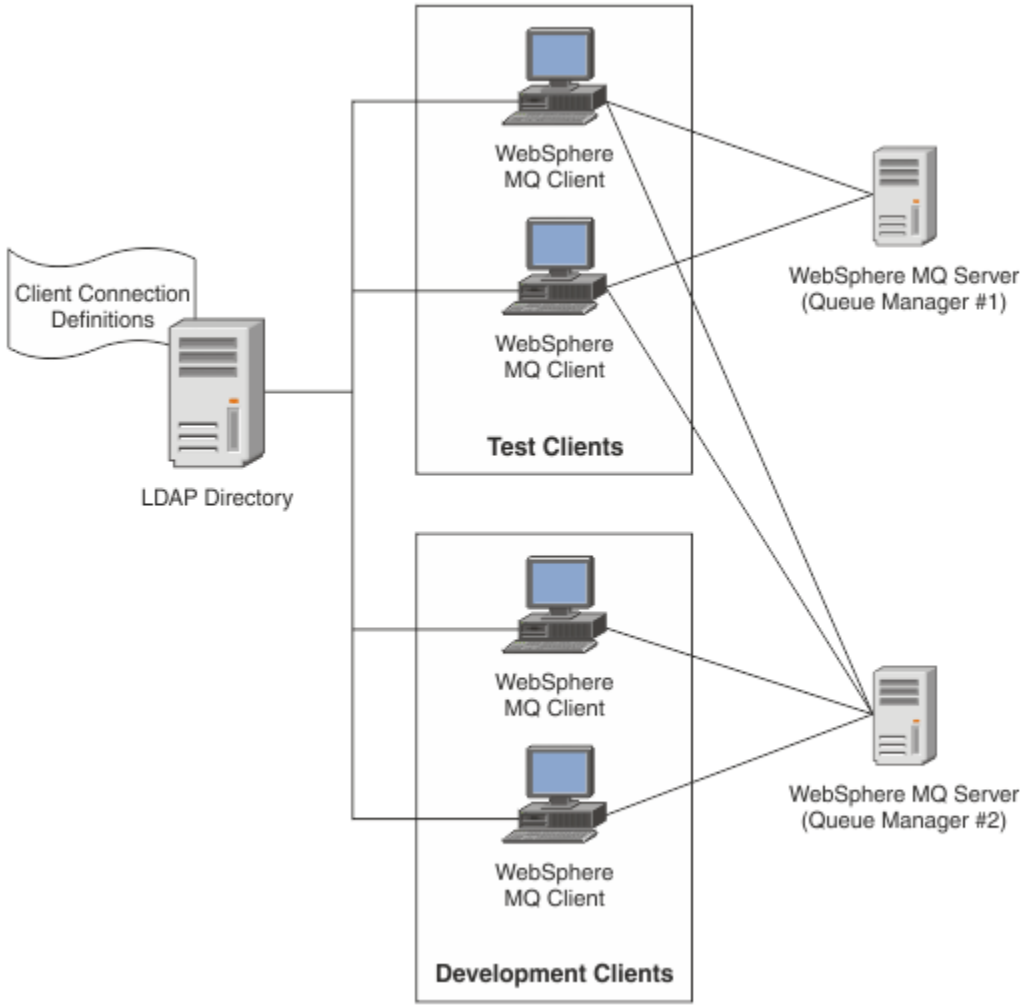


Il file CCDT deve essere distribuito a ciascun client IBM MQ . Dove migliaia di clienti possono esistere sia localmente che globalmente, diventerebbe presto difficile da mantenere e amministrare. Un approccio più flessibile è necessario per garantire che ciascun client disponga delle definizioni client corrette.

Un approccio di questo tipo è quello di memorizzare le definizioni di connessione client in un repository globale come una directory LDAP (Lightweight Directory Access Protocol). Una directory LDAP può anche fornire ulteriori funzionalità di sicurezza, indicizzazione e ricerca, consentendo a ogni client di accedere solo a quelle definizioni di connessione che li riguardano.

La directory LDAP può essere configurata in modo che solo definizioni specifiche siano disponibili per determinati gruppi di utenti. Ad esempio, i client di verifica possono accedere al gestore

code #1 e #2, mentre i client di sviluppo possono accedere solo al gestore code #2 .



Il modulo di uscita può ricercare un repository di LDAP, ad esempio IBM Tivoli Directory Server, per recuperare definizioni di canale. Utilizzando tali definizioni di connessione, un'applicazione client IBM MQ può stabilire una connessione a un gestore code.

Il modulo di uscita è un modulo di uscita di pre - connessione che consente di ottenere la definizione del canale durante la chiamata MQCONN/MQCONNX da un repository LDAP.

Il modulo di uscita e lo schema potrebbero essere implementati da:

- I clienti che hanno già creato una base di competenze utilizzando la tecnologia basata su file CCDT esistente e che desiderano semplificare i costi di gestione e distribuzione.
- Clienti esistenti che utilizzano già la propria tecnologia di proprietà per la distribuzione delle definizioni di connessione client.
- Clienti nuovi o esistenti che attualmente non utilizzano alcun tipo di soluzione di connessione client e che desiderano utilizzare le funzioni offerte da IBM MQ.
- Clienti nuovi o esistenti che desiderano utilizzare direttamente o ottimizzare il proprio modello di messaggistica in linea con qualsiasi architettura di business LDAP corrente.

ALW *Ambienti supportati*

Verificare di disporre di un sistema operativo supportato e del software pertinente prima di eseguire l'esempio Ricerca endpoint di connessione.

Il programma di esempio per IBM MQ Connection Endpoint Lookup richiede il seguente software:

- IBM WebSphere MQ 7.0o versioni successive

- Tivoli Directory Server V6.3 Client o versioni successive

Sistemi operativi supportati:

1. Windows (7/8/2008/2012)
2. AIX
3. Linux
   - RHEL v4 e v5 su System p
   - SUSE v9 e v10 su System p
   - RHEL v4 e v5 x86-64 a 32 bit e 64 bit
   - SUSE v9 e v10 x86-64 32 bit e 64 bit

Nota: L'esempio non è disponibile per le piattaforme seguenti:

- z/OS
- IBM i

ALW *Installazione e configurazione*
Installazione e configurazione del modulo di uscita e schema dell'endpoint di connessione.

Installazione del modulo di uscita

Durante l'installazione di IBM MQ, il modulo di uscita viene installato in `tools/samples/c/preconnexit/bin`. Per le piattaforme a 32 bit, il modulo di uscita deve essere copiare in `exit/`*`installation_name`*`/` prima di poter essere utilizzato. Per le piattaforme a 64 bit, il modulo di uscita deve essere copiato in exit64 /*nome_installazione*/ prima che possa essere utilizzato.

Installazione dello schema endpoint di connessione in corso

L'uscita utilizza lo schema Endpoint di connessione, `ibm-amq.schema`. Il file di schema deve essere importato in qualsiasi server LDAP prima di poter utilizzare l'uscita. Dopo aver importato lo schema, è necessario aggiungere i valori per gli attributi.

Di seguito è riportato un esempio per l'importazione dello schema dell'endpoint di connessione. L'esempio presuppone che venga utilizzato IBM Tivoli Directory Server (ITDS).

- Verificare che IBM Tivoli Directory Server sia in esecuzione, quindi copiare o inviare via FTP il file `ibm-amq.schema` sul server ITDS.
- Sul server ITDS, immettere il seguente comando per installare lo schema nell'archivio ITDS, dove *ID LDAP* e *password LDAP* sono il DN root e la password per il server LDAP:

  `ldapadd -D "`*`LDAP ID`*`" -w "`*`LDAP password`*`" -f ibm-amq.schema`
- In una finestra di comandi, immettere il seguente comando o utilizzare uno strumento di terze parti per sfogliare lo schema per la verifica:

```
ldapsearch objectclass=ibm-amqClientConnection
```

Fare riferimento alla documentazione del server LDAP per ulteriori dettagli sull'importazione del file di schema.

Configurazione

Una nuova sezione denominata PreConnect deve essere aggiunta al file di configurazione client, ad esempio `mqclient.ini`. La sezione PreConnect contiene le seguenti parole chiave:

Modulo
Il nome del modulo contenente il codice di uscita API. Se questo campo contiene il percorso completo del modulo, viene utilizzato così com' è. In caso contrario, viene eseguita la ricerca nella cartella `exit` o `exit64` nell'installazione di IBM MQ .

Funzione
Il nome del punto di immissione funzionale nella libreria che contiene il codice di uscita `LdapPreConnect` . La definizione della funzione aderisce al prototipo della funzione dell'azienda.

Attenzione: È necessario rimuovere le virgolette nell'istruzione della funzione quando si specifica il punto di ingresso di uscita effettivo.

Dati
URI del repository LDAP contenente le definizioni di canali.

Il seguente frammento è un esempio delle modifiche richieste nel file `mqclient.ini` .

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW *Panoramica dell'uscita e dello schema*
Sintassi e parametri utilizzati per stabilire una connessione a un gestore code.

IBM MQ 9.3 definisce la sintassi seguente per un punto di ingresso in un modulo di uscita.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Durante l'esecuzione della chiamata MQCONN/X, IBM MQ C Client carica il modulo di uscita contenente un'implementazione della sintassi della funzione. Richiama quindi una funzione di uscita per richiamare definizioni di canale. Le definizioni di canale richiamati vengono quindi utilizzate per stabilire la connessione a un gestore code.

Parametri

Parametri pExit
Tipo: input/output PMQNXP

La struttura del parametro di uscita PreConnection . La struttura è assegnata e gestita dal chiamante dell'uscita.

```
struct tagMQNXP
{
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     ExitId;              /* Type of exit */
  MQLONG     ExitReason;          /* Reason for invoking exit */
  MQLONG     ExitResponse;        /* Response from exit */
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

Nome pQMgr
Tipo: input/output PMQCHAR

Il nome del gestore code. In fase di input, questo parametro è la stringa di filtro fornita alla chiamata API MQCONN tramite il parametro **QMgrName** . Questo campo potrebbe essere vuoto, esplicito o contenere determinati caratteri jolly. Il campo viene modificato dall'uscita. Il parametro è NULL quando l'uscita viene chiamata con MQXR_TERM.

Opzioni ppConnect

Tipo: ppConnectOpts input/output

Opzioni che controllano l'azione di MQCONNX. Questo è un puntatore a una struttura di opzioni di connessione MQCNO che controlla l'azione della chiamata API MQCONN. Il parametro è NULL quando l'uscita viene chiamata con MQXR_TERM. Il client MQI fornisce sempre una struttura MQCNO all'uscita, anche se non è stata originariamente fornita dall'applicazione. Se un'applicazione fornisce una struttura MQCNO, il client effettua un duplicato per inoltrarlo all'uscita in cui viene modificato. Il client conserva la proprietà di MQCNO. Un MQCD a cui si fa riferimento tramite MQCNO ha la precedenza su qualsiasi definizione di connessione fornita tramite l'array. Il client utilizza la struttura MQCNO per connettersi al gestore code e gli altri vengono ignorati.

Codice pComp

Tipo: input / output PMQLONG

Codice di completamento. Puntatore a un MQLONG che riceve il codice di completamento delle uscite. Deve essere uno dei seguenti valori:

- MQCC_OK - Completamento riuscito
- MQCC_WARNING - Avviso (completamento parziale)
- MQCC_FAILED - Chiamata non riuscita

pReason

Tipo: input / output PMQLONG

Codice di qualificazione motivo pComp. Puntatore ad un MQLONG che riceve il codice motivo di uscita. Se il codice di completamento è MQCC_OK, l'unico valore valido è MQRC_NONE - (0, x '000') Nessun motivo per la notifica.

Se il codice di completamento è MQCC_FAILED o MQCC_WARNING, la funzione di uscita può impostare il campo del codice motivo su qualsiasi valore MQRC_ * valido.

ALW *Informazioni sul contesto LDAP MQ*

L'uscita utilizza la seguente struttura di dati per le informazioni di contesto.

MQNLDAPCTX

La struttura MQNLDAPCTX ha il seguente prototipo C.

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Linux Windows AIX *Codice di esempio per la creazione dell'uscita di ricerca endpoint di connessione*

È possibile utilizzare i frammenti di codice di esempio per compilare l'origine su AIX, o Linuxo Windows.

Compilazione dell'origine

È possibile compilare l'origine con qualsiasi libreria client LDAP, ad esempio, IBM Tivoli Directory Server V6.3 Librerie client. Questa documentazione presuppone che si stiano utilizzando le librerie client Tivoli Directory Server V6.3 .

Nota: La libreria di uscita pre - connessione è supportata con i seguenti server LDAP:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

I seguenti frammenti di codice descrivono come compilare le uscite:

Windows **Compilazione dell'uscita sulla piattaforma Windows**

È possibile utilizzare il seguente frammento per compilare l'origine di uscita:

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

Nota: Se si utilizzano le librerie client IBM Tivoli Directory Server V6.3 compilate con il compilatore Microsoft Visual Studio 2003 , è possibile che vengano visualizzati degli avvisi durante la compilazione delle librerie client IBM Tivoli Directory Server V6.3 con il compilatore Microsoft Visual Studio 2012o versioni successive.

Linux AIX **Compilazione dell'uscita su AIX, Linux**

Il seguente frammento di codice è per compilare l'origine di uscita su Linux. Alcune opzioni del compilatore potrebbero differire su AIX.

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server fornisce sia le librerie di collegamento statiche che quelle dinamiche, ma è possibile utilizzare solo un tipo di libreria. Questo script presuppone che si stiano utilizzando le librerie statiche.

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl
```

```
amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW *Richiamo del modulo di uscita PreConnect*

Il modulo di uscita PreConnect può essere richiamato con tre diversi codici di errore: il codice di errore MQXR_INIT per inizializzare e stabilire una connessione a un server LDAP, il codice di errore MQXR_PRECONNECT per richiamare le definizioni di canale da un server LDAP o il codice di errore MQXR_TERM quando l'uscita deve essere ripulita.

INIT MQXR

L'uscita viene richiamata con il codice motivo MQXR_INIT per l'inizializzazione e la connessione a un server LDAP.

Prima della chiamata MQXR_INIT, il campo pExitDataPtr della struttura MQNXP viene popolato con l'attributo Data dalla sezione PreConnect all'interno del file `mqclient.ini` (ossia, LDAP).

Un URL LDAP consiste almeno del protocollo, del nome host, del numero di porta e del DN di base per la ricerca. L'uscita analizza l' URL LDAP contenuto nel campo pExitDataPtr , assegna una struttura del contesto di ricerca LDAP MQNLDAPCTX e la popola di conseguenza. L'indirizzo di questa struttura viene memorizzato nel campo pExitUserAreaPtr. Se non si analizza correttamente l' URL LDAP si verifica l'errore MQCC_FAILED.

A questo punto, l'uscita si connette e si collega al server LDAP utilizzando i parametri **MQNLDAPCTX** . Anche gli handle API LDAP risultanti vengono memorizzati all'interno di questa struttura.

MQXR_PRECONNECT

Il modulo di exit viene richiamato con il codice di errore MQXR_PRECONNECT per richiamare le definizioni di canale da un server LDAP.

L'uscita ricerca nel server LDAP le definizioni di canale corrispondenti al filtro fornito. Se il **QMgrNameparameter** contiene un nome gestore code specifico, la ricerca restituirà tutte le definizioni di canale per cui il valore dell'attributo LDAP **ibm-amqQueueManagerName** corrisponde al nome gestore code fornito.

Se il parametro **QMgrName** è '*' o ' ' (vuoto), la ricerca restituisce tutte le definizioni di canale per cui l'attributo endpoint **ibm-amqIsClientDefault Connection** è impostato su TRUE.

Dopo una ricerca riuscita, l'exit prepara una o più definizioni MQCD e ritorna al chiamante.

MQXR_TERM

L'uscita viene richiamata con questo codice di errore quando l'uscita deve essere ripulita. Durante questa ripulitura, l'uscita si disconnette dal server LDAP e rilascia tutta la memoria assegnata e gestita dall'uscita, inclusa la struttura MQNLDAPCTX, l'array di puntatori e ogni MQCD a cui fa riferimento. Tutti gli altri campi sono impostati sui valori predefiniti. I parametri di uscita **pQMgrName** e **ppConnectOpts** non vengono utilizzati durante un'uscita con il codice motivo MQXR_TERM e possono essere NULL.

Riferimenti correlati

Stanza PreConnect del file di configurazione client

ALW *Schemi LDAP*

I dati di connessione client vengono memorizzati in un repository globale denominato LDAP (Lightweight Directory Access Protocol). Un client IBM MQ utilizza una directory LDAP per ottenere definizioni di connessione. La struttura di definizioni di connessione client IBM MQ all'interno della directory LDAP è nota come schema LDAP. Uno schema LDAP è la raccolta di definizioni del tipo di attributo, definizioni della classe di oggetti e altre informazioni che un server utilizza per determinare se un filtro o un'asserzione del valore di attributo corrisponde agli attributi di una voce e se consentire, aggiungere e modificare le operazioni.

Memorizzazione dei dati nella directory LDAP

Le definizioni di connessione client si trovano in un ramo specifico all'interno della struttura di directory nota come punto di connessione. Come tutti gli altri nodi all'interno di una directory LDAP, al punto di

connessione è associato un DN (Distinguished Name). È possibile utilizzare questo nodo come punto di partenza per qualsiasi query eseguita sulla directory. Utilizzare il filtro quando si interroga la directory LDAP per restituire un sottoinsieme di definizioni di connessione client. È possibile limitare l'accesso alle strutture ad albero secondarie in base alle autorizzazioni concesse in altre parti della struttura ad albero di directory, ad esempio, a utenti, dipartimenti o gruppi.

Definizione di attributi e classi personali

Memorizzare la definizione di canale client modificando lo schema LDAP. Tutte le definizioni di dati LDAP richiedono oggetti e attributi. Gli oggetti e gli attributi sono identificati da un numero OID (object identifier) che identifica in modo univoco l'oggetto o l'attributo. Tutte le classi all'interno di uno schema LDAP ereditano direttamente o indirettamente dall'oggetto superiore. L'oggetto di definizione del canale client contiene gli attributi dell'oggetto superiore. Tutte le definizioni di dati LDAP richiedono oggetti e attributi:

- Le definizioni oggetto sono raccolte di attributi LDAP.
- Gli attributi sono tipi di dati LDAP.

La descrizione di ciascun attributo e il modo in cui vengono associati alle normali proprietà IBM MQ sono descritti in Attributi LDAP.

ALW *Attributi LDAP*

Gli attributi LDAP definiti sono specifici di IBM MQ e vengono associati direttamente alle proprietà di connessione client.

Attributi stringa directory canale client IBM MQ

Gli attributi della stringa di caratteri con la relativa associazione alle proprietà IBM MQ sono riportati nella seguente tabella. Gli attributi possono contenere i valori della sintassi directoryString (Unicode con codificaUTF-8 , ossia un sistema di codifica a byte variabile che include IA5/ASCII come sottoinsieme). La sintassi è specificata dal relativo OID (object identification number).

Tabella 166. Attributi stringa directory del canale client IBM MQ

Attributo LDAP	Descrizione	IBM MQ Proprietà
CN	Il nome comune composto dal nome del canale e dal nome del gestore code di definizione.	
ibm -amqChannelamqChannel	Il nome della definizione del canale.	CHANNEL
ibm -amqConnectionamqConnection	L'identificativo della connessione di comunicazione.	CONNAME
ibm -amqDescription	La descrizione del canale.	DESCR
ibm -amqLocalIndirizzo	L'indirizzo di comunicazione locale del canale.	LOCLADDR
ibm -amqModeNome	Il nome della modalità LU 6.2.	MODENAME
ibm -amqPassword	La password che può essere utilizzata.	Password
ibm -amqQueueManagerName	Il nome del gestore code o del gruppo di gestori code a cui un'applicazione client IBM MQ può richiedere la connessione.	QMNAME
ibm -amqSecurityExitUserDati	I dati utente passati all'uscita di sicurezza.	SCYDATA
ibm -amqSecurityExitName	Il nome del programma di uscita che deve essere eseguito dall'uscita di sicurezza del canale.	SCYEXIT
ibm -amqSslCipherSpec	Un singolo CipherSpec per una connessione TLS.	SSLCIPH

Tabella 166. Attributi stringa directory del canale client IBM MQ (Continua)		
Attributo LDAP	**Descrizione**	**IBM MQ Proprietà**
ibm -amqSslPeerName	Verifica il DN (Distinguished Name) del certificato dal gestore code peer o dal client all'altra estremità di un canale IBM MQ .	SSLPEER
ibm -amqTransactionProgramName	Il nome del programma di transazione.	TPNAME
ibm - IDamqUser	L'ID utente che l'MCA deve utilizzare quando tenta di avviare una sessione SNA sicura con un MCA remoto.	USERID

Attributi numero intero connessione client IBM MQ

Gli attributi con valori predefiniti (ad esempio, un tipo enumerato) vengono memorizzati come numeri interi standard. Questi valori vengono memorizzati nella directory LDAP come valori interi e non utilizzando il nome costante associato.

Tabella 167. Attributi interi della directory del canale client IBM MQ		
Attributo LDAP	**Descrizione**	**IBM MQ Proprietà**
ibm -amqConnectionAffinità	Determina se le applicazioni client, che si connettono più volte tramite lo stesso nome gestore code, utilizzano lo stesso canale client.	AFFINITÀ
ibm -amqClientChannelWeight	Una ponderazione per influenzare quale definizione di canale di connessione client viene utilizzata.	CLNTWGHT
ibm -amqHeartBeatInterval	Il tempo approssimativo tra i flussi di heartbeat che devono essere inviati da un MCA di invio quando non sono presenti messaggi nella coda di trasmissione.	HBINT
ibm -amqKeepAliveInterval	Un valore di timeout per un canale.	KAINT
ibm -amqMaximumMessageLength	La lunghezza massima di un messaggio che è possibile trasmettere sul canale.	MAXMSGL
ibm -amqSharingConversazioni	Il numero massimo di conversazioni che condividono ciascuna istanza del canale TCP/IP.	SHARECNV
ibm -amqTransportamqTransport	Il tipo di trasporto da utilizzare.	TRPTYPE

Attributo booleano del canale client IBM MQ

Questo attributo booleano non è associato ad alcuna proprietà IBM MQ . La sintassi di questo attributo indica un valore booleano.

Tabella 168. Attributo booleano del canale client IBM MQ	
Attributo LDAP	**Descrizione**
ibm -amqIsClientDefault	Questo attributo booleano viene definito per risolvere il problema di ricerca delle voci il cui attributo ibm -amqQueueManagerName non è stato definito.

Attributi elenco canali client IBM MQ

Le proprietà IBM MQ vengono memorizzate come attributo elenco separato da virgole a valore singolo all'interno della directory LDAP. Gli attributi vengono definiti allo stesso modo degli altri attributi della stringa della directory. Gli attributi di elenco e la loro mappatura alle proprietà IBM MQ sono descritti nella seguente tabella.

Tabella 169. Attributi elenco canali client IBM MQ

Attributo LDAP	Descrizione	IBM MQ Proprietà
ibm -amqHeaderCompressione	Un elenco delle tecniche di compressione dei dati di intestazione supportate dal canale.	COMPHDR
ibm -amqMessageCompressione	Un elenco di tecniche di compressione dei dati dei messaggi supportate dal canale.	COMPMSG
ibm -amqSenddatiExitUser	I dati utente passati all'uscita di invio.	SENDDATA
ibm -amqSendNomeExitUser	Il nome del programma di uscita che deve essere eseguito dall'uscita di invio del canale.	SENDEXIT
ibm -amqReceivedatiExitUser	I dati utente passati all'uscita di ricezione.	RCVDATA
ibm -amqReceiveExitName	Il nome del programma di uscita utente che deve essere eseguito dall'uscita utente di ricezione del canale.	RCVEXIT

ALW *Nome comune (Common Name)*

Il CN (common name) è costituito dal nome del canale e dal nome del gestore code di definizione.

È un attributo preesistente.

Il formato del CN è:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

Ad esempio:

```
CN=TC1(QM_T1)
```

È possibile specificare solo un valore per questo attributo.

Questo attributo è un attributo stringa e i valori non sono sensibili al maiuscolo / minuscolo. La corrispondenza della sottostringa viene ignorata. La corrispondenza della sottostringa è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca, utilizzando una sottostringa (ad esempio, CN=jim * dove CN è un attributo) e contiene uno o più caratteri jolly.

ALW *ibm - NomeamqChannel*

Questo attributo specifica il nome della definizione di canale.

Questo attributo ha un singolo valore di stringa con un massimo di 20 caratteri non sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca, utilizzando una stringa secondaria e contiene uno o più caratteri jolly.

ALW *ibm -amqDescription*

Questo attributo LDAP fornisce la descrizione del canale.

Questo attributo ha un singolo valore stringa con un massimo di 64 byte, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqConnectionNome*
Questo attributo LDAP è l'identificativo della connessione di comunicazione. Specifica i particolari collegamenti di comunicazione che devono essere utilizzati da questo canale.

Questo attributo ha un singolo valore stringa con un massimo di 264 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm - IndirizzoamqLocal*
Questo attributo specifica l'indirizzo di comunicazione locale per il canale.

Questo attributo ha un singolo valore stringa con un massimo di 48 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqModeNome*
Questo attributo è per l'utilizzo con connessioni LU 6.2. Dà una definizione extra per le caratteristiche di sessione della connessione quando viene eseguita un'assegnazione di sessione di comunicazione.

Questo attributo ha un singolo valore di stringa di esattamente 8 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqPassword*
Questo attributo LDAP specifica una password che può essere utilizzata da MCA quando si tenta di avviare una sessione LU sicura 6.2 con un MCA remoto.

Questo attributo ha un singolo valore intero con un massimo di 12 cifre. Non è un attributo preesistente.

ALW *ibm -amqQueueManagerName*
Questo attributo specifica il nome del gestore code o del gruppo di gestori code a cui un'applicazione client IBM MQ può richiedere la connessione.

Questo attributo ha un singolo valore stringa con un massimo di 48 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

Riferimenti correlati
“ibm -amqIsClientDefault” a pagina 1161
Questo attributo booleano risolve il problema di ricerca delle voci in cui l'attributo ibm -amqQueueManagerName non è stato definito.

ALW *Dati ibm -amqSecurityExitUser*
Questo attributo LDAP specifica i dati utente passati all'uscita di sicurezza.

Questo attributo ha un singolo valore stringa con un massimo di 999 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqSecurityExitName*
Questo attributo LDAP specifica il nome del programma di uscita che deve essere eseguito dall'uscita di sicurezza del canale.

Lasciare vuoto se non è attiva alcuna uscita di sicurezza del canale.

Questo attributo ha un singolo valore stringa con un massimo di 999 caratteri, che non sono sensibili al maiuscolo / minuscolo. Questo attributo non è pre - esistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqSslCipherSpec*
Questo attributi LDAP specifica un singolo CipherSpec per una connessione TLS.

Questo attributo ha un singolo valore stringa con un massimo di 32 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqSslPeerName*
Questo attributo LDAP viene utilizzato per controllare il DN (Distinguished Name) del certificato dal gestore code peer o client all'altra estremità di un canale IBM MQ .

Questo attributo LDAP ha un singolo valore stringa con un massimo di 1024 byte, che non sono sensibili al maiuscolo / minuscolo. Non è una preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqTransactionProgramName*
Questo attributo LDAP specifica il nome del programma di transazione. Viene utilizzato con le connessioni LU 6.2 .

Questo attributo ha un singolo valore stringa con un massimo di 64 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è una preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *ibm -amqUserID*
Questo attributo LDAP specifica l'ID utente che deve essere utilizzato da MCA quando si tenta di avviare una sessione SNA sicura con un MCA remoto.

Questo attributo ha un singolo valore di stringa di esattamente 12 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

ALW *Affinità ibm -amqConnection*
Questo attributo LDAP specifica se le applicazioni client, che si collegano più volte utilizzando lo stesso nome gestore code, utilizzano lo stesso canale client.

Questo attributo ha un singolo valore intero. Non è un attributo preesistente.

ALW *ibm -amqClientChannelWeight*
Questo attributo LDAP specifica una ponderazione che influenza quale definizione di canale di connessione client viene utilizzata.

L'attributo di ponderazione del canale client viene utilizzato per modificare la selezione delle definizioni del canale client quando è disponibile più di una definizione adatta.

Questo attributo ha un singolo valore intero. Non è un attributo preesistente.

ALW *ibm -amqHeartBeatInterval*
Questo attributo LDAP specifica il tempo approssimativo tra i flussi heartbeat che devono essere passati da un MCA di invio quando non ci sono messaggi nella coda di trasmissione.

Questo attributo ha un singolo valore intero. Non è un attributo preesistente. Il valore predefinito è 1. Il valore predefinito è impostato nell'operazione della variabile di ambiente MQSERVER corrente.

ALW *ibm -amqKeepAliveInterval*
Questo attributo LDAP viene utilizzato per specificare un valore di timeout per un canale.

Il valore di questo attributo viene passato allo stack di comunicazioni specificando il tempo keepalive per il canale. È possibile utilizzarlo per specificare un valore keepalive diverso per ogni canale.

Questo attributo ha un singolo valore intero. Non è un attributo preesistente.

ALW *ibm -amqMaximumMessageLength*
Questo attributo LDAP specifica la lunghezza massima di un messaggio che può essere trasmesso sul canale.

Il valore predefinito di questo attributo è 104857600 come da operazione della variabile di ambiente MQSERVER corrente. Questo attributo ha un singolo valore intero e non è un attributo preesistente.

ALW *ibm - ConversazioniamqSharing*
Questo attributo LDAP specifica il numero massimo di conversazioni che condividono ciascuna istanza del canale TCP/IP.

Questo attributo ha un singolo valore intero. Questo attributo non è un attributo preesistente.

ALW *ibm - TipoamqTransport*
Questo attributo LDAP specifica il tipo di trasporto da utilizzare.

Questo attributo ha un singolo valore intero. Non è un attributo preesistente.

ALW *ibm -amqIsClientDefault*
Questo attributo booleano risolve il problema di ricerca delle voci in cui l'attributo ibm -amqQueueManagerName non è stato definito.

I moduli di uscita di preconnessione generalmente ricercano i server LDAP con il valore dell'attributo ibm -amqQueueManagerName come criterio di ricerca. Tale query restituisce tutte le voci in cui il valore dell'attributo ibm -amqQueueManagerName corrisponde al nome del gestore code specificato nella chiamata MQCONN/X. Tuttavia, quando si utilizzano le tabelle di definizione del canale client (CCDT),

è possibile impostare il nome del gestore code su una chiamata MQCONN/X come vuoto oppure anteporre al nome un asterisco (*). Se il nome del gestore code è vuoto, il client si connette al gestore code predefinito. Se il nome è preceduto da un asterisco (*) per il gestore code, il client connette qualsiasi gestore code.

Allo stesso modo, l'attributo ibm -amqQueueManagerName in una voce può essere lasciato indefinito. In questo caso, si prevede che il client che utilizza queste informazioni sull'endpoint possa connettersi a qualsiasi gestore code. Ad esempio, una voce contiene le seguenti righe:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

In questo esempio, il client tenta di connettersi al gestore code specificato in esecuzione su `myhost`.

Tuttavia, nei server LDAP, non viene eseguita una ricerca su un valore di attributo che non è stato definito. Ad esempio, se una voce contiene le informazioni di connessione tranne ibm -amqQueueManagerName, i risultati della ricerca non includeranno questa voce. Per risolvere questo problema, puoi impostare ibm -amqIsClientDefault. Si tratta di un attributo booleano e si presume che abbia un valore FALSE se non definito.

Per le voci in cui ibm -amqQueueManagerName non è stato definito e si prevede che facciano parte della ricerca, impostare ibm -amqIsClientDefault su TRUE. Quando viene specificato uno spazio vuoto o un asterisco (*) come nome del gestore code in una chiamata a MQCONN/X, l'uscita di preconnessione ricerca nel server LDAP tutte le voci in cui il valore di attributo ibm -amqIsClientDefault è impostato su TRUE.

Nota: Non impostare o definire l'attributo ibm -amqQueueManagerName se ibm -amqIsClientDefault è impostato su TRUE.

Riferimenti correlati

“ibm -amqQueueManagerName” a pagina 1159
Questo attributo specifica il nome del gestore code o del gruppo di gestori code a cui un'applicazione client IBM MQ può richiedere la connessione.

ALW *Compressione ibm -amqHeader*
Questo attributo LDAP è un elenco di tecniche di compressione dei dati di intestazione supportate dal canale.

La dimensione massima di questo attributo è 48 caratteri. Non è un attributo preesistente.

È possibile specificare solo un valore per questo attributo.

Questo attributo elenco viene specificato come stringhe di directory utilizzando un formato separato da virgole. Ad esempio, il valore specificato per **ibm-amqHeaderCompression** è `0` , associato a `NONE`. Tutti i valori che superano il limite massimo consentito vengono ignorati dal client. Ad esempio, ibm -amqHeaderCompression contiene un massimo di 2 numeri interi nell'elenco.

ALW *ibm - CompressioneamqMessage*
Questo attributo LDAP è un elenco di tecniche di compressione dei dati dei messaggi supportate dal canale.

La dimensione massima di questo attributo è 48 caratteri. Non è un attributo preesistente.

Questo attributo non supporta più valori.

Questo attributo elenco viene specificato come stringhe di directory utilizzando un formato separato da virgole. Ad esempio, il valore specificato per questo attributo è 1,2,4, che si associa alla sequenza di compressione sottostante RLE, ZLIBFAST e ZLIBHIGH.

Tutti i valori che superano il limite massimo consentito vengono ignorati dal client. Ad esempio, la compressione ibm -amqMessagecontiene un massimo di 16 numeri interi nell'elenco.

ALW *ibm - DatiExitUseramqSend*
Questo attributo LDAP specifica i dati utente passati all'uscita di invio.

Questo attributo LDAP ha un singolo valore di stringa con un massimo di 999 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

Nota: ibm-amqSendExitName e **ibm-amqSendExitUserData** devono essere sincronizzati in coppie. I dati utente devono essere sincronizzati con il nome di uscita. Quindi, se uno è specificato, anche l'altro deve essere specificato simmetricamente, anche se non contiene dati.

ALW *ibm -amqSendExitName*
Questo attributo LDAP specifica il nome del programma exit che deve essere eseguito dall'exit di invio del canale.

Questo attributo ha un singolo valore stringa con un massimo di 999 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

Nota: ibm-amqSendExitName e **ibm-amqSendExitUserData** devono essere sincronizzati in coppie. I dati utente devono essere sincronizzati con il nome uscita. Quindi, se uno è specificato, anche l'altro deve essere specificato simmetricamente anche se non contiene dati.

ALW *ibm - DatiExitUseramqReceive*
Questo attributo LDAP specifica i dati utente passati all'uscita di ricezione.

È possibile eseguire una sequenza di uscite di ricezione. La stringa di dati utente per una serie di uscite è separata da una virgola, spazi o entrambi.

Questo attributo ha un singolo valore stringa con un massimo di 999 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

Nota: ibm-amqReceiveExitName e **ibm-amqReceiveExitUserData** devono essere sincronizzati in coppie. I dati utente devono essere sincronizzati con il nome uscita. Quindi, se uno è specificato, anche l'altro deve essere specificato simmetricamente anche se non contiene dati.

ALW *ibm -amqReceiveExitName*
Questo attributo LDAP specifica il nome del programma di uscita utente che deve essere eseguito dall'uscita utente di ricezione del canale.

Questo attributo è un elenco di nomi di programmi da eseguire in successione. Lasciare vuoto, se non è attiva alcuna uscita utente di ricezione del canale.

Questo attributo ha un singolo valore stringa con un massimo di 999 caratteri, che non sono sensibili al maiuscolo / minuscolo. Non è un attributo preesistente.

La corrispondenza della sottostringa viene ignorata. La corrispondenza della stringa secondaria è una regola di corrispondenza utilizzata nello schema secondario che specifica il comportamento dell'attributo in un filtro di ricerca.

Nota: ibm-amqReceiveExitName e **ibm-amqReceiveExitUserData** devono essere sincronizzati in coppie. I dati utente devono essere sincronizzati con il nome uscita. Quindi, se uno è specificato, anche l'altro deve essere specificato simmetricamente, anche se non contiene dati.

z/OS Utilizzo dei programmi di esempio per z/OS

Le applicazioni procedurali di esempio fornite con IBM MQ for z/OS dimostrano gli utilizzi tipici di MQI (Message Queue Interface).

Informazioni su questa attività

IBM MQ for z/OS fornisce anche uscite di conversione dati di esempio, descritte in “Scrittura delle uscite di conversione dati” a pagina 985.

Tutte le applicazioni di esempio sono fornite in formato di origine; molte sono fornite anche in formato eseguibile. I moduli di origine includono pseudocodice che descrive la logica del programma.

Nota: Sebbene alcune delle applicazioni di esempio dispongano di interfacce basate su pannelli di base, non mirano a dimostrare come progettare l'aspetto delle applicazioni. Per ulteriori informazioni su come progettare interfacce basate su pannelli per terminali non programmabili, consultare il manuale *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) e il relativo addendum (GG22-9508). Queste linee guida consentono di progettare applicazioni coerenti all'interno dell'applicazione e tra altre applicazioni.

Procedura

- Utilizzare i seguenti collegamenti per ulteriori informazioni sui programmi di esempio:
  - “Funzioni dimostrate nelle applicazioni di esempio per z/OS” a pagina 1164
  - “Preparazione ed esecuzione di applicazioni di esempio per l'ambiente batch su z/OS” a pagina 1171
  - “Preparazione di applicazioni di esempio per l'ambiente TSO su z/OS” a pagina 1174
  - “Preparazione delle applicazioni di esempio per l'ambiente CICS su z/OS” a pagina 1176
  - “Preparazione dell'applicazione di esempio per l'ambiente IMS su z/OS” a pagina 1180
  - “Gli esempi Put su z/OS” a pagina 1181
  - “Gli esempi Get su z/OS” a pagina 1183
  - “L'esempio Sfoglia su z/OS” a pagina 1186
  - “L'esempio Stampa messaggio su z/OS” a pagina 1188
  - “L'esempio Attributi coda su z/OS .” a pagina 1192
  - “L'esempio Gestore posta su z/OS” a pagina 1193
  - “L'esempio Credit Check su z/OS” a pagina 1200
  - “L'esempio Gestore messaggi su z/OS” a pagina 1212
  - “L'esempio Put asincrono su z/OS” a pagina 1216
  - “L'esempio di consumo asincrono batch su z/OS” a pagina 1217
  - “L'esempio CICS Utilizzo asincrono e pubblicazione / sottoscrizione su z/OS” a pagina 1218
  - “L'esempio di pubblicazione / sottoscrizione su z/OS” a pagina 1221
  - “L'esempio di proprietà del messaggio Set and Inquire su z/OS” a pagina 1223

Attività correlate

“Utilizzo dei programmi di esempio su Multiplatforms” a pagina 1061
Questi programmi procedurali di esempio vengono forniti con il prodotto. Gli esempi sono scritti in C e COBOL e dimostrano gli utilizzi tipici di MQI (Message Queue Interface).

z/OS *Funzioni dimostrate nelle applicazioni di esempio per z/OS*

Questa sezione riepiloga le funzioni MQI dimostrate in ciascuna delle applicazioni di esempio, mostra i linguaggi di programmazione in cui è scritto ciascun esempio e l'ambiente in cui viene eseguito ciascun esempio.

z/OS *Inserisci esempi su z/OS*

Gli esempi Put dimostrano come inserire i messaggi su una coda utilizzando la chiamata MQPUT.

L'applicazione utilizza queste chiamate MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

Il programma viene distribuito in COBOL e C e viene eseguito in ambiente batch e CICS . Consultare Tabella 172 a pagina 1172 per l'applicazione batch e Tabella 179 a pagina 1177 per l'applicazione CICS .

z/OS *Ottieni esempi su z/OS*

Gli esempi Get dimostrano come richiamare i messaggi da una coda utilizzando la chiamata MQGET.

L'applicazione utilizza queste chiamate MQI:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

Il programma viene distribuito in COBOL e C e viene eseguito in ambiente batch e CICS . Consultare Tabella 172 a pagina 1172 per l'applicazione batch e Tabella 179 a pagina 1177 per l'applicazione CICS .

z/OS *Sfoglia esempio su z/OS*

L'esempio Sfoglia mostra come utilizzare l'opzione Sfoglia per trovare un messaggio, stamparlo, quindi esaminare i messaggi su una coda.

L'applicazione utilizza queste chiamate MQI:

- MQCONN
- MQOPEN
- MQGET per l'esplorazione dei messaggi
- MQCLOSE
- MQDISC

Il programma viene fornito nei linguaggi COBOL, assembler, PL/I e C. L'applicazione viene eseguita nell'ambiente batch. Consultare Tabella 173 a pagina 1172 per l'applicazione batch.

z/OS *Esempio di messaggio di stampa su z/OS*

L'esempio Stampa messaggio mostra come rimuovere un messaggio da una coda e stampare i dati nel messaggio, insieme a tutti i campi del descrittore del messaggio. Può, facoltativamente, visualizzare tutte le proprietà del messaggio associate a ciascun messaggio.

Rimuovendo i caratteri di commento da due righe nel modulo origine, è possibile modificare il programma in modo che ricerchi, invece di rimuovere, i messaggi su una coda. Questo programma può essere utilmente utilizzato per diagnosticare i problemi con un'applicazione che sta inserendo i messaggi su una coda.

L'applicazione utilizza queste chiamate MQI:

- MQCONN
- MQOPEN

- MQGET per la rimozione di messaggi da una coda (con un'opzione per sfogliare)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

Il programma viene fornito in linguaggio C. L'applicazione viene eseguita nell'ambiente batch. Consultare Tabella 174 a pagina 1173 per l'applicazione batch.

z/OS *Esempio di Attributi coda su z/OS*

L'esempio Attributi coda mostra come analizzare e impostare i valori degli attributi dell'oggetto IBM MQ for z/OS .

L'applicazione utilizza queste chiamate MQI:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

Il programma viene fornito nei linguaggi COBOL, assembler e C. L'applicazione viene eseguita nell'ambiente CICS . Consultare Tabella 180 a pagina 1178 per l'applicazione CICS .

z/OS *Esempio Gestore posta su z/OS*

Considerazioni da tenere presente quando si utilizza l'esempio Gestore posta.

L'esempio Gestore posta illustra queste tecniche:

- Utilizzo delle code alias
- Utilizzo di una coda modello per creare una coda dinamica temporanea
- Utilizzo delle code di risposta
- Utilizzo dei punti di sincronizzazione negli ambienti CICS e batch
- Invio dei comandi alla coda di input dei comandi di sistema
- Verifica dei codici di ritorno
- Invio di messaggi ai gestori code remoti, utilizzando una definizione locale di una coda remota e inserendo i messaggi direttamente su una coda denominata su un gestore code remoto

L'applicazione utilizza queste chiamate MQI:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Vengono fornite tre versioni dell'applicazione:

- Un'applicazione CICS scritta in COBOL
- Un'applicazione TSO scritta in COBOL
- Un'applicazione TSO scritta in C

Le applicazioni TSO utilizzano l'adattatore batch IBM MQ for z/OS e includono alcuni pannelli ISPF .

Consultare Tabella 177 a pagina 1175 per le applicazioni TSO e Tabella 181 a pagina 1178 per le applicazioni CICS .

z/OS *Esempio di verifica del credito su z/OS*

Queste informazioni contengono punti da considerare quando si utilizza l'esempio Controllo credito.

L'esempio Credit Check è una serie di programmi che dimostrano queste tecniche:

- Sviluppo di un'applicazione eseguita in più di un ambiente
- Utilizzo di una coda modello per creare una coda dinamica temporanea
- Utilizzo di un identificativo di correlazione
- Impostazione e trasmissione delle informazioni di contesto
- Utilizzo della priorità e della persistenza dei messaggi
- Avvio di programmi utilizzando il trigger
- Utilizzo delle code di risposta
- Utilizzo delle code alias
- Utilizzo di una coda di messaggi non recapitabili
- Utilizzo di un elenco nomi
- Verifica dei codici di ritorno

L'applicazione utilizza queste chiamate MQI:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET per la ricerca e il richiamo dei messaggi, utilizzando le opzioni di attesa e segnale e per il richiamo di un messaggio specifico
- MQINQ
- MQSET
- MQCLOSE

L'esempio può essere eseguito come applicazione CICS autonoma. Tuttavia, per dimostrare come progettare un'applicazione di accodamento dei messaggi che utilizza le funzioni fornite dagli ambienti CICS e IMS , viene fornito anche un modulo come un programma di elaborazione dei messaggi batch IMS .

I programmi CICS vengono forniti in C e COBOL. Il singolo programma IMS viene fornito in C.

Consultare Tabella 182 a pagina 1179 per l'applicazione CICS e Tabella 184 a pagina 1181 per l'applicazione IMS .

z/OS *L'esempio Gestore messaggi su z/OS*

L'esempio Gestore messaggi consente di ricercare, inoltrare ed eliminare i messaggi su una coda.

L'applicazione utilizza queste chiamate MQI:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET

- MQCLOSE
- MQDISC

Il programma è fornito in linguaggi di programmazione C e COBOL. L'applicazione viene eseguita in TSO. ConsultareTabella 178 a pagina 1176per l'applicazione TSO.

z/OS *Esempi di exit di accodamento distribuito su z/OS*

Una tabella di programmi di origine degli esempi di exit di accodamento distribuito.

I nomi dei programmi di origine degli esempi di uscita di accodamento distribuiti sono elencati nella seguente tabella:

Tabella 170. Origine per gli esempi di exit di accodamento distribuito

Nome membro	Per lingua	Descrizione	Fornito nella libreria
CSQ4BAX0	Assembler	Programma di origine	SSQASM
CSQ4BCX1	C	Programma di origine	SCSQC37S
CSQ4BCX2	C	Programma di origine	SCSQC37S
CSQ4BCX4	C	Programma di origine	SCSQC37S

Nota: I programmi di origine sono collegati - modificati con CSQXSTUB.

z/OS *Esempi di uscita di conversione dati su z/OS*

Viene fornita una struttura per una routine di uscita di conversione dati e viene fornito un esempio con IBM MQ che illustra la chiamata MQXCNVC.

I nomi dei programmi di origine degli esempi di uscita conversione dati sono elencati nella seguente tabella:

Tabella 171. Origine per gli esempi di uscita conversione dati (solo linguaggio assembler)

Nome membro	Descrizione	Fornito nella libreria
CSQ4BAX8	Programma di origine	SSQASM
CSQ4BAX9	Programma di origine	SSQASM
CSQ4CAX9	Programma di origine	SSQASM

Nota: I programmi di origine sono collegati con CSQASTUB.

Per ulteriori informazioni, consultare “Scrittura delle uscite di conversione dati” a pagina 985.

z/OS *Esempi di pubblicazione / sottoscrizione su z/OS*

I programmi di esempio di pubblicazione / sottoscrizione dimostrano l'utilizzo delle funzioni di pubblicazione e sottoscrizione in IBM MQ.

Esistono quattro programmi di esempio di linguaggio di programmazione C e due COBOL che dimostrano come programmare l'interfaccia di pubblicazione / sottoscrizione IBM MQ .

Le applicazioni utilizzano queste chiamate MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE

- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

I programmi di esempio Public / Subscribe vengono forniti nei linguaggi di programmazione C e COBOL. Le applicazioni di esempio vengono eseguite nell'ambiente batch. Consultare Esempi di pubblicazione / sottoscrizione per le applicazioni batch.

z/OS *Configurazione di un gestore code per accettare le connessioni client su z/OS*

Prima di poter eseguire le applicazioni di esempio, è necessario creare un gestore code. È quindi possibile configurare il gestore code per accettare in modo sicuro le richieste di connessione in entrata dalle applicazioni in esecuzione in modalità client.

Prima di iniziare

Verificare che il gestore code esista già e che sia stato avviato. Determinare se i record di autenticazione di canale sono già abilitati emettendo il comando MQSC:

```
DISPLAY QMGR CHLAUTH
```

Importante: Questa attività prevede che i record di autenticazione di canale siano abilitati. Se si tratta di un gestore code utilizzato da altri utenti e applicazioni, la modifica di questa impostazione influirà su tutti gli altri utenti e applicazioni. Se il gestore code non utilizza i record di autenticazione di canale, il passo 4 può essere sostituito con un metodo di autenticazione alternativo (ad esempio un'uscita di sicurezza) che imposta MCAUSER su *non - privileged - user - id* che si otterrà nel passo “1” a pagina 1169.

È necessario conoscere il nome del canale che l'applicazione prevede di utilizzare in modo che l'applicazione possa utilizzare il canale. È inoltre necessario conoscere quali oggetti, ad esempio code o argomenti, l'applicazione prevede di utilizzare in modo che l'applicazione possa utilizzarli.

Informazioni su questa attività

Questa attività crea un ID utente non privilegiato da utilizzare per un'applicazione client che si connette al gestore code. L'accesso viene concesso all'applicazione client solo per essere in grado di utilizzare il canale di cui ha bisogno e la coda di cui ha bisogno utilizzando questo ID utente.

Procedura

1. Ottenere un ID utente sul sistema su cui è in esecuzione il gestore code.

   Per questa attività questo ID utente non deve essere un utente di gestione privilegiato. Questo ID utente è l'autorità con cui la connessione client verrà eseguita sul gestore code.
2. Avviare un programma listener.
   a) Assicurarsi che l'iniziatore del canale sia avviato. In caso contrario, avviarlo immettendo il comando **START CHINIT** .
   b) Avviare il programma listener emettendo il seguente comando:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   dove *nnnn* è il numero di porta scelto.
3. Se l'applicazione utilizza il SISTEMA SYSTEM.DEF.SVRCONN quindi questo canale è già definito. Se l'applicazione utilizza un altro canale, crearlo immettendo il seguente comando MQSC:

```
DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Channel for use by sample programs')
```

channel - name è il nome del tuo canale.

4. Creare una regola di autenticazione di canale consentendo solo all'indirizzo IP del sistema client di utilizzare il canale immettendo il comando MQSC:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

dove

channel - name è il nome del tuo canale.
client - machine - IP - address è l'indirizzo IP del sistema client. Se l'applicazione client di esempio è in esecuzione sulla stessa macchina del gestore code, utilizzare l'indirizzo IP '127.0.0.1' se l'applicazione si connetterà utilizzando 'localhost'. Se diverse macchine client si collegheranno, è possibile utilizzare un modello o un intervallo invece di un singolo indirizzo IP. Consultare Indirizzi IP generici per i dettagli.
non - privileged - user - id è l'ID utente ottenuto nel passo “1” a pagina 1169

5. Se l'applicazione utilizza il SISTEMA SYSTEM.DEFAULT.LOCAL.QUEUE, questa coda è già definita. Se l'applicazione utilizza un'altra coda, crearla emettendo il comando MQSC:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

dove *nome - coda* è il nome della coda.

6. Concedere l'accesso per connettersi e interrogare il gestore code:
   a) Assicurarsi che l'iniziatore del canale sia avviato. In caso contrario, avviare l'iniziatore di canali emettendo il comando START CHINIT.
   b) Avviare un listener TCP, ad esempio immettere il seguente comando:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

dove *nnnn* è il numero di porta scelto.

7. Se l'applicazione è un'applicazione point - to - point, ovvero utilizza le code, concede l'accesso per consentire l'interrogazione e l'inserimento e il richiamo dei messaggi utilizzando la coda dall'ID utente da utilizzare, immettendo i comandi MQSC:

Immettere i comandi RACF :

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

dove

qmgr - name è il nome del tuo gestore code
queue - name è il nome della coda.
non - privileged - user - id è l'ID utente ottenuto nel passo “1” a pagina 1169

8. Se l'applicazione è un'applicazione di pubblicazione / sottoscrizione, ovvero utilizza argomenti, concedere l'accesso per consentire la pubblicazione e la sottoscrizione utilizzando l'argomento dall'ID utente da utilizzare, immettendo i seguenti comandi RACF :

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)
```

```
PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

dove

qmgr - name è il nome del tuo gestore code

non - privileged - user - id è l'ID utente ottenuto nel passo “1” a pagina 1169

Ciò fornirà a *non - privileged - user - id* l'accesso a qualsiasi argomento nella struttura ad albero degli argomenti, in alternativa, è possibile definire un oggetto argomento utilizzando **DEFINE TOPIC** e concedere gli accessi solo alla parte della struttura ad albero degli argomenti a cui fa riferimento tale oggetto argomento. Per ulteriori informazioni, consultare Controllo dell'accesso utente agli argomenti.

Operazioni successive

L'applicazione client può ora connettersi al gestore code e inserire o richiamare i messaggi utilizzando la coda.

Concetti correlati

z/OS Autorizzazione per gestire gli oggetti IBM MQ su z/OS

Riferimenti correlati

SET CHLAUTH

Definire il canale

DEFINE QLOCAL

SET AUTHREC

z/OS *Preparazione ed esecuzione di applicazioni di esempio per l'ambiente batch su z/OS*

Per preparare un'applicazione di esempio che viene eseguita nell'ambiente batch, eseguire le stesse operazioni che si eseguirebbero quando si crea un'applicazione IBM MQ for z/OS batch.

Questi passi sono elencati in “Creazione di applicazioni batch z/OS” a pagina 1026.

In alternativa, quando si fornisce un formato eseguibile di un esempio, è possibile eseguirlo dalla libreria di caricamento thlqual.SCSQLOAD .

Nota: La versione in linguaggio assembler dell'esempio Sfoglia utilizza DCB (data control block), quindi è necessario modificarlo tramite link utilizzando RMODE(24).

I membri della libreria da utilizzare sono elencati in Tabella 172 a pagina 1172, Tabella 173 a pagina 1172, Tabella 174 a pagina 1173e Tabella 175 a pagina 1173.

È necessario modificare il JCL di esecuzione fornito per gli esempi che si desidera utilizzare (consultare Tabella 172 a pagina 1172, Tabella 173 a pagina 1172, Tabella 174 a pagina 1173e Tabella 175 a pagina 1173).

L'istruzione PARM nel JCL fornito contiene un certo numero di parametri che è necessario modificare. Per eseguire i programmi di esempio C, separare i parametri per spazi; per eseguire i programmi di esempio assembler, COBOL e PL/I, separarli con virgole. Ad esempio, se il nome del gestore code è CSQ1 e si desidera eseguire l'applicazione con una coda denominata LOCALQ1, in COBOL, PL/I e in JCL in linguaggio assembler, l'istruzione PARM deve essere simile alla seguente:

```
PARM=(CSQ1,LOCALQ1)
```

Nel linguaggio C JCL, l'istruzione PARM dovrebbe essere simile alla seguente:

```
PARM=('CSQ1 LOCALQ1')
```

Si è ora pronti a inoltrare i lavori.

z/OS *Nomi delle applicazioni batch di esempio su z/OS*

Un riepilogo dei programmi forniti per le applicazioni batch di esempio.

I programmi di applicazione batch vengono riepilogati nelle seguenti tabelle:

- Esempi Tabella 172 a pagina 1172 Put e Get
- Tabella 173 a pagina 1172 Sfoglia esempio
- Tabella 174 a pagina 1173 Esempio di messaggio di stampa
- Tabella 175 a pagina 1173 Esempi di pubblicazione / sottoscrizione
- Tabella 176 a pagina 1174 Altri esempi

Tabella 172. Campioni di batch Put e Get

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria	File eseguibile fornito nella libreria
CSQ4BCJ1	C	Ottieni programma di origine	SCSQC37S	SSQLOAD
CSQ4BCK1	C	Programma origine di inserimento	SCSQC37S	SSQLOAD
CSQ4BCJR	C	Esempio di esecuzione JCL per CSQ4BCJ1 e CSQBCK1	PROCSCQ	Nessuna
CSQ4BVJ1	COBOL	Ottieni programma di origine	SCSQCOBS	SSQLOAD
CSQ4BVK1	COBOL	Programma origine di inserimento	SCSQCOBS	SSQLOAD
CSQ4BVJR	COBOL	JCL di esecuzione di esempio per CSQBVJ1 e CSQBVK1	PROCSCQ	Nessuna

Tabella 173. Esempio di ricerca batch

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria	File eseguibile fornito nella libreria
CSQ4BVA1	COBOL	Programma di origine	SCSQCOBS	SSQLOAD
CSQ4BVAR	COBOL	JCL di esecuzione di esempio per CSQ4BVA1	PROCSCQ	Nessuna
CSQ4BAA1	Assembler	Programma di origine	SSQASM	SSQLOAD
CSQ4BAAR	Assembler	JCL di esecuzione di esempio per CSQ4BAA1	PROCSCQ	Nessuna
CSQ4BCA1	C	Programma di origine	SCSQC37S	SSQLOAD

Tabella 173. Esempio di ricerca batch (Continua)

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria	File eseguibile fornito nella libreria
CSQ4BCAR	C	JCL di esecuzione di esempio per CSQ4BCA1	PROCSCQ	Nessuna
CSQ4BPA1	PL/I	Programma di origine	SCSQPLIS	SSQLOAD
CSQ4BPAR	PL/I	JCL di esecuzione di esempio per CSQ4BPA1	PROCSCQ	Nessuna

Tabella 174. Esempio di messaggio di stampa batch (solo linguaggio C)

Nome membro	Descrizione	File di origine fornito nella libreria	File eseguibile fornito nella libreria
CSQ4BCG1	Programma di origine	SCSQC37S	SSQLOAD
CSQ4BCGR	JCL di esempio per CSQ4BCG1	PROCSCQ	Nessuna
CSQ4BCL1	Sfoglia programma origine	SCSQC37S	SSQLOAD
CSQ4BCLR	JCL di esecuzione di esempio per CSQ4BCL1	PROCSCQ	Nessuna

Tabella 175. Esempi di pubblicazione / sottoscrizione

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria	JCL in SCSQPROC	File eseguibile fornito nella libreria
CSQ4BCP1	C	Pubblica su programma di origine argomento	SCSQC37S	CSQ4BCPP	SSQLOAD
CSQ4BCP2	C	Sottoscrivi l'argomento e ottieni il programma di origine dei messaggi	SCSQC37S	CSQ4BCPS	SSQLOAD
CSQ4BCP3	C	Sottoscrivi l'argomento utilizzando una destinazione fornita dall'utente e ottieni il programma di origine dei messaggi	SCSQC37S	CSQ4BCPD	SSQLOAD
CSQ4BCP4	C	Sottoscrivi all'argomento utilizzando le opzioni estese e ottieni il programma di origine dei messaggi	SCSQC37S	CSQ4BCPE	SSQLOAD

Tabella 175. Esempi di pubblicazione / sottoscrizione (Continua)					
Nome membro	**Per lingua**	**Descrizione**	**File di origine fornito nella libreria**	**JCL in SCSQPROC**	**File eseguibile fornito nella libreria**
CSQ4BVP1	COBOL	Pubblica su programma di origine argomento	SCSQCOBS	CSQ4BVPP	SSQLOAD
CSQ4BVP2	COBOL	Sottoscrivi l'argomento e ottieni il programma di origine dei messaggi	SCSQCOBS	CSQ4BVPS	SSQLOAD

Tabella 176. Altri campioni					
Nome membro	**Per lingua**	**Descrizione**	**File di origine fornito nella libreria**	**JCL in SCSQPROC**	**File eseguibile fornito nella libreria**
CSQ4BCS1	C	Programma origine di consumo asincrono	SCSQC37S	CSQ4BCSC	SSQLOAD
CSQ4BCS2	C	Programma origine di inserimento e controllo asincrono	SCSQC37S	CSQ4BCSP	SSQLOAD
CSQ4BCM1	C	Interroga programma di origine delle proprietà del messaggio	SCSQC37S	CSQ4BCMP	SSQLOAD
CSQ4BCM2	C	Programma di origine Imposta proprietà messaggio	SCSQC37S	CSQ4BCMP	SSQLOAD

z/OS *Preparazione di applicazioni di esempio per l'ambiente TSO su z/OS*

Per preparare un'applicazione di esempio che viene eseguita nell'ambiente TSO, eseguire le stesse operazioni che si eseguirebbero quando si crea un'applicazione IBM MQ for z/OS batch.

Questi passi sono elencati in “Creazione di applicazioni batch z/OS” a pagina 1026. I membri della libreria da utilizzare sono elencati in Tabella 177 a pagina 1175.

In alternativa, quando si fornisce un formato eseguibile di un esempio, è possibile eseguirlo dalla libreria di caricamento thlqual.SCSQLOAD .

Per l'applicazione di esempio di Mail Manager, assicurarsi che le code utilizzate siano disponibili nel sistema. Sono definiti nel membro **thlqual**.SCSQPROC (CSQ4CVD). Per garantire che queste code siano sempre disponibili, è possibile aggiungere questi membri al dataset di input di inizializzazione CSQINP2 oppure utilizzare il programma CSQUTIL per caricare queste definizioni di coda.

z/OS *Nomi delle applicazioni TSO di esempio su z/OS*

Informazioni sui nomi dei programmi forniti per ciascuna delle applicazioni TSO di esempio e sulle librerie in cui risiedono l'origine, JCL e, solo per l'esempio Gestore messaggi, i file eseguibili.

I programmi applicativi TSO sono riepilogati nelle tabelle seguenti:

- Esempio Gestore posta Tabella 177 a pagina 1175
- Tabella 178 a pagina 1176 Esempio di gestore messaggi

Questi esempi utilizzano pannelli ISPF . È pertanto necessario includere lo stub ISPF , ISPLINK, quando si collegano - modificano i programmi.

Tabella 177. Esempio di TSO Mail Manager

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria
CSQ4CVD	indipendente	Definizioni di oggetti IBM MQ for z/OS	PROCSCQ
CSQ40	indipendente	Messaggi ISPF	SSQMSGE
CSQ4RVD1	COBOL	CLIST per avviare CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Programma di origine per il programma Menu	SCSQCOBS
CSQ4TVD2	COBOL	Programma di origine per il programma Get Mail	SCSQCOBS
CSQ4TVD4	COBOL	Programma di origine per il programma di invio posta	SCSQCOBS
CSQ4TVD5	COBOL	Programma di origine per il programma Nickname	SCSQCOBS
CSQ4VDP1-6	COBOL	Definizioni di pannello	SCSQPNLA
CSQ4VD0	COBOL	Definizione dati	SCSQCOBC
CSQ4VD1	COBOL	Definizione dati	SCSQCOBC
CSQ4VD2	COBOL	Definizione dati	SCSQCOBC
CSQ4VD4	COBOL	Definizione dati	SCSQCOBC
CSQ4RCD1	C	CLIST per avviare CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Programma di origine per il programma Menu	SCSQC37S
CSQ4TCD2	C	Programma di origine per il programma Get Mail	SCSQC37S
CSQ4TCD4	C	Programma di origine per il programma di invio posta	SCSQC37S
CSQ4TCD5	C	Programma di origine per il programma Nickname	SCSQC37S
CSQ4CDP1-6	C	Definizioni di pannello	SCSQPNLA
CSQ4TC0	C	Includi file	SCSQC370

Tabella 178. Esempio gestore messaggi TSO				
Nome membro	**Per lingua**	**Descrizione**	**File di origine fornito nella libreria**	**File eseguibile fornito nella libreria**
CSQ4TCH0	C	Definizione dati	SCSQC370	Nessuna
CSQ4TCH1	C	Programma di origine	SCSQC37S	SSQLOAD
CSQ4TCH2	C	Programma di origine	SCSQC37S	SSQLOAD
CSQ4TCH3	C	Programma di origine	SCSQC37S	SSQLOAD
CSQ4RCH1	C e COBOL	CLIST per avviare CSQ4TCH1 o CSQ4TVH1	SCSQCLST	Nessuna
CSQ4CHP1	C e COBOL	Definizione pannello	SCSQPNLA	Nessuna
CSQ4CHP2	C e COBOL	Definizione pannello	SCSQPNLA	Nessuna
CSQ4CHP3	C e COBOL	Definizione pannello	SCSQPNLA	Nessuna
CSQ4CHP9	C e COBOL	Definizione pannello	SCSQPNLA	Nessuna
CSQ4TVH0	COBOL	Definizione dati	SCSQCOBC	Nessuna
CSQ4TVH1	COBOL	Programma di origine	SCSQCOBS	SSQLOAD
CSQ4TVH2	COBOL	Programma di origine	SCSQCOBS	SSQLOAD
CSQ4TVH3	COBOL	Programma di origine	SCSQCOBS	SSQLOAD

z/OS *Preparazione delle applicazioni di esempio per l'ambiente CICS su z/OS*

Prima di eseguire i programmi di esempio CICS , accedere a CICS utilizzando un LOGMODE di 32702. Ciò è dovuto al fatto che i programmi di esempio sono stati scritti per utilizzare uno schermo 3270 modalità 2.

Per preparare un'applicazione di esempio eseguita nell'ambiente CICS , effettuare le seguenti operazioni:

1. Creare la mappa della descrizione simbolica e la mappa della schermata fisica per l'esempio assemblando l'origine della definizione della schermata BMS (fornita nella libreria **thlqual**.SCSQMAPS, dove **thlqual** è il qualificatore di alto livello utilizzato dall'installazione). Quando si denominano le mappe, utilizzare il nome dell'origine di definizione schermo BMS (non disponibile per i programmi di esempio Put e Get), ma omettere l'ultimo carattere di tale nome.
2. Eseguire le stesse operazioni che si eseguirebbero durante la creazione di qualsiasi applicazione CICS IBM MQ for z/OS . Questi passi sono elencati in “Creazione di applicazioni CICS in z/OS” a pagina 1029. I membri della libreria da utilizzare sono elencati in Tabella 179 a pagina 1177, Tabella 180 a pagina 1178, Tabella 181 a pagina 1178e Tabella 182 a pagina 1179.

   In alternativa, quando si fornisce un formato eseguibile di un esempio, è possibile eseguirlo dalla libreria di caricamento thlqual.SCSQCICS .

3. Identificare la serie di mappe, programmi e transazioni in CICS aggiornando il data set CSD (CICS system definition). Le definizioni richieste si trovano nel membro **thlqual**.SCSQPROC (CSQ4S100). Per istruzioni su come eseguire questa operazione, consultare la sezione *The CICS-IBM MQ Adapter* nella documentazione del prodotto CICS Transaction Server for z/OS 4.1 all'indirizzo: CICS Transaction Server for z/OS 4.1, The CICS-IBM MQ adapter.

   Nota: Per l'applicazione di esempio Controllo credito, si riceve un messaggio di errore in questa fase se non è stato già creato il dataset VSAM utilizzato dall'esempio.

4. Per le applicazioni di esempio Credit Check e Mail Manager, verificare che le code utilizzate siano disponibili nel sistema. Per l'esempio Controllo del credito, sono definiti nel membro **thlqual**.SCSQPROC (CSQ4CVB) per COBOL e **thlqual**.SCSQPROC (CSQ4CCB) per C. Per l'esempio Gestore posta, sono definiti nel membro **thlqual**.SCSQPROC (CSQ4CVD). Per garantire che queste code siano sempre disponibili, è possibile aggiungere questi membri al dataset di input di inizializzazione CSQINP2 oppure utilizzare il programma CSQUTIL per caricare queste definizioni di coda.

   Per l'applicazione di esempio Attributi coda, è possibile utilizzare una o più code fornite per le altre applicazioni di esempio. In alternativa, è possibile utilizzare le proprie code. Tuttavia, nel formato fornito, questo esempio funziona solo con code con i caratteri CSQ4SAMP nei primi otto byte del nome.

z/OS *Nomi delle applicazioni CICS di esempio su z/OS*

Questo argomento fornisce un riepilogo dei programmi forniti per le applicazioni CICS di esempio.

I programmi di applicazione CICS sono riepiloghi nelle tabelle seguenti:

- Esempi Tabella 179 a pagina 1177 Put e Get
- Tabella 180 a pagina 1178 Esempio Attributi coda
- Esempio Tabella 181 a pagina 1178 Mail Manager (solo COBOL)
- Tabella 182 a pagina 1179 Esempio di controllo del credito
- Tabella 183 a pagina 1180 Esempi di utilizzo e di pubblicazione / sottoscrizione asincroni

Tabella 179. Esempi CICS Put e Get

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria	File eseguibile fornito nella libreria
CSQ4CCK1	C	Programma origine di inserimento	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Ottieni programma di origine	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Ottieni programma di origine	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Programma origine di inserimento	SCSQCOBS	SCSQCICS
CSQ4S100	indipendente	Dataset di definizione del sistema CICS	PROCSCQ	Nessuna

Tabella 180. CICS Esempio Attributi coda

Nome membro	Per lingua	Descrizione	File di origine fornito nella libreria	File eseguibile fornito nella libreria
CSQ4CVC1	COBOL	Programma di origine	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Definizione del messaggio	SCSQCOBC	Nessuna
CSQ4VCMS	COBOL	Definizione di schermo BMS	SCSQMAPS	SCSQCICS (denominato CSQ4ACM)
CSQ4CAC1	Assembler	Programma di origine	SSQASM	SCSQCICS
CSQ4AMSG	Assembler	Definizione del messaggio	SCSQMACS	Nessuna
CSQ4ACMS	Assembler	Definizione di schermo BMS	SCSQMAPS	SCSQCICS (denominato CSQ4ACM)
CSQ4CCC1	C	Programma di origine	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Definizione del messaggio	SCSQC370	Nessuna
CSQ4CCMS	C	Definizione di schermo BMS	SCSQMAPS	SCSQCICS (denominato CSQ4ACM)
CSQ4S100	indipendente	Dataset di definizione del sistema CICS	PROCSCQ	Nessuna

Tabella 181. Esempio CICS Mail Manager (solo COBOL)

Nome membro	Descrizione	File di origine fornito nella libreria
CSQ4CVD	Definizioni di oggetti IBM MQ for z/OS	PROCSCQ
CSQ4CVD1	Origine per il programma Menu	SCSQCOBS
CSQ4CVD2	Origine per il programma Get Mail	SCSQCOBS
CSQ4CVD3	Origine per il programma di visualizzazione messaggi	SCSQCOBS
CSQ4CVD4	Origine per il programma di invio posta	SCSQCOBS
CSQ4CVD5	Origine per il programma Nickname	SCSQCOBS
CSQ4VDMS	Origine definizione schermo BMS	SCSQMAPS
CSQ4S100	Dataset di definizione del sistema CICS	PROCSCQ

Tabella 181. Esempio CICS Mail Manager (solo COBOL) (Continua)		
Nome membro	**Descrizione**	**File di origine fornito nella libreria**
CSQ4VD0	Definizione dati	SCSQCOBC
CSQ4VD3	Definizione dati	SCSQCOBC
CSQ4VD4	Definizione dati	SCSQCOBC

Tabella 182. CICS Esempio di controllo del credito			
Nome membro	**Per lingua**	**Descrizione**	**File di origine fornito nella libreria**
CSQ4CVB	indipendente	Definizioni di oggetti IBM MQ	PROCSCQ
CSQ4CCB	indipendente	Definizioni di oggetti IBM MQ	PROCSCQ
CSQ4CVB1	COBOL	Origine per il programma di interfaccia utente	SCSQCOBS
CSQ4CVB2	COBOL	Origine per il gestore delle applicazioni di credito	SCSQCOBS
CSQ4CVB3	COBOL	Origine per il programma di account di controllo	SCSQCOBS
CSQ4CVB4	COBOL	Origine per il programma di distribuzione	SCSQCOBS
CSQ4CVB5	COBOL	Origine per il programma di query dell'agenzia	SCSQCOBS
CSQ4CCB1	C	Origine per il programma di interfaccia utente	SCSQC37S
CSQ4CCB2	C	Origine per il gestore delle applicazioni di credito	SCSQC37S
CSQ4CCB3	C	Origine per il programma di account di controllo	SCSQC37S
CSQ4CCB4	C	Origine per il programma di distribuzione	SCSQC37S
CSQ4CCB5	C	Origine per il programma di query dell'agenzia	SCSQC37S
CSQ4CB0	C	Includi file	SCSQC370
CSQ4CBMS	C	Origine definizione schermo BMS	SCSQMAPS
CSQ4VBMS	COBOL	Origine definizione schermo BMS	SCSQMAPS
CSQ4VB0	COBOL	Definizione dati	SCSQCOBC
CSQ4VB1	COBOL	Definizione dati	SCSQCOBC
CSQ4VB2	COBOL	Definizione dati	SCSQCOBC
CSQ4VB3	COBOL	Definizione dati	SCSQCOBC
CSQ4VB4	COBOL	Definizione dati	SCSQCOBC

Tabella 182. CICS Esempio di controllo del credito (Continua)			
Nome membro	**Per lingua**	**Descrizione**	**File di origine fornito nella libreria**
CSQ4VB5	COBOL	Definizione dati	SCSQCOBC
CSQ4VB6	COBOL	Definizione dati	SCSQCOBC
CSQ4VB7	COBOL	Definizione dati	SCSQCOBC
CSQ4VB8	COBOL	Definizione dati	SCSQCOBC
CSQ4BAQ	indipendente	Origine per dataset VSAM	PROCSCQ
CSQ4FILE	indipendente	JCL per creare dataset VSAM utilizzato da CSQ4CVB3	PROCSCQ
CSQ4S100	indipendente	Dataset di definizione del sistema CICS	PROCSCQ

Tabella 183. CICS Esempi di utilizzo e di pubblicazione / sottoscrizione asincroni		
Nome membro	**Descrizione**	**File di origine fornito nella libreria**
CSQ4CVCN	Origine per il programma Simple Message Consumption	SCSQCOBS
CSQ4CVCT	Origine per il programma di consumo messaggi di controllo	SCSQCOBS
CSQ4CVEV	Origine per il programma Gestore eventi	SCSQCOBS
CSQ4CVPT	Origine per il programma Message Put Client	SCSQCOBS
CSQ4CVRG	Origine per il programma client di registrazione	SCSQCOBS
CSQ4S100	Dataset Definizione sistema CICS	PROCSCQ

z/OS *Preparazione dell'applicazione di esempio per l'ambiente IMS su z/OS*

Parte dell'applicazione di esempio Credit Check può essere eseguita nell'ambiente IMS .

Per preparare questa parte dell'applicazione da eseguire con l'esempio CICS , eseguire prima la procedura descritta in “Preparazione delle applicazioni di esempio per l'ambiente CICS su z/OS” a pagina 1176.

Quindi, effettuare le seguenti operazioni:

1. Eseguire le stesse operazioni che si eseguirebbero durante la creazione di qualsiasi applicazione IMS IBM MQ for z/OS . Questi passi sono elencati in “Creazione di applicazioni IMS (BMP o MPP)” a pagina 1030. I membri della libreria da utilizzare sono elencati in Tabella 184 a pagina 1181.
2. Identificare il programma di applicazione e il database per IMS. Gli esempi vengono forniti con le istruzioni PSBGEN, DBDGEN, definizione ACB, IMSGEN e IMSDALOC per abilitare questa funzione.
3. Caricare il database CSQ4CA personalizzando ed eseguendo il JCL di esempio fornito a questo scopo (CSQ4ILDB). Questo JCL carica il database con i dati dal file CSQ4BAQ. Aggiornare la control region IMS con un'istruzione DD per il database CSQ4CA.
4. Avviare il programma di controllo come programma BMP (batch message processing) adattando ed eseguendo il JCL di esempio fornito per questo scopo. Questo JCL avvia un programma BMP orientato

al batch. Per eseguire il programma come un programma BMP orientato ai messaggi, rimuovere i caratteri di commento dalla riga nel JCL che contiene l'istruzione IN=.

z/OS *Nomi dell'applicazione IMS di esempio su z/OS*

Queste informazioni forniscono una tabella con l'elenco delle origini e dei JCL forniti per l'applicazione IMS di esempio Controllo credito.

Tabella 184. Origine e JCL per l'esempio Controllo credito IMS (solo C)

Nome membro	Descrizione	Fornito nella libreria
CSQ4CVB	Definizioni di oggetti IBM MQ	PROCSCQ
CSQ4ICB3	Origine per il programma di account di controllo	SCSQC37S
CSQ4ICBL	Origine per il caricamento del database del conto di controllo	SCSQC37S
CSQ4CBI	Definizione dati	SCSQC370
CSQ4PSBL	PSBGEN JCL per il programma di caricamento database	PROCSCQ
CSQ4PSB3	PSBGEN JCL per il programma di controllo account	PROCSCQ
CSQ4DBDS	DBDGEN JCL per il database CSQ4CA	PROCSCQ
CSQ4GIMS	IMSDefinizioni di macro GEN per CSQ4IVB3 e CSQ4CA	PROCSCQ
CSQ4ACBG	Definizione ACB (Application Control Block) per CSQ4IVB3	PROCSCQ
CSQ4BAQ	Origine per il database	PROCSCQ
CSQ4ILDB	JCL di esecuzione di esempio per il lavoro di caricamento database	PROCSCQ
CSQ4ICBR	JCL di esecuzione di esempio per il programma di account di controllo	PROCSCQ
CSQ4DYNA	Definizioni di macro DALOC IMSper il database	PROCSCQ

z/OS *Gli esempi Put su z/OS*

I programmi di esempio Put inserano i messaggi su una coda utilizzando la chiamata MQPUT.

I programmi di origine vengono forniti in C e COBOL negli ambienti batch e CICS (consultare Tabella 172 a pagina 1172 e Tabella 179 a pagina 1177).

Progettazione del campione Put

Il flusso attraverso la logica del programma è:

1. Connettersi al gestore code utilizzando la chiamata MQCONN. Se questa chiamata ha esito negativo, stampare i codici di completamento e di errore e arrestare l'elaborazione.

   Nota: Se si sta eseguendo l'esempio in un ambiente CICS , non è necessario emettere una chiamata MQCONN; in tal caso, restituisce DEF_HCONN. È possibile utilizzare l'handle di connessione MQHC_DEF_HCONN per le seguenti chiamate MQI.

2. Aprire la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_OUTPUT. All'immissione di questa chiamata, il programma utilizza l'handle di connessione restituito nel passo “1” a pagina 1183. Per la struttura descrittore oggetto (MQOD), utilizza i valori predefiniti per tutti i campi tranne il campo nome coda, che viene passato come parametro al programma. Se la chiamata MQOPEN non riesce, stampare i codici di completamento e motivo e arrestare l'elaborazione.
3. Creare un loop all'interno del programma che emette chiamate MQPUT fino a quando il numero richiesto di messaggi non viene inserito nella coda. Se una chiamata MQPUT ha esito negativo, il loop viene abbandonato in anticipo, non vengono tentate ulteriori chiamate MQPUT e vengono restituiti i codici di completamento e motivo.
4. Chiudere la coda utilizzando la chiamata MQCLOSE con l'handle dell'oggetto restituito nel passo “2” a pagina 1183. Se questa chiamata ha esito negativo, stampare i codici di completamento e di errore.
5. Disconnettersi dal gestore code utilizzando la chiamata MQDISC con l'handle di connessione restituito nel passo “1” a pagina 1183. Se questa chiamata ha esito negativo, stampare i codici di completamento e di errore.

   Nota: Se si sta eseguendo l'esempio in un ambiente CICS , non è necessario emettere una chiamata MQDISC.

z/OS *Gli esempi Put per l'ambiente batch su z/OS*

Utilizzare questo argomento quando si considerano gli esempi Put per l'ambiente batch.

Per eseguire gli esempi, modificare ed eseguire il JCL di esempio, come descritto in “Preparazione ed esecuzione di applicazioni di esempio per l'ambiente batch su z/OS” a pagina 1171.

I programmi utilizzano i seguenti parametri in un EXEC PARM, separati da spazi in C e virgole in COBOL:

1. Il nome del gestore code (4 caratteri)
2. Il nome della coda di destinazione (48 caratteri)
3. Il numero di messaggi (fino a 4 cifre)
4. Il carattere di riempimento da scrivere nel messaggio (1 carattere)
5. Il numero di caratteri da scrivere nel messaggio (fino a 4 cifre)
6. La persistenza del messaggio (1 carattere: P per persistente o N per non persistente)

Se si immette uno di questi parametri in modo errato, si ricevono messaggi di errore appropriati.

Tutti i messaggi degli esempi vengono scritti nel dataset SYSPRINT.

Note d'utilizzo

- Per semplificare gli esempi, vi sono alcune differenze funzionali minori tra le versioni linguistiche. Tuttavia, queste differenze vengono ridotte se si utilizza il layout dei parametri mostrati nel JCL di esecuzione di esempio, CSQ4BCJRe CSQ4BVJR. Nessuna delle differenze è relativa alla MQI.
- CSQ4BCK1 consente di immettere più di quattro cifre per il numero di messaggi inviati e la lunghezza dei messaggi.
- Per i due campi numerici, immettere una cifra compresa tra 1 e 9999. Il valore immesso deve essere un numero positivo. Ad esempio, per inserire un singolo messaggio, è possibile immettere 1, 01, 001 o 0001 come valore. Se si immettono valori non numerici o negativi, si potrebbe ricevere un errore. Ad esempio, se si immette -1, il programma COBOL invia un messaggio a 1 byte, ma il programma C riceve un errore.
- Per entrambi i programmi, CSQ4BCK1 e CSQ4BVK1, è necessario immettere P nel parametro di persistenza, + + PER + +, se si desidera che il messaggio sia persistente. In caso contrario, il messaggio non sarà persistente.

z/OS *Gli esempi Put per l'ambiente CICS su z/OS*

Utilizzare questo argomento quando si considerano gli esempi Put per l'ambiente CICS .

Le transazioni assumono i seguenti parametri separati da virgole:

1. Il numero di messaggi (fino a 4 cifre)
2. Il carattere di riempimento da scrivere nel messaggio (1 carattere)
3. Il numero di caratteri da scrivere nel messaggio (fino a 4 cifre)
4. La persistenza del messaggio (1 carattere: P per persistente o N per non persistente)
5. Il nome della coda di destinazione (48 caratteri)

Se si immette uno di questi parametri in modo non corretto, si ricevono messaggi di errore appropriati.

Per l'esempio COBOL, richiamare l'esempio Put nell'ambiente CICS immettendo:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

Per l'esempio C, richiamare l'esempio Put nell'ambiente CICS immettendo:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Tutti i messaggi degli esempi vengono visualizzati sullo schermo.

Note d'utilizzo

- Per semplificare gli esempi, vi sono alcune differenze funzionali minori tra le versioni linguistiche. Nessuna delle differenze è relativa alla MQI.
- Se si immette un nome coda più lungo di 48 caratteri, la sua lunghezza viene troncata al massimo di 48 caratteri ma non viene restituito alcun messaggio di errore.
- Prima di immettere la transazione, premere il tasto CLEAR.
- Per i due campi numerici, immettere un numero compreso tra 1 e 9999. Il valore immesso deve essere un numero positivo. Ad esempio, per inserire un singolo messaggio, è possibile immettere il valore 1, 01, 001 o 0001. Se si immettono valori non numerici o negativi, si potrebbe ricevere un errore. Ad esempio, se si immette -1, il programma COBOL invia un messaggio a 1 byte e il programma C termina in modo anomalo con un errore da malloc ().
- Per entrambi i programmi, CSQ4CCK1 e CSQ4CVK1, immettere P nel parametro di persistenza se si desidera che il messaggio sia persistente. Per i messaggi non persistenti, immettere N nel parametro persistenza. Se si immette un altro valore, si riceve un messaggio di errore.
- I messaggi vengono inseriti nel punto di sincronizzazione perché i valori predefiniti vengono utilizzati per tutti i parametri tranne quelli impostati durante il richiamo del programma.

z/OS *Gli esempi Get su z/OS*

I programmi di esempio Get ricevono i messaggi da una coda utilizzando la chiamata MQGET.

I programmi di origine vengono forniti in C e COBOL negli ambienti batch e CICS (consultare Tabella 172 a pagina 1172 e Tabella 179 a pagina 1177).

z/OS *Progettazione dell'esempio Get su z/OS*

Scopri la progettazione dell'esempio Get e alcune note di utilizzo da considerare.

Il flusso attraverso la logica del programma è:

1. Connettersi al gestore code utilizzando la chiamata MQCONN. Se questa chiamata ha esito negativo, stampare i codici di completamento e di errore e arrestare l'elaborazione.

   Nota: Se si sta eseguendo l'esempio in un ambiente CICS , non è necessario emettere una chiamata MQCONN; in tal caso, restituisce DEF_HCONN. È possibile utilizzare l'handle di connessione MQHC_DEF_HCONN per le seguenti chiamate MQI.
2. Aprire la coda utilizzando la chiamata MQOPEN con le opzioni MQOO_INPUT_SHARED e MQOO_BROWSE. All'immissione di questa chiamata, il programma utilizza l'handle di connessione restituito nel passo “1” a pagina 1183. Per la struttura descrittore oggetto (MQOD), utilizza i valori

predefiniti per tutti i campi tranne il campo nome coda, che viene passato come parametro al programma. Se la chiamata MQOPEN non riesce, stampare i codici di completamento e motivo e arrestare l'elaborazione.

3. Creare un loop all'interno del programma che emette chiamate MQGET fino a quando il numero richiesto di messaggi non viene richiamato dalla coda. Se una chiamata MQGET non riesce, il loop viene abbandonato in anticipo, non vengono tentate ulteriori chiamate MQGET e vengono restituiti i codici di completamento e motivo. Le opzioni riportate di seguito sono specificate nella chiamata MQGET:

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT o MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST e MQGMO_BROWSE_NEXT

   Per una descrizione di queste opzioni, consultare MQGET. Per ogni messaggio, il numero di messaggio viene stampato seguito dalla lunghezza del messaggio e dai relativi dati.
4. Chiudere la coda utilizzando la chiamata MQCLOSE con l'handle dell'oggetto restituito nel passo “2” a pagina 1183. Se questa chiamata ha esito negativo, stampare i codici di completamento e di errore.
5. Disconnettersi dal gestore code utilizzando la chiamata MQDISC con l'handle di connessione restituito nel passo “1” a pagina 1183. Se questa chiamata ha esito negativo, stampare i codici di completamento e di errore.

   Nota: Se si sta eseguendo l'esempio in un ambiente CICS , non è necessario emettere una chiamata MQDISC.

Note d'utilizzo

- Per semplificare gli esempi, vi sono alcune differenze funzionali minori tra le versioni linguistiche. Tuttavia, queste differenze vengono ridotte se si utilizza il layout dei parametri mostrati nel JCL di esecuzione di esempio, CSQ4BCJRe CSQ4BVJR. Nessuna delle differenze è relativa alla MQI.
- CSQ4BCJ1 consente di immettere più di quattro cifre per il numero di messaggi richiamati.
- I messaggi più lunghi di 64 KB vengono troncati.
- CSQ4BCJ1 può visualizzare correttamente solo i messaggi di caratteri perché viene visualizzato solo fino a quando non viene visualizzato il primo carattere NULL (\0).
- Per il campo numero numerico di messaggi, immettere una cifra compresa nell'intervallo tra 1 e 9999. Il valore immesso deve essere un numero positivo. Ad esempio, per ottenere un singolo messaggio, è possibile immettere 1, 01, 001 o 0001 come valore. Se si immettono valori non numerici o negativi, si potrebbe ricevere un errore. Ad esempio, se si immette -1, il programma COBOL richiama un messaggio, ma il programma C non richiama alcun messaggio.
- Per entrambi i programmi, CSQ4BCJ1 e CSQ4BVJ1, immettere B nel parametro get, + + GET + +, se si desidera sfogliare i messaggi.
- Per entrambi i programmi, CSQ4BCJ1 e CSQ4BVJ1, immettere S nel parametro del punto di sincronizzazione, + + SYNC + +, per i messaggi da richiamare nel punto di sincronizzazione.

z/OS *Gli esempi Get per l'ambiente batch su z/OS*

Per eseguire gli esempi, modificare ed eseguire il JCL di esempio, come descritto in “Preparazione ed esecuzione di applicazioni di esempio per l'ambiente batch su z/OS” a pagina 1171.

I programmi utilizzano i seguenti parametri in un EXEC PARM, separati da spazi in C e virgole in COBOL:

1. Il nome del gestore code (4 caratteri)
2. Il nome della coda di destinazione (48 caratteri)
3. Il numero di messaggi da richiamare (fino a 4 cifre)

4. L'opzione sfoglia / ottieni messaggio (1 carattere: B per sfogliare o D per ottenere i messaggi in modo distruttivo)
5. Il controllo del punto di sincronizzazione (1 carattere: S per il punto di sincronizzazione o N per nessun punto di sincronizzazione)

Se si immette uno di questi parametri in modo non corretto, si ricevono messaggi di errore appropriati.

L'output degli esempi viene scritto nel dataset SYSPRINT:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *Gli esempi Get per l'ambiente CICS su z/OS*

Considerazioni speciali per gli esempi Get per l'ambiente CICS .

Le transazioni assumono i parametri seguenti in un EXEC PARM, separati da virgole:

1. Il numero di messaggi da richiamare (fino a quattro cifre)
2. L'opzione sfoglia / ottieni messaggio (un carattere: B per sfogliare o D per ottenere i messaggi in modo distruttivo)
3. Il controllo del punto di sincronizzazione (un carattere: S per il punto di sincronizzazione o N per nessun punto di sincronizzazione)
4. Il nome della coda di destinazione (48 caratteri)

Se si immette uno di questi parametri in modo non corretto, si ricevono messaggi di errore appropriati.

Per l'esempio COBOL, richiamare l'esempio Get nell'ambiente CICS immettendo:

```
MVGT,9999,B,S,QUEUE.NAME
```

Per l'esempio C, richiamare l'esempio Get nell'ambiente CICS immettendo:

```
MCGT,9999,B,S,QUEUE.NAME
```

Quando i messaggi vengono richiamati dalla coda, vengono inseriti in una coda di memoria temporanea CICS con lo stesso nome della transazione CICS (ad esempio, MCGT per l'esempio C).

Di seguito è riportato un esempio degli esempi Get:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE *********************
```

Note d'utilizzo

- Per semplificare gli esempi, vi sono alcune differenze funzionali minori tra le versioni linguistiche. Nessuna delle differenze è relativa alla MQI.

- Se si immette un nome coda più lungo di 48 caratteri, la sua lunghezza viene troncata al massimo di 48 caratteri ma non viene restituito alcun messaggio di errore.
- Prima di immettere la transazione, premere il tasto CLEAR.
- CSQ4CCJ1 può visualizzare correttamente solo i messaggi di caratteri poiché viene visualizzato solo fino a quando non viene visualizzato il primo carattere NULL (\0).
- Per il campo numerico, immettere un qualsiasi numero compreso tra 1 e 9999. Il valore immesso deve essere un numero positivo. Ad esempio, per ottenere un singolo messaggio, è possibile immettere il valore 1, 01, 001 o 0001. Se si immette un valore non numerico o negativo, si potrebbe ricevere un errore.
- I messaggi più lunghi di 24 526 byte in C e 9 950 byte in COBOL vengono troncati. Ciò è dovuto al modo in cui vengono utilizzate le code di storage temporaneo CICS .
- Per entrambi i programmi, CSQ4CCK1 e CSQ4CVK1, immettere B nel parametro get se si desidera esaminare i messaggi, altrimenti immettere D. Questo esegue chiamate MQGET distruttive. Se si immette un altro valore, si riceve un messaggio di errore.
- Per entrambi i programmi, CSQ4CCJ1 e CSQ4CVJ1, immettere S nel parametro del punto di sincronizzazione per richiamare i messaggi nel punto di sincronizzazione. Se si immette N nel parametro del punto di sincronizzazione, le chiamate MQGET vengono emesse fuori dal punto di sincronizzazione. Se si immette un altro valore, si riceve un messaggio di errore.

z/OS *L'esempio Sfoglia su z/OS*

L'esempio Sfoglia è un'applicazione batch che illustra come sfogliare i messaggi su una coda utilizzando la chiamata MQGET.

L'applicazione passa attraverso tutti i messaggi in una coda, stampando i primi 80 byte di ciascuno. È possibile utilizzare questa applicazione per esaminare i messaggi in coda senza modificarli.

I programmi di origine e il JCL di esempio vengono forniti nei linguaggi COBOL, assembler, PL/I e C (consultare Tabella 173 a pagina 1172).

Per avviare l'applicazione, modificare ed eseguire il JCL di esecuzione di esempio, come descritto in “Preparazione ed esecuzione di applicazioni di esempio per l'ambiente batch su z/OS” a pagina 1171. È possibile esaminare i messaggi su una delle code specificando il nome della coda nel JCL di esecuzione.

Quando si esegue l'applicazione (e sono presenti alcuni messaggi sulla coda), il dataset di output è il seguente:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

Se non ci sono messaggi nella coda, il dataset contiene solo le intestazioni e il messaggio `Fine del report` . Se si verifica un errore con una delle chiamate MQI, i codici di completamento e di motivo vengono aggiunti al dataset di output.

z/OS *Progettazione dell'esempio Sfoglia su z/OS*

L'applicazione di esempio Sfoglia utilizza un singolo modulo di programma; uno viene fornito in ciascuno dei linguaggi di programmazione supportati.

Il flusso attraverso la logica del programma è:

1. Aprire un dataset di stampa e stampare la riga del titolo del report. Verificare che i nomi del gestore code e della coda siano stati passati dal JCL di esecuzione. Se sono stati passati entrambi i nomi, stampare le righe del report che contengono i nomi. In caso contrario, stampare un messaggio di errore, chiudere la serie di dati di stampa e arrestare l'elaborazione.

   Il modo in cui il programma verifica i parametri trasmessi dal JCL dipende dal linguaggio in cui è scritto il programma; per ulteriori informazioni, consultare "Considerazioni sulla progettazione in base alla lingua su z/OS" a pagina 1188.
2. Connettersi al gestore code utilizzando la chiamata MQCONN. Se questa chiamata non riesce, stampare i codici di completamento e di errore, chiudere la serie di dati di stampa e arrestare l'elaborazione.
3. Aprire la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_BROWSE. In fase di immissione di questa chiamata, il programma utilizza l'handle di collegamento restituito nel passo "2" a pagina 1187. Per MQOD (object descriptor structure), utilizza i valori predefiniti per tutti i campi tranne il nome della coda (che è stato passato nel passo "1" a pagina 1187). Se questa chiamata non riesce, stampare i codici di completamento e di errore, chiudere la serie di dati di stampa e arrestare l'elaborazione.
4. Sfoglia il primo messaggio sulla coda, utilizzando la chiamata MQGET. All'immissione di questa chiamata, il programma specifica:
   - Le connessioni e gli handle di coda dai passi "2" a pagina 1187 e "3" a pagina 1187
   - Una struttura MQMD con tutti i campi impostati sui loro valori iniziali
   - Due opzioni:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - Un buffer di dimensione 80 byte per contenere i dati copiati dal messaggio

   L'opzione MQGMO_ACCEPT_TRUNCATED_MSG consente il completamento della chiamata anche se il messaggio è più lungo del buffer di 80 byte specificato nella chiamata. Se il messaggio è più lungo del buffer, il messaggio viene troncato per adattarsi al buffer e i codici di errore e di completamento sono impostati per visualizzarlo. L'esempio è stato progettato in modo che i messaggi vengano troncati a 80 caratteri per semplificare la lettura del report. La dimensione del buffer è impostata da un'istruzione `DEFINE` , quindi è possibile modificarla facilmente se si desidera.
5. Eseguire il seguente loop finché la chiamata MQGET non ha esito negativo:
   a. Stampa una riga del report che mostra:
      - Il numero di sequenza del messaggio (questo è un conteggio delle operazioni di ricerca).
      - La lunghezza reale del messaggio (non la lunghezza troncata). Questo valore viene restituito nel campo `DataLength` della chiamata MQGET.
      - I primi 80 byte dei dati del messaggio.

   b. Reimpostare i campi `MsqId` e `CorrelId` della struttura MQMD su valori null
   c. Sfoglia il messaggio successivo, utilizzando la chiamata MQGET con queste due opzioni:
      - MQGMO_BROWSE_SUCCESSIVO
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. Se la chiamata MQGET ha esito negativo, verificare il codice di errore per verificare se la chiamata ha avuto esito negativo perché il cursore di ricerca è arrivato alla fine della coda. In questo caso, stampare il messaggio `Fine del prospetto` e andare alla fase "7" a pagina 1187 ; altrimenti, stampare i codici di completamento e di errore, chiudere la serie di dati di stampa e arrestare l'elaborazione.
7. Chiudere la coda utilizzando la chiamata MQCLOSE con l'handle dell'oggetto restituito nel passo "3" a pagina 1187.

8. Disconnettersi dal gestore code utilizzando la chiamata MQDISC con l'handle di connessione restituito nel passo “2” a pagina 1187.
9. Chiudere il dataset di stampa e arrestare l'elaborazione.

z/OS *Considerazioni sulla progettazione in base alla lingua su z/OS*

I moduli di origine vengono forniti per l'esempio Sfoglia in quattro linguaggi di programmazione.

Esistono due differenze principali tra i moduli di origine:

- Quando si verificano i parametri passati dal JCL di esecuzione, i moduli COBOL, PL/I e linguaggio assembler ricercano il carattere virgola (,). Se il JCL passa `PARM=(,LOCALQ1)`, l'applicazione tenta di aprire la coda LOCALQ1 sul gestore code predefinito. Se non c'è un nome dopo la virgola (o nessuna virgola), l'applicazione restituisce un errore. Il modulo C non ricerca il carattere virgola. Se il JCL passa un singolo parametro (ad esempio, `PARM=('LOCALQ1')`), il modulo C lo utilizza come nome coda sul gestore code predefinito.
- Per mantenere semplice il modulo in linguaggio assembler, utilizza il formato data *aa/ggg* (ad esempio, 05/116) quando crea il prospetto di stampa. Gli altri moduli utilizzano la data del calendario in formato *mm/gg/aa* .

z/OS *L'esempio Stampa messaggio su z/OS*

L'esempio Stampa messaggio è un'applicazione batch che mostra come rimuovere tutti i messaggi da una coda utilizzando la chiamata MQGET.

L'esempio Stampa messaggio utilizza tre parametri:

1. Il nome del gestore code
2. Il nome della coda di origine
3. Un parametro facoltativo per le proprietà

Stampa inoltre, per ogni messaggio, i campi del descrittore del messaggio, seguiti dai dati del messaggio. Il programma stampa i dati sia in esadecimale che come caratteri (se stampabili). Se un carattere non è stampabile, il programma lo sostituisce con un carattere punto (.). È possibile utilizzare il programma durante la diagnosi dei problemi con un'applicazione che sta inserendo messaggi in una coda.

I valori consentiti per il parametro di proprietà sono:

Tabella 185. Valori consentiti per il parametro della proprietà

Valore	Comportamento
0	Comportamento predefinito. Le proprietà che vengono consegnate all'applicazione dipendono dall'attributo coda **PropertyControl** da cui viene richiamato il messaggio.

1	Un handle del messaggio viene creato e utilizzato con MQGET. Le proprietà del messaggio, eccetto quelle contenute nel descrittore del messaggio (o estensione), vengono visualizzate in modo simile al descrittore del messaggio. Ad esempio:
	`****Message properties****`
	`  property name: property value`
	Oppure, se non sono disponibili proprietà:
	`****Message properties****`
	`  None`
	I valori numerici vengono visualizzati utilizzando `printf`, i valori stringa vengono racchiusi tra virgolette singole e le stringhe di byte vengono racchiuse tra virgolette singole e X, come per il descrittore del messaggio.

Tabella 185. Valori consentiti per il parametro della proprietà (Continua)	
Valore	**Comportamento**
2	Viene specificato MQGMO_NO_PROPERTIES, in modo che vengano restituite solo le proprietà del descrizione del messaggio.
3	Viene specificato MQGMO_PROPERTIES_FORCE_MQRFH2 , in modo che tutte le proprietà vengano restituite nei dati del messaggio.
4	MQGMO_PROPERTIES_COMPATIBILITY è specificato, in modo che tutte le proprietà possano essere restituite a seconda che sia inclusa una proprietà IBM MQ , altrimenti le proprietà vengono eliminate.

È possibile modificare l'applicazione in modo che esplora i messaggi, piuttosto che rimuoverli dalla coda. A tale scopo, compilare con l'opzione -DBROWSE, per definire la macro BROWSE, come indicato in “Progettazione dell'esempio di messaggio di stampa su z/OS” a pagina 1190. Il codice eseguibile viene fornito per l'utente nella libreria SCSQLOAD. Il modulo CSQ4BCG0 viene creato con -DBROWSE; il modulo CSQ4BCG1 legge in modo distruttivo la coda.

L'applicazione ha un singolo programma di origine, che è scritto in linguaggio C. Viene fornito anche il codice JCL di esecuzione di esempio (consultare Tabella 174 a pagina 1173).

Per avviare l'applicazione, modificare ed eseguire il JCL di esecuzione di esempio, come descritto in “Preparazione ed esecuzione di applicazioni di esempio per l'ambiente batch su z/OS” a pagina 1171. Quando si esegue l'applicazione (e sono presenti alcuni messaggi sulla coda), il dataset di output è simile a quello in Figura 139 a pagina 1190.

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR  '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                 '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '  '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

Figura 139. Esempio di report dall'applicazione di esempio Stampa messaggio

z/OS *Progettazione dell'esempio di messaggio di stampa su z/OS*

L'applicazione di esempio Print Message utilizza un singolo programma scritto in linguaggio C.

Il flusso attraverso la logica del programma è:

1. Verificare che i nomi del gestore code e della coda siano stati passati dal JCL di esecuzione. In caso contrario, stampare un messaggio di errore e arrestare l'elaborazione.
2. Connettersi al gestore code utilizzando la chiamata MQCONN. Se questa chiamata non ha esito positivo, stampare i codici di completamento e motivo e arrestare l'elaborazione; altrimenti, stampare il nome del gestore code.
3. Aprire la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_INPUT_SHARED.

   Nota: Se si desidera che l'applicazione ricerchi i messaggi anziché rimuoverli dalla coda, compilare l'esempio con -DBROWSE oppure aggiungere `#define BROWSE` all'inizio dell'origine. Quando si esegue questa operazione, il preprocessore della macro aggiunge la riga nel programma che seleziona l'opzione MQOO_BROWSE nella compilazione.

In fase di immissione di questa chiamata, il programma utilizza l'handle di collegamento restituito nel passo “2” a pagina 1190. Per MQOD (object descriptor structure), utilizza i valori predefiniti per tutti i campi tranne il nome della coda (che è stato passato nel passo “1” a pagina 1190). Se questa chiamata non ha esito positivo, stampare i codici di errore e di completamento e arrestare l'elaborazione; altrimenti, stampare il nome della coda.

4. Se si utilizza un handle del messaggio per ottenere le proprietà del messaggio, utilizzare MQCRTMH per creare tale handle da utilizzare con le successive chiamate MQGET. Se questa chiamata non ha esito positivo, stampare i codici di errore e di completamento e arrestare l'elaborazione.
5. Impostare le opzioni di richiamo del messaggio per riflettere l'azione di richiesta per le proprietà del messaggio.
6. Eseguire il seguente loop finché la chiamata MQGET non ha esito negativo:
   a. Inizializzare il buffer in spazi vuoti in modo che i dati del messaggio non vengano danneggiati da dati già presenti nel buffer.
   b. Impostare i campi `MsgId` e `CorrelId` della struttura MQMD su valori null in modo che la chiamata MQGET selezioni il primo messaggio dalla coda.
   c. Richiamare un messaggio dalla coda, utilizzando la chiamata MQGET. All'immissione di questa chiamata, il programma specifica:
      - La connessione e l'oggetto vengono gestiti dai passi “2” a pagina 1190 e “3” a pagina 1190.
      - Una struttura MQMD con tutti i campi impostati sui valori iniziali. (`MsgId` e `CorrelId` vengono reimpostati su valori null per ogni chiamata MQGET.)
      - L'opzione MQGMO_NO_WAIT.

        Nota: Se si desidera che l'applicazione ricerchi i messaggi anziché rimuoverli dalla coda, compilare l'esempio con -DBROWSE oppure aggiungere `#define BROWSE` all'inizio dell'origine. Quando si esegue questa operazione, il preprocessore macro aggiunge la riga nel programma che seleziona l'opzione MQGMO_BROWSE_NEXT alla compilazione. Quando questa opzione viene utilizzata in una chiamata su una coda per cui non è stato precedentemente utilizzato alcun cursore di ricerca con l'handle dell'oggetto corrente, il cursore di ricerca viene posizionato logicamente prima del primo messaggio.
      - Un buffer di dimensione 64KB per contenere i dati copiati dal messaggio.
   d. Richiamare la sottoroutine printMD . Questo stampa il nome di ciascun campo nel descrittore del messaggio, seguito dal contenuto.
   e. Se è stato creato un handle del messaggio nel passo “4” a pagina 1191 , richiamare la sottoroutine printProperties per visualizzare le proprietà del messaggio.
   f. Stampare la lunghezza del messaggio seguito dai dati del messaggio. Ogni riga di dati del messaggio è nel formato seguente:
      - Posizione relativa (in esadecimale) di questa parte dei dati
      - 16 byte di dati esadecimali
      - Gli stessi 16 byte di dati in formato carattere, se stampabili (i caratteri non stampabili vengono sostituiti da punti)
7. Se la chiamata MQGET ha esito negativo, verificare il codice motivo per verificare se la chiamata non è riuscita perché non ci sono più messaggi nella coda. In questo caso, stampare il messaggio: Non più messaggi; altrimenti, stampare i codici di completamento e di errore. In entrambi i casi, andare alla fase “9” a pagina 1192.

   Nota: La chiamata MQGET ha esito negativo se trova un messaggio con più di 64KB di dati. Per modificare il programma in modo da gestire messaggi più grandi, è possibile effettuare una delle seguenti operazioni:

   - Aggiungere l'opzione MQGMO_ACCEPT_TRUNCATED_MSG alla chiamata MQGET, in modo che la chiamata riceva i primi 64KB di dati ed elimini il resto
   - Fare in modo che il programma lasci il messaggio sulla coda quando ne trova uno con questa quantità di dati

- Aumentare la dimensione del buffer

8. Se è stato creato un gestore messaggi nel passo “4” a pagina 1191 , richiamare MQDLTMH per eliminarlo.
9. Chiudere la coda utilizzando la chiamata MQCLOSE con l'handle dell'oggetto restituito nel passo “3” a pagina 1190.
10. Disconnettersi dal gestore code utilizzando la chiamata MQDISC con l'handle di connessione restituito nel passo “2” a pagina 1190.

z/OS *L'esempio Attributi coda su z/OS .*

L'esempio Attributi coda è un'applicazione CICS in modalità conversazione che illustra l'utilizzo delle chiamate MQINQ e MQSET.

Mostra come analizzare i valori degli attributi **InhibitPut** e **InhibitGet** delle code e come modificarli in modo che i programmi non possano inserire o richiamare messaggi da una coda. È possibile che si desideri *bloccare* una coda in questo modo quando si esegue il test di un programma.

Per evitare interferenze accidentali con le proprie code, questo esempio funziona solo su un oggetto coda con i caratteri CSQ4SAMP nei primi otto byte del nome. Tuttavia, il codice sorgente include commenti per mostrare come rimuovere questa restrizione.

I programmi di origine vengono forniti nei linguaggi COBOL, assembler e C (consultare Tabella 180 a pagina 1178).

La versione in linguaggio assembler dell'esempio utilizza il codice riutilizzabile. Per effettuare questa operazione, si noterà che il codice per ogni chiamata MQI in quella versione dell'esempio include la parola chiave MF; ad esempio:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(La parola chiave VL indica che è possibile utilizzare la transazione fornita da CEDF (CICS Execution Diagnostic Facility) per il debug del programma.) Per ulteriori informazioni sulla scrittura di programmi riattivabili, consultare Coding in System/390 assembler language.

Per avviare l'applicazione, avviare il sistema CICS e utilizzare le seguenti transazioni CICS :

- Per COBOL, MVC1
- Per il linguaggio assembler, MAC1
- Per C, MCC1

È possibile modificare il nome di una di queste transazioni modificando il dataset CSD menzionato nel passo 3.

Progettazione del campione

Quando si avvia l'esempio, viene visualizzata una mappa dello schermo che contiene campi per:

- Il nome della coda
- Richiesta utente (le azioni valide sono: inquire, allow o inhibit)
- Stato corrente delle operazioni di inserimento per la coda
- Stato corrente delle operazioni di acquisizione per la coda

I primi due campi sono per l'input dell'utente. Gli ultimi due campi vengono compilati dall'applicazione: mostrano la parola `INHIBITED` o `ALLOWED`.

L'applicazione convalida i valori immessi nei primi due campi. Controlla che il nome della coda inizi con i caratteri CSQ4SAMP e che sia stata immessa una delle tre richieste valide nel campo Azione. L'applicazione converte tutto l'input in maiuscolo, quindi non è possibile utilizzare code con nomi che contengono caratteri minuscoli.

Se si immette `inquire` nel campo **Azione** , il flusso attraverso la logica del programma è:

1. Aprire la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_INQUIRE
2. Richiamare l'MQINQ utilizzando i selettori MQIA_INIB_GET e MQIA_INIB_PUT
3. Chiudere la coda utilizzando la chiamata MQCLOSE
4. Analizzare gli attributi che vengono restituiti nel parametro **IntAttrs** della chiamata MQINQ e spostare le parole INIBITED o ALLOWED, a seconda dei casi, nei campi della schermata pertinenti

Se si immette `inhibit` nel campo **Azione** , il flusso attraverso la logica del programma è:

1. Aprire la coda utilizzando la chiamata MQOPEN con l'opzione MQOO_SET
2. Richiamare MQSET utilizzando i selettori MQIA_INIB_GET e MQIA_INIB_PUT e con i valori MQQA_GET_INIITED e MQQA_PUT_INIITED nel parametro **IntAttrs**
3. Chiudere la coda utilizzando la chiamata MQCLOSE
4. Spostare la parola INIBITA nei campi dello schermo pertinenti

Se si immette `allow` nel campo **Azione** , l'applicazione esegue un'elaborazione simile a quella per una richiesta di inibizione. Le uniche differenze sono le impostazioni degli attributi e le parole visualizzate sullo schermo.

Quando l'applicazione apre la coda, utilizza l'handle di connessione predefinito al gestore code. (CICS stabilisce una connessione al gestore code quando si avvia il sistema CICS .) L'applicazione può eseguire il trap dei seguenti errori in questa fase:

- L'applicazione non è connessa al gestore code
- La coda non esiste
- L'utente non è autorizzato ad accedere alla coda
- L'applicazione non è autorizzata ad aprire la coda

Per altri errori MQI, l'applicazione visualizza i codici di completamento e motivo.

z/OS *L'esempio Gestore posta su z/OS*

L'applicazione di esempio Mail Manager è una suite di programmi che illustra l'invio e la ricezione di messaggi, sia in un singolo ambiente che in ambienti differenti. L'applicazione è un semplice sistema di posta elettronica che consente agli utenti di scambiare messaggi, anche se utilizzano gestori code diversi.

L'applicazione illustra come creare le code utilizzando la chiamata MQOPEN e inserendo i comandi IBM MQ for z/OS nella coda di input del sistema.

Vengono fornite tre versioni dell'applicazione:

- Un'applicazione CICS scritta in COBOL
- Un'applicazione TSO scritta in COBOL
- Un'applicazione TSO scritta in C

z/OS *Preparazione dell'esempio Gestore posta su z/OS*

Il Gestore posta viene fornito nelle versioni eseguite in due ambienti. La preparazione da eseguire prima di eseguire l'applicazione dipende dall'ambiente che si desidera utilizzare.

Gli utenti possono accedere alle code di posta e alle code di nickname sia da TSO che da CICS purché i loro ID utente di collegamento siano gli stessi su ogni sistema.

Prima di poter inviare messaggi a un altro gestore code, è necessario configurare un canale di messaggi per tale gestore code. Per effettuare questa operazione, utilizzare la funzione di controllo del canale di IBM MQ, descritta in Funzione di controllo del canale.

Preparazione dell'esempio per l'ambiente TSO

Eseguire queste operazioni:

1. Preparare l'esempio come descritto in “Preparazione di applicazioni di esempio per l'ambiente TSO su z/OS” a pagina 1174.
2. Adattare il CLIST fornito per l'esempio per definire:
   - L'ubicazione dei pannelli
   - L'ubicazione del file di messaggi
   - L'ubicazione dei moduli di caricamento
   - Il nome del gestore code che si desidera utilizzare con l'applicazione

   Viene fornito un CLIST separato per ciascuna versione di lingua dell'esempio:
   - Per la versione COBOL: CSQ4RVD1
   - Per la versione C: CSQ4RCD1
3. Verificare che le code utilizzate dall'applicazione siano disponibili sul gestore code. Le code sono definite in CSQ4CVD.

Nota: VS COBOL II non supporta il multitasking con ISPF. Ciò significa che non è possibile utilizzare l'applicazione di esempio Gestore posta su entrambi i lati di una schermata suddivisa. In tal caso, i risultati sono imprevedibili.

z/OS *Esecuzione dell'esempio Gestore posta su z/OS*

Per avviare l'esempio nell'ambiente CICS Transaction Server per z/OS , eseguire la transazione MAIL. Se non si è già collegati a CICS, l'applicazione richiede di immettere un ID utente a cui inviare la posta.

Quando si avvia l'applicazione, questa apre la propria coda di posta. Se questa coda non esiste, l'applicazione ne crea una. Le code di posta hanno il formato CSQ4SAMP.MAILMGR. *userid*, dove *userid* dipende dall'ambiente:

In TSO
L'ID TSO dell'utente

In CICS
Il collegamento CICS dell'utente o l'ID utente immesso dall'utente quando richiesto all'avvio del gestore posta

Tutte le parti dei nomi delle code utilizzate dal gestore posta devono essere in maiuscolo.

L'applicazione presenta quindi un pannello di menu che dispone di opzioni per:

- Leggi posta in arrivo
- Invia email
- Crea nickname

Il pannello del menu mostra anche quanti messaggi sono in attesa nella coda di posta. Ciascuna delle opzioni di menu visualizza un ulteriore pannello:

Leggi posta in arrivo
Il Gestore posta visualizza un elenco dei messaggi che si trovano nella coda di posta. (Vengono visualizzati solo i primi 99 messaggi sulla coda). Per un esempio di questo pannello, consultare Figura 142 a pagina 1198. Quando si seleziona un messaggio da questo elenco, viene visualizzato il contenuto del messaggio (consultare Figura 143 a pagina 1199).

Invia email
Un pannello richiede di immettere:

- Il nome dell'utente a cui si desidera inviare un messaggio
- Il nome del gestore code a cui appartiene la coda di posta
- Il testo del tuo messaggio

Nel campo del nome utente, è possibile immettere un ID utente o un nickname creato utilizzando il Gestore posta. È possibile lasciare vuoto il campo del nome del gestore code se la coda di posta

dell'utente è di proprietà dello stesso gestore code che si sta utilizzando ed è necessario lasciarlo vuoto se è stato immesso un nickname nel campo del nome utente:

- Se si specifica solo un nome utente, il programma assume innanzitutto che il nome sia un nickname e invia il messaggio all'oggetto definito da tale nome. Se non esiste un tale nickname, il programma tenta di inviare il messaggio ad una coda locale con quel nome.
- Se si specifica sia un nome utente che un nome gestore code, il programma invia il messaggio alla coda di posta definita da questi due nomi.

Ad esempio, se si desidera inviare un messaggio all'utente JONESM sul gestore code remoto QM12, è possibile inviare un messaggio in due modi:

- Utilizzare entrambi i campi per specificare l'utente JONESM nel gestore code QM12.
- Definire un nickname (ad esempio, MARY) per tale utente e inviare un messaggio inserendo MARY nel campo del nome utente e nulla nel campo del nome del gestore code.

Crea nickname

È possibile definire un nome facile da ricordare che è possibile utilizzare quando si invia un messaggio a un altro utente che si contatta frequentemente. Viene richiesto di immettere l'ID utente dell'altro utente e il nome del gestore code proprietario della coda di posta.

I nickname sono code con nomi in formato CSQ4SAMP.MAILMGR. *userid.nickname*, dove *userid* è il proprio ID utente e *nickname* è il nickname che si desidera utilizzare. Con i nomi strutturati in questo modo, gli utenti possono ognuno avere la propria serie di nickname.

Il tipo di coda creato dal programma dipende dal modo in cui vengono completati i campi del pannello Crea nickname:

- Se si specifica solo un nome utente o se il nome del gestore code è uguale a quello del gestore code a cui è connesso il gestore posta, il programma crea una coda alias.
- Se si specifica sia un nome utente che un nome gestore code (e il gestore code non è quello a cui è connesso il gestore posta), il programma crea una definizione locale di una coda remota. Il programma non verifica l'esistenza della coda in cui si risolve questa definizione o anche l'esistenza del gestore code remoto.

Ad esempio, se l'ID utente è SMITHK e si crea un nickname denominato MARY per l'utente JONESM (che utilizza il gestore code remoto QM12), il programma nickname crea una definizione locale di una coda remota denominata CSQ4SAMP.MAILMGR.SMITHK.MARY. Questa definizione si risolve nella coda di posta di Mary, che è CSQ4SAMP.MAILMGR.JONESM sul gestore code QM12. Se si utilizza il gestore code QM12 , il programma crea invece una coda alias con lo stesso nome (CSQ4SAMP.MAILMGR.SMITHK.MARY).

La versione C dell'applicazione TSO fa un uso maggiore delle capacità di gestione dei messaggi di ISPFrispetto alla versione COBOL. Si potrebbe notare che le versioni C e COBOL visualizzano diversi messaggi di errore.

z/OS *Progettazione dell'esempio Gestore posta su z/OS*

Le seguenti sezioni descrivono ciascuno dei programmi che costituiscono l'applicazione di esempio di Mail Manager.

Le relazioni tra i programmi e i pannelli utilizzati dall'applicazione vengono mostrati in Figura 140 a pagina 1196 per la versione TSO e in Figura 141 a pagina 1197 per la versione CICS Transaction Server per z/OS .

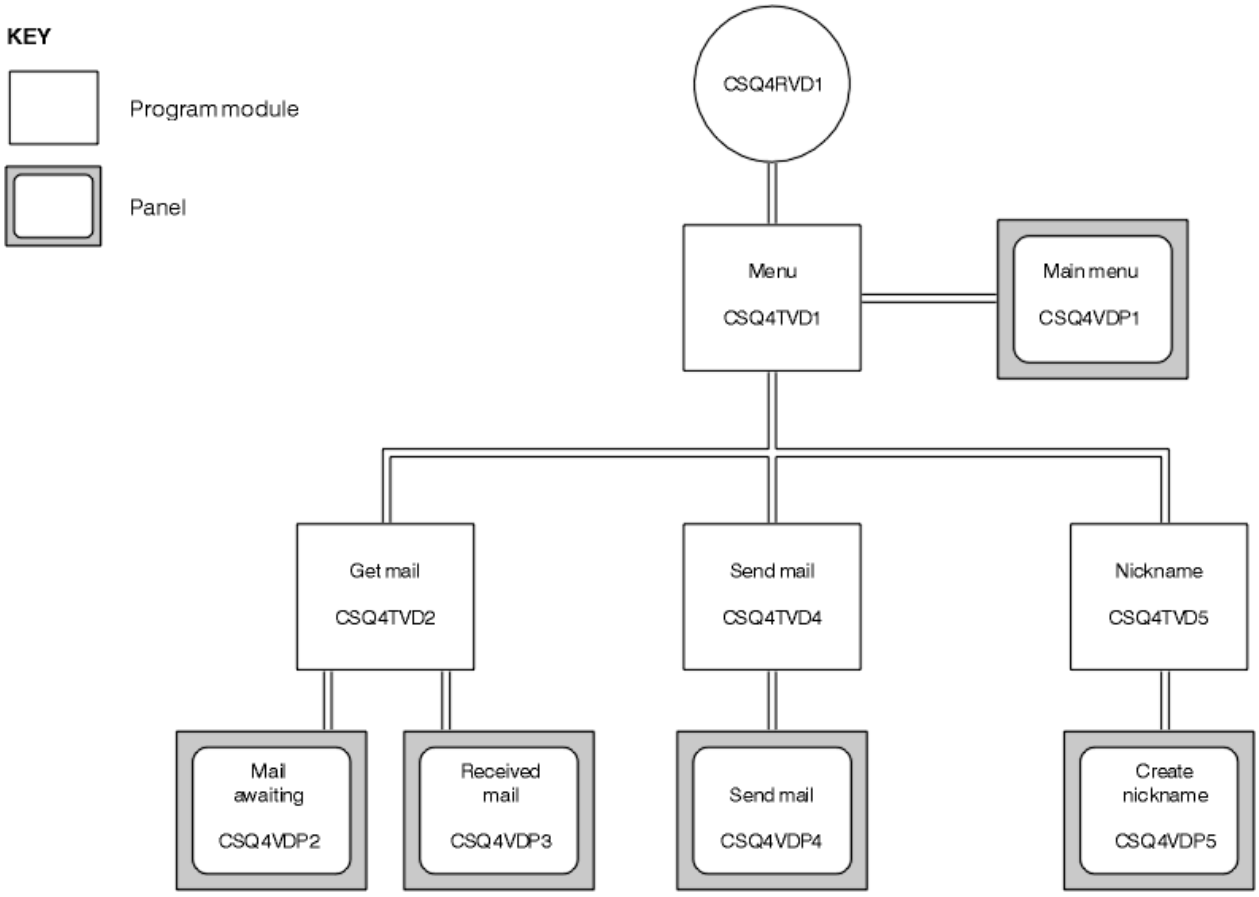


Figura 140. Programmi e pannelli per le versioni TSO di Mail Manager

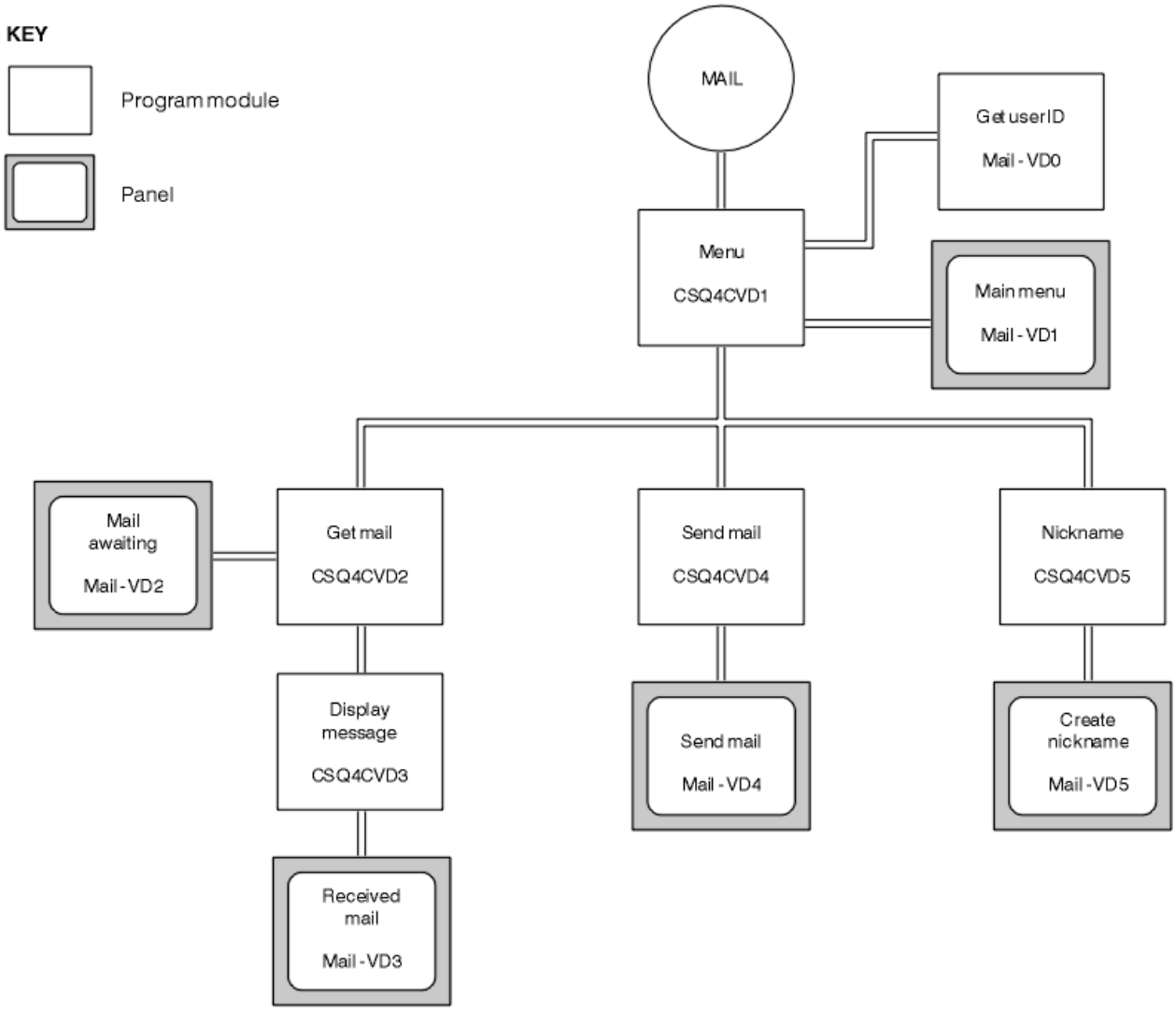


Figura 141. Programmi e pannelli per la versione CICS di Mail Manager

z/OS *Programma di menu su z/OS*

Nell'ambiente TSO, il programma di menu viene richiamato dal CLIST. Nell'ambiente CICS , il programma viene richiamato dalla transazione MAIL.

Il programma menu (CSQ4TVD1 per TSO, CSQ4CVD1 per CICS) è il programma iniziale nella suite. Visualizza il menu (CSQ4VDP1 per TSO, VD1 per CICS) e richiama gli altri programmi quando vengono selezionati dal menu.

Il programma ottiene innanzitutto l'ID utente:

- Nella versione CICS del programma, se l'utente si è collegato a CICS, l'ID utente viene ottenuto utilizzando il comando CICS ASSIGN USERID. Se l'utente non è collegato, il programma visualizza il pannello di collegamento (CSQ4VD0) per richiedere all'utente di immettere un ID utente. Non esiste alcuna elaborazione di sicurezza all'interno di questo programma; l'utente può fornire qualsiasi ID utente.
- Nella versione TSO, l'ID utente viene ottenuto da TSO nel CLIST. Viene passato al programma di menu come una variabile nel pool condiviso ISPF .

Una volta ottenuto l'ID utente, il programma verifica che l'utente abbia una coda di posta (CSQ4SAMP.MAILMGR. *ID utente*). Se una coda di posta non esiste, il programma ne crea una inserendo un messaggio nella coda di immissione del comando di sistema. Il messaggio contiene il comando IBM

MQ for z/OS DEFINE QLOCAL. La definizione oggetto utilizzata da questo comando imposta la profondità massima della coda su 9999 messaggi.

Il programma crea anche una coda dinamica temporanea per gestire le risposte dalla coda di input del comando di sistema. A tale scopo, il programma utilizza la chiamata MQOPEN, specificando SYSTEM.DEFAULT.MODEL.QUEUE come modello per la coda dinamica. Il gestore code crea la coda dinamica temporanea con un nome con prefisso CSQ4SAMP; il resto del nome viene generato dal gestore code.

Il programma apre quindi la coda di posta dell'utente e trova il numero di messaggi sulla coda interrogando la profondità corrente della coda. A tale scopo, il programma utilizza la chiamata MQINQ, specificando il selettore MQIA_CURRENT_Q_DEPTH.

Il programma esegue quindi un loop che visualizza il menu ed elabora la selezione effettuata dall'utente. Il loop viene arrestato quando l'utente preme il tasto PF3 . Quando viene effettuata una selezione valida, viene avviato il programma appropriato, altrimenti viene visualizzato un messaggio di errore.

z/OS *Ricevi - mail e visualizza - programmi di messaggio su z/OS*

Nelle versioni TSO dell'applicazione, le funzioni get - mail e display - message vengono eseguite dallo stesso programma (CSQ4TVD2). Nella versione CICS dell'applicazione, queste funzioni sono eseguite da programmi separati (CSQ4CVD2 e CSQ4CVD3).

Il pannello Posta in attesa (CSQ4VDP2 per TSO, VD2 per CICS ; vedere Figura 142 a pagina 1198 per un esempio) mostra tutti i messaggi che si trovano nella coda di posta dell'utente. Per creare questo elenco, il programma utilizza la chiamata MQGET per esaminare tutti i messaggi sulla coda, salvando le informazioni su ciascuno di essi. Oltre alle informazioni visualizzate, il programma registra `MsgId` e `CorrelId` di ciascun messaggio.

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

Figura 142. Esempio di pannello che mostra un elenco di messaggi in attesa

Dal pannello Posta in attesa l'utente può selezionare un messaggio e visualizzare il contenuto del messaggio (consultare Figura 143 a pagina 1199 per un esempio). Il programma usa la chiamata MQGET per rimuovere questo messaggio dalla coda, utilizzando `MsgId` e `CorrelId` che il programma ha annotato quando ha sfogliato tutti i messaggi. Questa chiamata MQGET viene eseguita utilizzando l'opzione MQGMO_SYNCPOINT. Il programma visualizza il contenuto del messaggio, quindi dichiara un punto di sincronizzazione: questo esegue il commit della chiamata MQGET, in modo che il messaggio non esista più.

```
-------------------- IBM MQ for z/OS Sample Programs --------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
------------------------------------ Message -------------------------------
| HELLO FROM JOHNJ                                |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
'-----------------------------------------------------------------------------'
```

Figura 143. Esempio di pannello che visualizza il contenuto di un messaggio

Un'estensione ovvia della funzione fornita da Mail Manager è quella di dare all'utente la possibilità di lasciare il messaggio sulla coda dopo averne visualizzato il contenuto. Per fare ciò, è necessario eseguire il backout della chiamata MQGET che rimuove il messaggio dalla coda, dopo aver visualizzato il messaggio.

z/OS *Programma di invio posta su z/OS*

Quando l'utente ha completato il pannello Invia posta (CSQ4VDP4 per TSO, VD4 per CICS), il programma di invio posta (CSQ4TVD4 per TSO, CSQ4CVD4 per CICS) inserisce il messaggio nella coda di posta del destinatario.

A tale scopo, il programma utilizza la chiamata MQPUT1 . La destinazione del messaggio dipende dal modo in cui l'utente ha riempito i campi nel pannello Invia posta:

- Se l'utente ha specificato solo un nome utente, il programma assume innanzitutto che il nome sia un nickname e invia il messaggio all'oggetto definito da tale nome. Se non esiste un tale nickname, il programma tenta di inviare il messaggio ad una coda locale con quel nome.
- Se l'utente ha specificato sia un nome utente che un nome gestore code, il programma invia il messaggio alla coda di posta definita da questi due nomi.

Il programma non accetta messaggi vuoti ed elimina gli spazi vuoti iniziali da ogni riga del testo del messaggio.

Se la chiamata MQPUT1 ha esito positivo, il programma visualizza un messaggio che indica che il nome utente e il nome del gestore code in cui è stato inserito il messaggio. Se la chiamata ha esito negativo, il programma controlla in modo specifico i codici motivo che indicano che la coda o il gestore code non esistono; si tratta di MQRC_UNKNOWN_OBJECT_NAME e MQRC_UNKNOWN_OBJECT_Q_MGR. Il programma visualizza il suo messaggio di errore per ciascuno di questi errori; per altri errori, il programma visualizza i codici di completamento e di errore restituiti dalla chiamata.

z/OS *Programma nickname su z/OS*

Quando l'utente definisce un nickname, il programma del nickname (CSQ4TVD5 per TSO, CSQ4CVD5 per CICS) crea una coda che ha il nickname come parte del nome.

Il programma esegue questa operazione inserendo un messaggio nella coda di immissione del comando di sistema. Il messaggio contiene il comando IBM MQ for z/OS DEFINE QALIAS o DEFINE QREMOTE. Il tipo di coda che il programma crea dipende da come l'utente ha riempito i campi del pannello Crea nickname (CSQ4VDP5 per TSO, VD5 per CICS):

- Se l'utente ha specificato solo un nome utente o se il nome del gestore code è uguale a quello del gestore code a cui è connesso il gestore posta, il programma crea una coda alias.
- Se l'utente ha specificato sia un nome utente che un nome gestore code (e il gestore code non è quello a cui è connesso il gestore posta), il programma crea una definizione locale di una coda remota. Il programma non verifica l'esistenza della coda in cui si risolve questa definizione o anche l'esistenza del gestore code remoto.

Il programma crea anche una coda dinamica temporanea per gestire le risposte dalla coda di input del comando di sistema.

Se il gestore code non è in grado di creare la coda del nickname per un motivo previsto dal programma (ad esempio, la coda già esiste), il programma visualizza il proprio messaggio di errore. Se il gestore code non riesce a creare la coda per un motivo che il programma non prevede, il programma visualizza fino a due dei messaggi di errore restituiti al programma dal server dei comandi.

Nota: Per ogni nome abbreviato, il programma crea solo una coda alias o una definizione locale di una coda remota. Le code locali in cui si risolvono questi nomi di coda vengono create solo quando l'ID utente contenuto nel nickname viene utilizzato per avviare l'applicazione Gestore posta.

z/OS *L'esempio Credit Check su z/OS*

L'applicazione di esempio Controllo del credito è una suite di programmi che dimostra come utilizzare molte funzioni fornite da IBM MQ for z/OS. Mostra il modo in cui i numerosi programmi componenti di un'applicazione possono passare i messaggi l'uno all'altro utilizzando le tecniche di accodamento dei messaggi.

L'esempio può essere eseguito come applicazione CICS autonoma. Tuttavia, per dimostrare come progettare un'applicazione di accodamento dei messaggi che utilizza le funzioni fornite dagli ambienti CICS e IMS , viene fornito anche un modulo come un programma di elaborazione dei messaggi batch IMS . Questa estensione dell'esempio è descritta in “L'estensione IMS all'esempio Controllo del credito su z/OS” a pagina 1211.

È inoltre possibile eseguire l'esempio su più di un gestore code e inviare messaggi tra ciascuna istanza dell'applicazione. Per fare ciò, consultare “L'esempio Controllo credito con più gestori code su z/OS” a pagina 1211.

I programmi CICS vengono forniti in C e COBOL. Il singolo programma IMS viene fornito solo in C. I dataset forniti sono riportati in Tabella 182 a pagina 1179 e Tabella 184 a pagina 1181.

L'applicazione dimostra un metodo di valutazione del rischio quando i clienti della banca chiedono prestiti. L'applicazione mostra come una banca può funzionare in due modi per elaborare le richieste di prestito:

- Quando si ha a che fare direttamente con un cliente, il personale della banca desidera accedere immediatamente alle informazioni sul conto e sul rischio di credito.
- Quando si tratta di domande scritte, il personale della banca può presentare una serie di richieste di informazioni sul conto e sul rischio di credito e trattare le risposte in un secondo momento.

I dettagli finanziari e di sicurezza nell'applicazione sono stati mantenuti semplici in modo che le tecniche di accodamento dei messaggi siano chiare.

z/OS *Preparazione ed esecuzione dell'esempio Controllo credito su z/OS*

Per preparare ed eseguire l'esempio Controllo credito, effettuare le seguenti operazioni:

1. Creare il dataset VSAM che contiene le informazioni su alcuni account di esempio. Eseguire questa operazione modificando ed eseguendo il JCL fornito nel dataset CSQ4FILE.
2. Seguire le istruzioni contenute in “Preparazione delle applicazioni di esempio per l'ambiente CICS su z/OS” a pagina 1176. (I passi aggiuntivi che è necessario eseguire se si desidera utilizzare l'estensione IMS per l'esempio sono descritti in “L'estensione IMS all'esempio Controllo del credito su z/OS” a pagina 1211.)

3. Avviare il controllo dei trigger CKTI (fornito con IBM MQ for z/OS) sulla coda CSQ4SAMP.INITIATION.QUEUE, utilizzando la transazione CICS CKQC.
4. Per avviare l'applicazione, avviare il sistema CICS e utilizzare la transazione MVB1.
5. Selezionare **Immediata** o **Batch** dal primo pannello.

   I pannelli di interrogazione immediata e batch sono simili; Figura 144 a pagina 1201 mostra il pannello di interrogazione immediata.

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . _______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : _____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

Figura 144. Pannello di richiesta immediata per l'applicazione di esempio Credit Check

6. Immettere un numero di conto e un importo del prestito nei campi appropriati. Consultare “Immissione di informazioni nei pannelli di interrogazione” a pagina 1201 per istruzioni su quali informazioni immettere in questi campi.

Immissione di informazioni nei pannelli di interrogazione

L'applicazione di esempio Controllo credito verifica che i dati immessi nel campo **Importo richiesto** dei pannelli di interrogazione siano sotto forma di numeri interi.

Se si immette uno dei seguenti numeri di conto, l'applicazione trova il nome del conto appropriato, il saldo medio e l'indice di affidabilità creditizia nel dataset VSAM CSQ4BAQ:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

È possibile immettere qualsiasi informazione o no negli altri campi. L'applicazione conserva tutte le informazioni immesse e restituisce le stesse informazioni nei report che genera.

z/OS *Progettazione del campione Credit Check su z/OS*

Questa sezione descrive la progettazione di ciascuno dei programmi che costituiscono l'applicazione di esempio Controllo credito.

Per ulteriori informazioni su alcune delle tecniche considerate durante la progettazione dell'applicazione, consultare “Considerazioni sulla progettazione per l'esempio Controllo credito su z/OS” a pagina 1208.

La Figura 145 a pagina 1203 mostra i programmi che costituiscono l'applicazione e anche le code utilizzate da questi programmi. In questa figura, il prefisso CSQ4SAMP è stato omesso da tutti i nomi di coda per rendere la figura più semplice da comprendere.

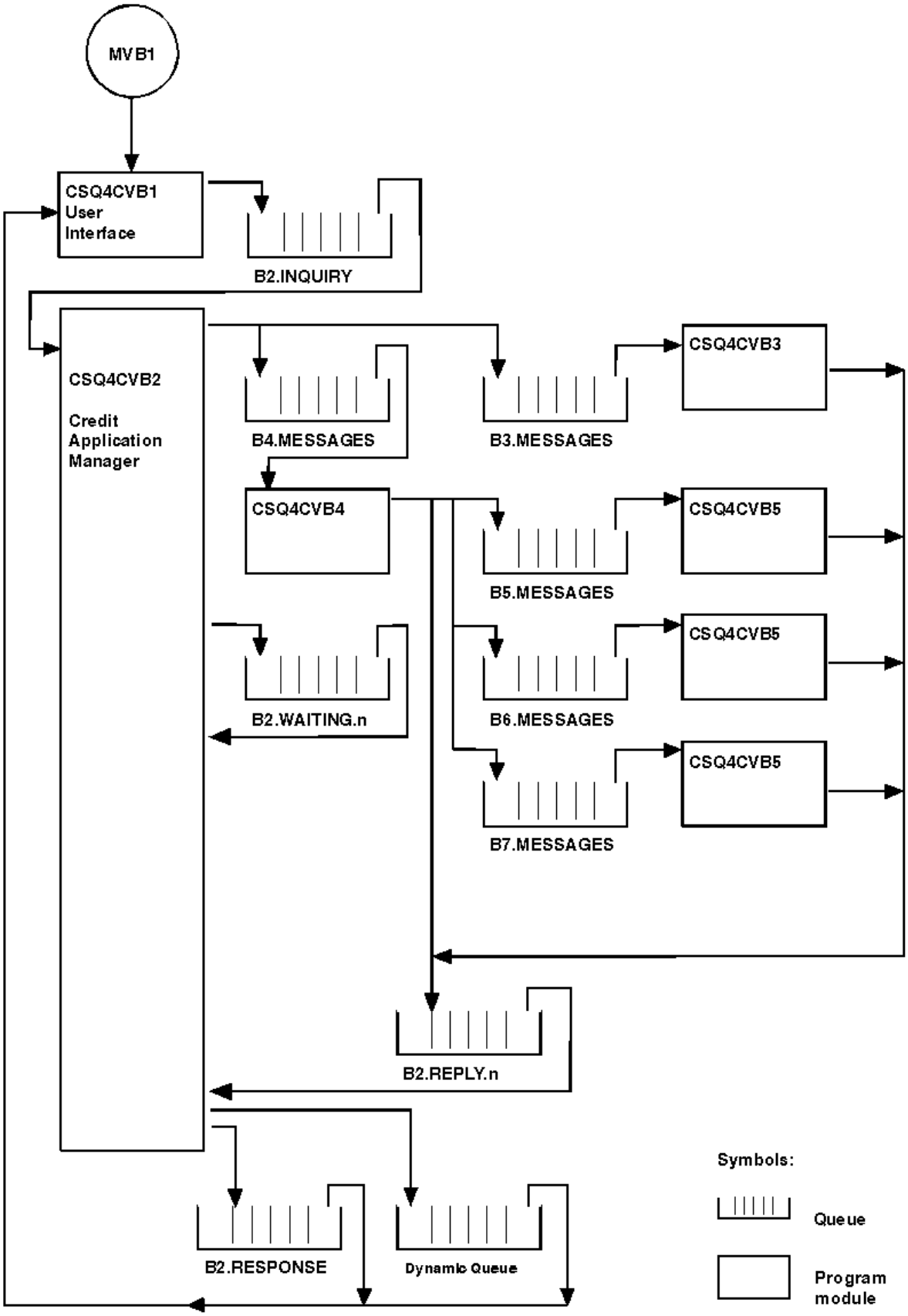


Figura 145. Programmi e code per l'applicazione di esempio Credit Check (solo programmi COBOL)

z/OS *Programma interfaccia utente (CSQ4CVB1) su z/OS*

Quando si avvia la CICS transazione MVB1in modalità conversazione, viene avviato il programma dell'interfaccia utente per l'applicazione.

Questo programma inserisce i messaggi di interrogazione nella coda CSQ4SAMP.B2.INQUIRY e riceve risposte a tali richieste da una coda di risposta che specifica quando effettua l'interrogazione. Dall'interfaccia utente è possibile inoltrare richieste immediate o batch:

- Per richieste immediate, il programma crea una coda dinamica temporanea che utilizza come coda di risposta. Ciò significa che ogni interrogazione ha la sua coda di risposta.
- Per le richieste batch, il programma di interfaccia utente riceve risposte dalla coda CSQ4SAMP.B2.RESPONSE. Per semplicità, il programma riceve risposte per tutte le sue richieste da questa coda di risposta. È facile vedere che una banca potrebbe voler utilizzare una coda di risposta separata per ciascun utente di MVB1, in modo che ciascuno possa visualizzare le risposte solo a quelle richieste che avevano avviato.

Le differenze importanti tra le proprietà dei messaggi utilizzati nell'applicazione in modalità batch e immediata sono:

- Per il lavoro batch, i messaggi hanno una priorità bassa, quindi vengono elaborati dopo eventuali richieste di prestito immesse in modalità immediata. Inoltre, i messaggi sono persistenti, quindi vengono ripristinati se l'applicazione o il gestore code devono essere riavviati.
- Per un utilizzo immediato, i messaggi hanno una priorità elevata, quindi vengono elaborati prima di qualsiasi richiesta di prestito immessa in modalità batch. Inoltre, i messaggi non sono persistenti, quindi vengono eliminati se l'applicazione o il gestore code devono essere riavviati.

Tuttavia, in tutti i casi, le proprietà dei messaggi di richiesta di prestito vengono propagate in tutta l'applicazione. Pertanto, ad esempio, tutti i messaggi che risultano da una richiesta ad alta priorità avranno anche una priorità elevata.

z/OS *Credit application manager (CSQ4CVB2) su z/OS*

Il programma CAM (Credit Application Manager) esegue la maggior parte dell'elaborazione per l'applicazione Controllo credito.

CAM viene avviato dal controllo trigger CKTI (fornito con IBM MQ for z/OS) quando si verifica un evento trigger su una coda CSQ4SAMP.B2.INQUIRY o coda CSQ4SAMP.B2.REPLY. *n*, dove *n* è un numero intero che identifica una delle code di risposta. Il messaggio trigger contiene dati che includono il nome della coda in cui si è verificato l'evento trigger.

CAM utilizza code con nomi in formato CSQ4SAMP CSQ4SAMP.B2.WAITING.n per memorizzare le informazioni sulle richieste che sta elaborando. Le code vengono denominate in modo che siano ciascuna associata ad una coda di risposta; ad esempio, la coda CSQ4SAMP.B2.WAITING.3 contiene i dati di input per una particolare interrogazione e coda CSQ4SAMP.B2.REPLY.3 contiene una serie di messaggi di risposta (da programmi che interrogano i database) tutti relativi alla stessa interrogazione. Per comprendere le ragioni di questa progettazione, consultare la sezione “Separare le code di interrogazione e risposta in CAM” a pagina 1209.

Logica di avvio

Se l'evento trigger si verifica nella coda CSQ4SAMP.B2.INQUIRY, CAM apre la coda per l'accesso condiviso. Quindi tenta di aprire ogni coda di risposta fino a quando non ne viene trovata una libera. Se non riesce a trovare una coda di risposta libera, il CAM registra il fatto e termina normalmente.

Se l'evento trigger si verifica nella coda CSQ4SAMP.B2.REPLY.n, CAM apre la coda per l'accesso esclusivo. Se il codice di ritorno indica che l'oggetto è già in uso, CAM termina normalmente. Se si verifica un altro errore, il CAM registra l'errore e termina. Il CAM apre la coda di attesa corrispondente e la coda di interrogazione, quindi inizia a ricevere ed elaborare i messaggi. Dalla coda di attesa, CAM recupera i dettagli delle richieste completate parzialmente.

Per semplicità in questo esempio, i nomi delle code utilizzate sono contenuti nel programma. In un ambiente di business, i nomi delle code verrebbero probabilmente conservati in un file a cui accede il programma.

Richiamo di un messaggio dalla coda di interrogazione

Il CAM tenta per primo di richiamare un messaggio dalla coda di interrogazione utilizzando la chiamata MQGET con opzione MQGMO_SET_SIGNAL. Se un messaggio è disponibile immediatamente, il messaggio viene elaborato; se non è disponibile alcun messaggio, viene impostato un segnale.

Il CAM tenta quindi di richiamare un messaggio dalla coda di risposta, utilizzando nuovamente la chiamata MQGET con la stessa opzione. Se un messaggio è disponibile immediatamente, il messaggio viene elaborato; altrimenti viene impostato un segnale.

Quando vengono impostati entrambi i segnali, il programma attende che venga inviato uno dei segnali. Se viene inviato un segnale per indicare che un messaggio è disponibile, il messaggio viene richiamato ed elaborato. Se il segnale scade o il gestore code è in fase di chiusura, il programma viene terminato.

Elaborazione del messaggio richiamato da CAM

Un messaggio richiamato da CAM può essere di quattro tipi:

- Un messaggio di interrogazione
- Un messaggio di risposta
- Un messaggio di propagazione
- Un messaggio imprevisto o indesiderato

CAM elabora questi messaggi come descritto in “Elaborazione del messaggio richiamato da CAM su z/OS” a pagina 1206.

Invio di una risposta

Quando il CAM ha ricevuto tutte le risposte previste per un'interrogazione, elabora le risposte e crea un singolo messaggio di risposta. Consolida in un unico messaggio tutti i dati di tutti i messaggi di risposta che hanno lo stesso `CorrelId`. Questa risposta viene inserita nella coda di risposta specificata nella richiesta di prestito originale. Il messaggio di risposta viene inserito all'interno della stessa unità di lavoro che contiene il richiamo del messaggio di risposta finale. Questo per semplificare il recupero assicurando che non ci sia mai un messaggio completato sulla coda CSQ4SAMP.B2.WAITING.n.

Recupero di indagini parzialmente completate

Le copie CAM nella coda CSQ4SAMP.B2.WAITING.n tutti i messaggi che riceve. Imposta i campi del descrittore del messaggio come segue:

- *`Priority`* è determinato dal tipo di messaggio:
  - Per i messaggi di richiesta, priorità = 3
  - Per i datagrammi, priorità = 2
  - Per i messaggi di risposta, priorità = 1
- *`CorrelId`* è impostato su *`MsgId`* del messaggio di richiesta di prestito
- Gli altri campi MQMD vengono copiati da quelli del messaggio ricevuto

Quando un'interrogazione è stata completata, i messaggi per un'interrogazione specifica vengono rimossi dalla coda di attesa durante l'elaborazione della risposta. Pertanto, in qualsiasi momento, la coda di attesa contiene tutti i messaggi rilevanti per le richieste in corso. Questi messaggi vengono utilizzati per recuperare i dettagli delle richieste in corso se il programma deve essere riavviato. Le diverse priorità vengono impostate in modo che i messaggi di interrogazione vengano recuperati prima delle propagazioni o dei messaggi di risposta.

z/OS *Elaborazione del messaggio richiamato da CAM su z/OS*

Un messaggio richiamato da CAM (Credit Application Manager) può essere di quattro tipi. Il modo in cui CAM elabora un messaggio dipende dal suo tipo.

Un messaggio richiamato da CAM può essere di quattro tipi:

- Un messaggio di interrogazione
- Un messaggio di risposta
- Un messaggio di propagazione
- Un messaggio imprevisto o indesiderato

CAM elabora questi messaggi nel modo seguente:

Messaggio di interrogazione

I messaggi di interrogazione provengono dal Programma interfaccia utente. Crea un messaggio di interrogazione per ogni richiesta di prestito.

Per tutte le richieste di prestito, il CAM richiede il saldo medio del conto corrente del cliente. Lo fa inserendo un messaggio di richiesta sulla coda alias CSQ4SAMP.B2.OUTPUT.ALIAS. Questo nome coda viene risolto nella coda CSQ4SAMP.B3.MESSAGES, elaborato dal programma di controllo - account, CSQ4CVB3. Quando CAM inserisce un messaggio su questa coda alias, specifica il CSQ4SAMP CSQ4SAMP.B2.REPLY.n coda per la coda di risposta. Viene utilizzata una coda alias in modo che il programma CSQ4CVB3 possa essere facilmente sostituito da un altro programma che elabora una coda di base con un nome differente. Per fare ciò, si ridefinisce la coda alias in modo che il suo nome si risolva nella nuova coda. Inoltre, è possibile assegnare autorizzazioni di accesso differenti alla coda alias e a quella base.

Se un utente richiede un prestito superiore a 10000 unità, CAM avvia i controlli anche su altri database. Lo fa inserendo un messaggio di richiesta sulla coda CSQ4SAMP.B4.MESSAGES, elaborato dal programma di distribuzione, CSQ4CVB4. Il processo che serve questa coda propaga il messaggio alle code servite da programmi che hanno accesso ad altri record come la cronologia della carta di credito, i conti di risparmio e i pagamenti ipotecari. I dati di questi programmi vengono restituiti alla coda di risposta specificata nell'operazione di inserimento. Inoltre, un messaggio di propagazione viene inviato alla coda di risposta da questo programma per specificare quanti messaggi di propagazione sono stati inviati.

In un ambiente aziendale, il programma di distribuzione probabilmente riformatterebbe i dati forniti per corrispondere al formato richiesto da ciascuno degli altri tipi di conto bancario.

Qualsiasi coda a cui si fa riferimento può trovarsi su un sistema remoto.

Per ogni messaggio di interrogazione, il CAM avvia una voce nell'IRT (memory-resident Inquiry Record Table). Questo record contiene:

- Il `MsgId` del messaggio di interrogazione
- Nel campo ReplyExp , il numero di risposte previste (uguale al numero di messaggi inviati)
- Nel campo ReplyRec , il numero di risposte ricevute (zero in questa fase)
- Nel campo PropsOut , indica se è previsto un messaggio di propagazione

Il CAM copia il messaggio di interrogazione nella coda di attesa con:

- `Priority` impostato su 3
- `CorrelId` impostato sul `MsgId` del messaggio di interrogazione
- Gli altri campi descrittore del messaggio impostati su quelli del messaggio di interrogazione

Messaggio di propagazione

Un messaggio di propagazione contiene il numero di code a cui il programma di distribuzione ha inoltrato l'interrogazione. Il messaggio viene elaborato come segue:

1. Aggiungere al campo ReplyExp del record appropriato nell'IRT il numero di messaggi inviati. Queste informazioni si trovano nel messaggio.

2. Incrementare di 1 il campo ReplyRec del record nell'IRT.
3. Decrementare di 1 il campo PropsOut del record nell'IRT.
4. Copiare il messaggio nella coda di attesa. CAM imposta `Priority` su 2 e gli altri campi del descrittore del messaggio su quelli del messaggio di propagazione.

Messaggio di risposta

Un messaggio di risposta contiene la risposta a una delle richieste al programma di account di controllo o a uno dei programmi di query dell'agenzia. I messaggi di risposta vengono elaborati come segue:

1. Incrementare di 1 il campo ReplyRec del record nell'IRT.
2. Copiare il messaggio nella coda di attesa con `Priority` impostato su 1 e gli altri campi del descrittore del messaggio impostati su quelli del messaggio di risposta.
3. Se ReplyRec = ReplyExpe PropsOut = 0, impostare l'indicatore MsgComplete .

È possibile che vengano emessi anche altri messaggi

L'applicazione non prevede altri messaggi. Tuttavia, l'applicazione potrebbe ricevere messaggi trasmessi dal sistema o messaggi di risposta con un `CorrelId`sconosciuto.

CAM inserisce questi messaggi nella coda CSQ4SAMP.DEAD.QUEUE, dove possono essere esaminati. Se questa operazione di inserimento non riesce, il messaggio viene perso e il programma continua. Per ulteriori informazioni sulla progettazione di questa parte del programma, consultare "Modalità di gestione dei messaggi non previsti da parte dell'esempio" a pagina 1209.

z/OS *Programma di controllo account (CSQ4CVB3) su z/OS*

Il programma di controllo - account viene avviato da un evento trigger sulla coda CSQ4SAMP.B3.MESSAGES. Dopo aver aperto la coda, questo programma riceve un messaggio dalla coda utilizzando la chiamata MQGET con l'opzione di attesa e con l'intervallo di attesa impostato su 30 secondi.

Il programma ricerca il dataset VSAM CSQ4BAQ per il numero di conto nel messaggio di richiesta di prestito. Richiama il nome del conto corrispondente, il saldo medio e l'indice di affidabilità creditizia oppure nota che il numero di conto non si trova nel dataset.

Quindi, il programma inserisce un messaggio di risposta (utilizzando la chiamata MQPUT1) nella coda di risposta denominata nel messaggio di richiesta prestito. Per questo messaggio di risposta, il programma:

- Copia il `CorrelId` del messaggio di richiesta di prestito
- Utilizza l'opzione MQPMO_PASS_IDENTITY_CONTEXT

Il programma continua a richiamare i messaggi dalla coda fino alla scadenza dell'intervallo di attesa.

z/OS *Programma di distribuzione (CSQ4CVB4) su z/OS*

Il programma di distribuzione viene avviato da un evento trigger sulla coda CSQ4SAMP.B4.MESSAGES.

Per simulare la distribuzione della richiesta di prestito ad altre agenzie che hanno accesso a record quali la cronologia della carta di credito, i conti di risparmio e i pagamenti del mutuo, il programma inserisce una copia dello stesso messaggio in tutte le code nell'elenco dei nomi CSQ4SAMP.B4.NAMELIST. Esistono tre di queste code, con nomi nel formato CSQ4SAMP.B *n*.MESSAGES, dove *n* è 5, 6 o 7. In un'applicazione di business, le agenzie potrebbero trovarsi in ubicazioni separate, quindi queste code potrebbero essere code remote. Se si desidera modificare l'applicazione di esempio per visualizzarla, consultare "L'esempio Controllo credito con più gestori code su z/OS" a pagina 1211.

Il programma di distribuzione effettua le seguenti operazioni:

1. Dall'elenco nomi, richiama i nomi delle code che il programma deve utilizzare. Il programma esegue questa operazione utilizzando la chiamata MQINQ per richiedere informazioni sugli attributi dell'oggetto elenco nomi.
2. Apre queste code e anche CSQ4SAMP.B4.MESSAGES.
3. Esegue il seguente loop fino a quando non sono più presenti messaggi sulla coda CSQ4SAMP.B4.MESSAGES:

a. Richiamare un messaggio utilizzando la chiamata MQGET con l'opzione wait e con l'intervallo di attesa impostato su 30 secondi.
b. Inserire un messaggio su ciascuna coda elencata nell'elenco nomi, specificando il nome del CSQ4SAMP.B2.REPLY.n coda per la coda di risposta. Il programma copia il *CorrelId* del messaggio di richiesta di prestito in questi messaggi di copia e utilizza l'opzione MQPMO_PASS_IDENTITY_CONTEXT sulla chiamata MQPUT.
c. Inviare un messaggio datagram alla coda CSQ4SAMP.B2.REPLY.n per mostrare quanti messaggi ha inserito correttamente.
d. Dichiarare un punto di sincronizzazione.

z/OS *Programma di query dell'agenzia (CSQ4CVB5/CSQ4CCB5) su z/OS*

Il programma agency - query viene fornito sia come programma COBOL che come programma C. Entrambi i programmi hanno lo stesso design. Ciò mostra che i programmi di diversi tipi possono coesistere facilmente all'interno di un'applicazione IBM MQ e che i moduli di programma che costituiscono tale applicazione possono essere facilmente sostituiti.

Un'istanza del programma viene avviata da un evento trigger su una delle seguenti code:

- Per il programma COBOL (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- Per il programma C (CSQ4CCB5), coda CSQ4SAMP.B8.MESSAGES

Nota: Se si desidera utilizzare il programma C, è necessario modificare la definizione dell'elenco nomi CSQ4SAMP.B4.NAMELIST per sostituire la coda CSQ4SAMP.B7.MESSAGES con CSQ4SAMP.B8.MESSAGES. Per eseguire questa operazione, è possibile utilizzare uno dei seguenti:

- Le operazioni e i pannelli di controllo IBM MQ for z/OS
- Comando ALTER NAMELIST
- Programma di utilità CSQUTIL

Dopo aver aperto la coda appropriata, questo programma richiama un messaggio dalla coda utilizzando la chiamata MQGET con l'opzione di attesa e con l'intervallo di attesa impostato su 30 secondi.

Il programma simula la ricerca di un database dell'agenzia ricercando il data set VSAM CSQ4BAQ per il numero di conto passato nel messaggio di richiesta di prestito. Quindi genera una risposta che include il nome della coda che sta servendo e un indice di affidabilità creditizia. Per semplificare l'elaborazione, l'indice di affidabilità creditizia viene selezionato a caso.

Quando si immette il messaggio di replica, il programma utilizza la chiamata MQPUT1 e:

- Copia il `CorrelId` del messaggio di richiesta di prestito
- Utilizza l'opzione MQPMO_PASS_IDENTITY_CONTEXT

Il programma invia il messaggio di risposta alla coda di risposta indicata nel messaggio di richiesta di prestito. (Il nome del gestore code proprietario della coda di risposta è specificato anche nel messaggio di richiesta di prestito.)

z/OS *Considerazioni sulla progettazione per l'esempio Controllo credito su z/OS*

Considerazioni sulla progettazione per l'esempio Controllo del credito.

Questo argomento contiene informazioni su:

- “Separare le code di interrogazione e risposta in CAM” a pagina 1209
- “Come l'esempio gestisce gli errori” a pagina 1209
- “Modalità di gestione dei messaggi non previsti da parte dell'esempio” a pagina 1209
- “Modalità di utilizzo dei punti di sincronizzazione da parte dell'esempio” a pagina 1210

- “Modalità di utilizzo delle informazioni di contesto del messaggio da parte dell'esempio” a pagina 1210
- “Utilizzo degli identificativi di messaggio e correlazione in CAM” a pagina 1211

Separare le code di interrogazione e risposta in CAM

L'applicazione poteva utilizzare una singola coda sia per le richieste che per le risposte, ma è stata progettata per utilizzare code separate per i seguenti motivi:

- Quando il programma gestisce il numero massimo di richieste, è possibile lasciare ulteriori richieste sulla coda. Se viene utilizzata una singola coda, questa dovrebbe essere tolta dalla coda e memorizzata altrove.
- Altre istanze di CAM potrebbero essere avviate automaticamente per servire la stessa coda di interrogazione se il traffico dei messaggi fosse abbastanza elevato da garantirne l'uso. Ma il programma deve tenere traccia delle richieste in corso, e per farlo, deve ottenere tutte le risposte alle richieste che ha avviato. Se viene utilizzata una sola coda, il programma dovrà esaminare i messaggi per vedere se sono per questo o per un altro programma. Ciò renderebbe l'operazione molto meno efficiente.

  L'applicazione può supportare più CAM e può recuperare le richieste in corso in modo efficace utilizzando code di risposta e di attesa accoppiate.
- Il programma può attendere su più code in modo efficace utilizzando la segnalazione.

Come l'esempio gestisce gli errori

Il programma dell'interfaccia utente gestisce gli errori notificandoli direttamente all'utente.

Gli altri programmi non dispongono di interfacce utente, quindi devono gestire gli errori in altri modi. Inoltre, in molte situazioni (ad esempio, se una chiamata MQGET non riesce) questi altri programmi non conoscono l'identità dell'utente dell'applicazione.

Gli altri programmi immettono messaggi di errore su una coda di storage temporaneo CICS denominata CSQ4SAMP. È possibile sfogliare questa coda utilizzando la transazione CEBR fornita da CICS. I programmi inoltre scrivono messaggi di errore nel log CSML di CICS .

Modalità di gestione dei messaggi non previsti da parte dell'esempio

Quando si progetta un'applicazione di accodamento messaggi, è necessario decidere come gestire inaspettatamente i messaggi che arrivano su una coda.

Le due scelte di base sono:

- L'applicazione non funziona più fino a quando non ha elaborato il messaggio non previsto. Ciò probabilmente significa che l'applicazione notifica un operatore, termina se stessa e garantisce che non venga riavviata automaticamente (può farlo disattivando l'attivazione). Questa scelta indica che tutta l'elaborazione per l'applicazione può essere interrotta da un singolo messaggio non previsto e l'intervento di un operatore è richiesto per riavviare l'applicazione.
- L'applicazione rimuove il messaggio dalla coda che sta servendo, lo inserisce in un'altra posizione e continua l'elaborazione. Il posto migliore per inserire questo messaggio è nella coda di messaggi non recapitabili del sistema.

Se si sceglie la seconda opzione:

- Un operatore, o un altro programma, dovrebbe esaminare i messaggi che vengono inseriti nella coda di messaggi non recapitabili per scoprire da dove provengono i messaggi.
- Un messaggio non previsto viene perso se non può essere inserito nella coda di messaggi non recapitabili.
- Un messaggio lungo e imprevisto viene troncato se è più lungo del limite per i messaggi nella coda di messaggi non recapitabili o più lungo della dimensione del buffer nel programma.

Per garantire che l'applicazione gestisca senza problemi tutte le richieste con un effetto minimo dalle attività esterne, l'applicazione di esempio Credit Check utilizza la seconda opzione. Per consentire di

mantenere l'esempio separato dalle altre applicazioni che utilizzano lo stesso gestore code, l'esempio Controllo credito non utilizza la coda di messaggi non recapitabili del sistema; utilizza invece la propria coda di messaggi non recapitabili. Questa coda è denominata CSQ4SAMP.DEAD.QUEUE. L'esempio tronca tutti i messaggi che sono più lunghi dell'area di buffer fornita per i programmi di esempio. È possibile utilizzare l'applicazione di esempio Sfoglia per sfogliare i messaggi su questa coda oppure utilizzare l'applicazione di esempio Stampa messaggio per stampare i messaggi insieme ai relativi descrittori di messaggi.

Tuttavia, se si estende l'esempio in modo che venga eseguito su più di un gestore code, i messaggi non previsti o i messaggi che non possono essere consegnati potrebbero essere inseriti nella coda di messaggi non instradabili del sistema dal gestore code.

Modalità di utilizzo dei punti di sincronizzazione da parte dell'esempio

I programmi nell'applicazione di esempio Controllo credito dichiarano i punti di sincronizzazione per garantire che:

- Solo un messaggio di risposta viene inviato in risposta a ciascun messaggio previsto
- Più copie di messaggi non previsti non vengono mai inserite nella coda di messaggi non recapitabili dell'esempio
- CAM può ripristinare lo stato di tutte le richieste completate parzialmente richiamando i messaggi persistenti dalla coda di attesa

Per ottenere ciò, viene utilizzata una singola unità di lavoro per coprire l'acquisizione di un messaggio, l'elaborazione di tale messaggio e tutte le successive operazioni di inserimento.

Modalità di utilizzo delle informazioni di contesto del messaggio da parte dell'esempio

Quando il programma di interfaccia utente (CSQ4CVB1) invia messaggi, utilizza l'opzione MQPMO_DEFAULT_CONTEXT. Ciò significa che il gestore code genera informazioni sul contesto di origine e di identità. Il gestore code riceve queste informazioni dalla transazione che ha avviato il programma (MVB1) e dall'ID utente che ha avviato la transazione.

Quando CAM invia messaggi di interrogazione, utilizza l'opzione MQPMO_PASS_IDENTITY_CONTEXT. Ciò significa che le informazioni sul contesto identità del messaggio inserito vengono copiate dal contesto identità del messaggio di interrogazione originale. Con questa opzione, le informazioni sul contesto di origine vengono generate dal gestore code.

Quando il CAM invia messaggi di risposta, utilizza l'opzione MQPMO_ALTERNATE_USER_AUTHORITY. Ciò fa sì che il gestore code utilizzi un ID utente alternativo per il relativo controllo di sicurezza quando CAM apre una coda di risposta. CAM utilizza l'ID utente del mittente del messaggio di interrogazione originale. Ciò significa che agli utenti è consentito visualizzare le risposte solo alle domande che hanno originato. L'ID utente alternativo viene ottenuto dalle informazioni di contesto dell'identità nel descrittore del messaggio di interrogazione originale.

Quando i programmi di query (CSQ4CVB3/4/5) inviano messaggi di risposta, utilizzano l'opzione MQPMO_PASS_IDENTITY_CONTEXT. Ciò significa che le informazioni sul contesto identità del messaggio inserito vengono copiate dal contesto identità del messaggio di interrogazione originale. Con questa opzione, le informazioni sul contesto di origine vengono generate dal gestore code.

Nota: L'ID utente associato alle transazioni MVB3/4/5 richiede l'accesso a B2.REPLY.n code. Questi ID utente potrebbero non essere uguali a quelli associati alla richiesta in fase di elaborazione. Per aggirare questo possibile rischio di sicurezza, i programmi di query potrebbero utilizzare l'opzione MQPMO_ALTERNATE_USER_AUTHORITY quando inseriscono le loro risposte. Ciò significa che ogni singolo utente di MVB1 necessita dell'autorità per aprire B2.REPLY.n code.

Utilizzo degli identificativi di messaggio e correlazione in CAM

L'applicazione deve monitorare l'avanzamento di tutte le richieste in tempo reale che sta elaborando in qualsiasi momento. A tale scopo, utilizza l'identificativo di messaggio univoco di ciascun messaggio di richiesta di prestito per associare tutte le informazioni che ha su ciascuna richiesta.

Il CAM copia il `MsgId` del messaggio di interrogazione nel `CorrelId` di tutti i messaggi di richiesta inviati per tale interrogazione. Gli altri programmi nell'esempio (CSQ4CVB3 - 5) copiano il `CorrelId` di ogni messaggio che ricevono nel `CorrelId` del relativo messaggio di risposta.

z/OS *L'esempio Controllo credito con più gestori code su z/OS*

È possibile utilizzare l'applicazione di esempio Controllo del credito per dimostrare l'accodamento distribuito installando l'esempio su due gestori code e sistemi CICS (con ciascun gestore code connesso a un sistema CICS differente).

Quando il programma di esempio è installato e il controllo trigger (CKTI) è in esecuzione su ciascun sistema, è necessario:

1. Impostare il collegamento di comunicazione tra i due gestori code. Per informazioni su come eseguire questa operazione, consultare Configurazione dell'accodamento distribuito.
2. Su un gestore code, creare una definizione locale per ciascuna delle code remote (sull'altro gestore code) che si desidera utilizzare. Queste code possono essere una qualsiasi di CSQ4SAMP.B *n*.MESSAGES, dove *n* è 3, 5, 6 o 7. (Queste sono le code servite dal programma checking-account e dal programma agency - query.) Per informazioni su come effettuare questa operazione, vedere DEFINE QREMOTE e DEFINE queues.
3. Modificare la definizione dell'elenco nomi (CSQ4SAMP.B4.NAMELIST) in modo che contenga i nomi delle code remote che si desidera utilizzare. Per informazioni su come eseguire questa operazione, vedere DEFINE NAMELIST.

z/OS *L'estensione IMS all'esempio Controllo del credito su z/OS*

Una versione del programma di controllo viene fornita come un programma BMP (batch message processing) IMS . È scritto in linguaggio C.

Il programma esegue la stessa funzione della versione CICS , tranne per il fatto che per ottenere le informazioni sull'account, il programma legge un database IMS invece di un file VSAM. Se si sostituisce la versione CICS del programma di controllo con la versione IMS , non si vede alcuna differenza nel metodo di utilizzo dell'applicazione.

Per preparare ed eseguire la versione IMS , è necessario:

1. Segui la procedura descritta in “Preparazione ed esecuzione dell'esempio Controllo credito su z/OS” a pagina 1200.
2. Segui la procedura descritta in “Preparazione dell'applicazione di esempio per l'ambiente IMS su z/OS” a pagina 1180.
3. Modificare la definizione della coda alias CSQ4SAMP.B2.OUTPUT.ALIAS per risolvere la coda CSQ4SAMP.B3.IMS.MESSAGES (invece di CSQ4SAMP.B3.MESSAGES). A tale scopo, è possibile utilizzare una delle seguenti operazioni:
   - Le operazioni e i pannelli di controllo IBM MQ for z/OS
   - Il comando ALTER QALIAS .

Un altro modo per utilizzare il programma di account di controllo IMS è quello di rendere disponibile una delle code che riceve i messaggi dal programma di distribuzione. Nel modulo fornito dell'applicazione di esempio Credit Check, sono presenti tre di queste code (B5/6/7.MESSAGES), tutti serviti dal programma agency - query. Questo programma ricerca un dataset VSAM. Per confrontare l'utilizzo del dataset VSAM e del database IMS , è possibile che il programma di account di controllo IMS serva una di queste code. Per fare ciò, è necessario modificare la definizione dell'elenco nomi CSQ4SAMP.B4.NAMELIST per sostituire uno dei CSQ4SAMP.B *n*.MESSAGES accoda con CSQ4SAMP.B3.IMS.Coda MESSAGES. È possibile utilizzare uno dei seguenti:

- Le operazioni e i pannelli di controllo IBM MQ for z/OS
- Il comando ALTER NAMELIST .

È quindi possibile eseguire l'esempio dalla transazione CICS MVB1. L'utente non vede alcuna differenza nell'operazione o nella risposta. IMS BMP si arresta dopo aver ricevuto un messaggio di arresto o dopo essere stato inattivo per cinque minuti.

Progettazione del programma di controllo - account IMS (CSQ4ICB3)

Questo programma viene eseguito come BMP. Avviare il programma utilizzando il relativo JCL prima che gli vengano inviati tutti i messaggi IBM MQ .

Il programma ricerca in un database IMS il numero di conto nei messaggi di richiesta di prestito. Richiama il nome conto corrispondente, il saldo medio e l'indice di affidabilità creditizia.

Il programma invia i risultati della ricerca del database alla coda di risposta indicata nel messaggio IBM MQ in fase di elaborazione. Il messaggio restituito accoda il tipo di account e i risultati della ricerca al messaggio ricevuto in modo che la transazione che crea la risposta possa confermare che la query corretta è in fase di elaborazione. Il messaggio è sotto forma di tre gruppi di 79 caratteri, come segue:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

Quando viene eseguito come BMP orientato ai messaggi, il programma scarica la coda messaggi IMS , quindi legge i messaggi dalla coda IBM MQ for z/OS e li elabora. Non vengono ricevute informazioni dalla coda messaggi IMS . Il programma si riconnette al gestore code dopo ogni checkpoint perché gli handle sono stati chiusi.

Quando viene eseguito in un BMP orientato al batch, il programma continua ad essere connesso al gestore code dopo ogni punto di controllo poiché gli handle non vengono chiusi.

z/OS *L'esempio Gestore messaggi su z/OS*

L'applicazione TSO di esempio Gestore messaggi consente di sfogliare, inoltrare ed eliminare i messaggi su una coda. L'esempio è disponibile in C e COBOL.

Preparazione ed esecuzione dell'esempio

Eseguire queste operazioni:

1. Preparare l'esempio come descritto in “Preparazione di applicazioni di esempio per l'ambiente TSO su z/OS” a pagina 1174.
2. Personalizzare il CLIST (CSQ4RCH1) fornito per l'esempio per definire l'ubicazione dei pannelli, il percorso del file di messaggi e il percorso dei moduli di caricamento.

È possibile utilizzare CLIST CSQ4RCH1 per eseguire sia la versione C che COBOL dell'esempio. La versione fornita di CSQ4RCH1 esegue la versione C e contiene istruzioni sulla personalizzazione necessaria per la versione COBOL.

Nota:

1. Non sono presenti definizioni di coda di esempio fornite con l'esempio.
2. VS COBOL II non supporta la funzione multitasking con ISPF, pertanto non utilizzare l'applicazione di esempio Gestore messaggi su entrambi i lati di uno schermo suddiviso. In tal caso, i risultati sono imprevedibili.

z/OS *Utilizzo dell'esempio Gestore messaggi su z/OS*

Dopo aver installato l'esempio e averlo richiamato dal CLIST personalizzato CSQ4RCH1, viene visualizzata la schermata mostrata in Figura 146 a pagina 1213 .

```
---------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :


Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Figura 146. Schermata iniziale per l'esempio Gestore messaggi

Immettere il gestore code e il nome della coda da visualizzare (sensibile al maiuscolo / minuscolo) e viene visualizzata la schermata dell'elenco dei messaggi (consultare Figura 147 a pagina 1213).

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

Figura 147. Schermata Elenco messaggi per l'esempio Gestore messaggi

Questo pannello mostra i primi 99 messaggi sulla coda e, per ognuno di essi, mostra i seguenti campi:

Numero messaggio
Numero messaggio

Data di inserimento MM/GG/AAAA
La data in cui il messaggio è stato inserito nella coda (GMT)

Ora di inserimento HH:MM: SS
L'ora in cui il messaggio è stato inserito nella coda (GMT)

Nome formato
MQMD.Format

Identificativo utente
MQMD.UserIdentifier

Tipo applicazione di immissione
MQMD.PutApplType

Inserisci nome dell'applicazione
MQMD.PutApplName

Viene visualizzato anche il numero totale di messaggi sulla coda.

Da questo pannello è possibile scegliere un messaggio, per numero e non per posizione del cursore, e quindi visualizzarlo. Per un esempio, consultare Figura 148 a pagina 1214.

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                       :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
--------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount   : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                 `
ReplyToQMgr   : `VM03                       `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                            `
PutApplType   : 000000002
PutApplName   : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .............`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

Figura 148. Viene visualizzato il messaggio scelto

Una volta che il messaggio è stato visualizzato, può essere eliminato, lasciato sulla coda o inoltrato a un'altra coda. I campi `Forward to Q Mgr` e `Forward to Queue` sono inizializzati con valori da MQMD, che possono essere modificati prima di inoltrare il messaggio.

La progettazione di esempio consente di selezionare e visualizzare solo i messaggi con combinazioni `MsgId / CorrelId` univoche, poiché il messaggio viene richiamato utilizzando `MsgId` e `CorrelId` come chiave. Se la chiave non è univoca, l'esempio non può richiamare il messaggio scelto con certezza.

Nota: Quando si utilizza l'esempio SCSQCLST (CSQ4RCH1) per sfogliare i messaggi, ogni richiamo aumenta il conteggio di backout del messaggio. Se si desidera modificare il comportamento di questo esempio, copiare l'esempio e modificarne il contenuto come necessario. Tenere presente che altre applicazioni che si basano su questo conteggio di backout possono essere influenzate da questo conteggio crescente.

z/OS *Progettazione dell'esempio Gestore messaggi su z/OS*

Questo argomento descrive la progettazione di ciascuno dei programmi che costituiscono l'applicazione di esempio Gestore messaggi.

Programma di convalida oggetto

Ciò richiede un coda e un gestore code validi.

Se non si specifica un nome di gestore code, viene utilizzato il gestore code predefinito, se disponibile. È possibile utilizzare solo code locali; viene emesso un MQINQ per controllare che il tipo di coda e un errore vengano riportati se la coda non è locale. Se la coda non viene aperta correttamente o se la chiamata MQGET è inibita sulla coda, vengono restituiti messaggi di errore che indicano il CompCode e il codice di ritorno Reason.

Programma elenco messaggi

In questo modo viene visualizzato un elenco di messaggi su una coda con informazioni su di essi, ad esempio la data di immissione, l'ora di immissione e il formato del messaggio.

Il numero massimo di messaggi memorizzati nell'elenco è 99. Se ci sono più messaggi sulla coda di questo, viene visualizzata anche la profondità della coda corrente. Per scegliere un messaggio da visualizzare, immettere il numero del messaggio nel campo di immissione (il valore predefinito è 01). Se la voce non è valida, si riceve un messaggio di errore appropriato.

Programma di contenuto del messaggio

Visualizza il contenuto del messaggio.

Il contenuto viene formattato e suddiviso in due parti:

1. Descrittore messaggio
2. Buffer del messaggio

Il descrittore del messaggio visualizza il contenuto di ciascun campo su una riga separata.

Il buffer di messaggi è formattato in base al contenuto. Se il buffer contiene un'intestazione di lettera non recapitabile (MQDLH) o un'intestazione della coda di trasmissione (MQXQH), queste vengono formattate e visualizzate prima del buffer stesso.

Prima di formattare i dati del buffer, una riga del titolo mostra la lunghezza del buffer del messaggio in byte. La dimensione massima del buffer è di 32768 byte e qualsiasi messaggio più lungo viene troncato. La dimensione completa del buffer viene visualizzata insieme a un messaggio che indica che vengono visualizzati solo i primi 32768 byte del messaggio.

I dati del buffer sono formattati in due modi:

1. Una volta stampato lo scostamento nel buffer, i dati del buffer vengono visualizzati in formato esadecimale.
2. I dati del buffer vengono quindi visualizzati di nuovo come valori EBCDIC. Se un valore EBCDIC non può essere stampato, viene invece stampato un punto (.).

È possibile immettere D per l'eliminazione o F per l'inoltro nel campo azione. Se si sceglie di inoltrare il messaggio, `forward-to queue` e `queue manager name` devono essere impostati correttamente. I valori predefiniti per questi campi vengono letti dai campi del gestore code ReplyToQ e ReplyTo.

Se si inoltra un messaggio, qualsiasi blocco di intestazione memorizzato nel buffer viene eliminato. Se il messaggio viene inoltrato correttamente, viene rimosso dalla coda originale. Se si immettono azioni non valide, vengono visualizzati messaggi di errore.

È inoltre disponibile un pannello di aiuto di esempio denominato CSQ4CHP9 .

z/OS *L'esempio Put asincrono su z/OS*

Il programma di esempio Asynchronous Put inserisce i messaggi su una coda utilizzando la chiamata MQPUT asincrona. L'esempio richiama anche le informazioni sullo stato utilizzando la chiamata MQSTAT.

Le applicazioni Put asincrone utilizzano queste chiamate MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

I programmi di esempio vengono forniti nel linguaggio di programmazione C.

Le applicazioni Put asincrone vengono eseguite nell'ambiente batch. Consultare Altri esempi per le applicazioni batch.

Questo argomento fornisce inoltre informazioni sulla progettazione del programma di consumo asincrono ed esegue l'esempio CSQ4BCS2 .

- “Esecuzione dell'esempio CSQ4BCS2” a pagina 1216
- “Progettazione del programma di esempio Put asincrono” a pagina 1216

Esecuzione dell'esempio CSQ4BCS2

Questo programma di esempio utilizza fino a sei parametri:

1. Il nome della coda di destinazione (obbligatorio).
2. Il nome del gestore code (facoltativo).
3. Opzioni di apertura (facoltativo).
4. Opzioni di chiusura (facoltative).
5. Il nome del gestore code di destinazione (facoltativo).
6. Il nome della coda dinamica (facoltativo).

Se un gestore code non è specificato, CSQ4BCS2 si connette al gestore code predefinito. Il contenuto del messaggio viene fornito tramite l'input standard (**SYSIN DD**).

Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BCSP.

Progettazione del programma di esempio Put asincrono

Il programma utilizza la chiamata MQOPEN con le opzioni di output fornite o con le opzioni MQOO_OUTPUT e MQOO_FAIL_IF_QUIESCING per aprire la coda di destinazione per l'inserimento dei messaggi.

Se il programma non riesce ad aprire la coda, emette un messaggio di errore contenente il codice motivo restituito dalla chiamata MQOPEN. Per mantenere il programma semplice in questa e nelle chiamate MQI successive, vengono utilizzati i valori predefiniti per molte delle opzioni.

Per ogni riga di input, il programma legge il testo in un buffer e utilizza la chiamata MQPUT con MQPMO_ASYNC_RESPONSE per creare un messaggio datagramma contenente il testo di tale riga e inserisce in maniera asincrona il messaggio sulla coda di destinazione. Il programma continua fino a quando non raggiunge la fine dell'input o fino a quando la chiamata MQPUT non ha esito negativo. Se il programma raggiunge la fine dell'input, chiude la coda utilizzando la chiamata MQCLOSE.

Il programma quindi emette la chiamata MQSTAT che restituisce una struttura MQSTS e visualizza i messaggi contenenti il numero di messaggi immessi correttamente, il numero di messaggi immessi con un'avvertenza e il numero di errori.

Nota: Per osservare cosa accade quando viene rilevato un errore MQPUT dalla chiamata MQSTAT, impostare MAXDEPTH sulla coda di destinazione su un valore basso.

z/OS *L'esempio di consumo asincrono batch su z/OS*

Il programma di esempio CSQ4BCS1 viene consegnato in C, dimostra l'utilizzo di MQCB e MQCTL per utilizzare i messaggi da più code in modo asincrono.

Gli esempi di consumo asincrono vengono eseguiti nell'ambiente batch. Consultare Altri esempi per le applicazioni batch.

Esiste anche un esempio COBOL che viene eseguito nell'ambiente CICS , consultare “L'esempio CICS Utilizzo asincrono e pubblicazione / sottoscrizione su z/OS” a pagina 1218.

Le applicazioni utilizzano queste chiamate MQI:

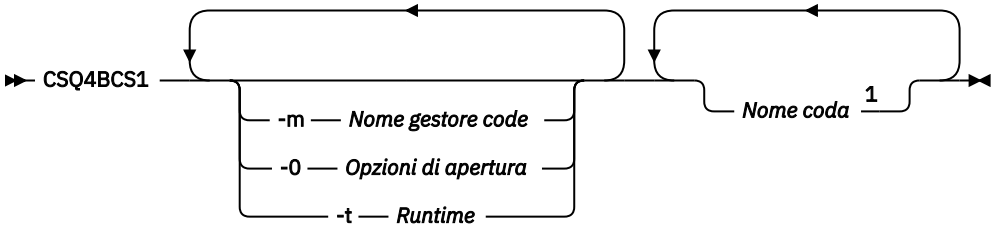
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

Questo argomento fornisce anche informazioni sulle seguenti intestazioni:

- “Esecuzione dell'esempio CSQ4BCS1” a pagina 1217
- “Progettazione del programma di esempio Batch Asynchronous Consumption” a pagina 1217

Esecuzione dell'esempio CSQ4BCS1

Questo programma di esempio segue la seguente sintassi:



Note:

[1] Questo esempio supporta un numero massimo di 10 code. Fornire più di un nome coda per leggere i messaggi da più code.

Esiste un JCL di esempio per eseguire questo programma, che risiede in CSQ4BCSC.

Progettazione del programma di esempio Batch Asynchronous Consumption

L'esempio mostra come leggere i messaggi da più code in ordine di arrivo. Ciò richiederebbe più codice utilizzando MQGET sincrono. Con l'utilizzo asincrono, non è richiesto alcun polling e la gestione del thread e della memoria viene eseguita da IBM MQ. Nel programma di esempio, gli errori vengono scritti nella console.

Il codice di esempio ha i passi seguenti:

1. Definire la funzione callback di consumo del singolo messaggio.

```
void MessageConsumer(MQHCONN   hConn,
```

```
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Connettersi al gestore code.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Aprire le code di input e associare ciascuna coda alla funzione di callback `MessageConsumer` .

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` non deve essere impostato per ciascuna coda; è un campo di sola immissione. È possibile associare una funzione di callback differente a ciascuna coda.

4. Avviare l'utilizzo dei messaggi.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Attendere che l'utente prema Invio, quindi arrestare l'utilizzo dei messaggi.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Infine, disconnettersi dal gestore code.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *L'esempio CICS Utilizzo asincrono e pubblicazione / sottoscrizione su z/OS*

I programmi di esempio Asynchronous Consumption and Publish / Subscribe dimostrano l'uso del consumo asincrono e le funzioni di pubblicazione e sottoscrizione all'interno di CICS.

Un programma *Client di registrazione* registra tre gestori callback (un gestore eventi e due utenti di messaggi) e avvia il consumo asincrono. Un programma *Client di messaggistica* inserisce i messaggi in una coda o pubblica i messaggi adatti da una console CICS per l'utilizzo da parte dei due utenti dei messaggi (CSQ4CVCN e CSQ4CVCT).

Per fornire il controllo di runtime sul comportamento dell'esempio, uno dei consumatori di messaggi può essere istruito utilizzando i messaggi che riceve, per SUSPEND, RESUME o DEREGISTER uno dei gestori di callback. Può anche essere utilizzato per emettere un MQCTL STOP per terminare il consumo asincrono sotto controllo. L'altro utente del messaggio è registrato per sottoscrivere un argomento.

Ogni programma emette istruzioni COBOL DISPLAY in punti appropriati per visualizzare il comportamento dell'esempio.

Le applicazioni utilizzano queste chiamate MQI:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

I programmi vengono forniti in linguaggio COBOL. Consultare CICS Esempi di utilizzo e pubblicazione / sottoscrizione asincroni per le applicazioni CICS .

Questo argomento fornisce anche le seguenti informazioni:

- "Configura" a pagina 1219
- "Client di registrazione CSQ4CVRG" a pagina 1219
- "Gestore eventi CSQ4CVEV" a pagina 1219
- "Utente messaggio semplice CSQ4CVCN" a pagina 1219
- "Utente messaggio di controllo CSQ4CVCT" a pagina 1219
- "Client di messaggistica CSQ4CVPT" a pagina 1220

Configura

I nomi della coda e dell'argomento utilizzati dai consumatori di messaggi sono codificati nei programmi client di registrazione e messaggistica.

La coda, **SAMPLE.CONTROL.QUEUE**, deve essere definito sul gestore code associato alla region CICS prima di eseguire l'esempio. L'argomento, **Notizie / Media / Film**, può essere definito se necessario oppure viene creato al runtime nell'oggetto di gestione predefinito, se non esiste.

I programmi CICS e le definizioni di transazione possono essere installati installando un gruppo: CSQ4SAMP.

Client di registrazione CSQ4CVRG

Il programma client di registrazione deve essere avviato nella transazione CICS MVRG. Non richiede alcun input.

Quando viene avviato, il client di registrazione registra i seguenti gestori callback utilizzando MQCB:

- CSQ4CVEV come gestore eventi.
- CSQ4CVCN come Message Consumer su un argomento, **Notizie / Media / Film**.
- CSQ4CVCT come utente di messaggi su una coda, **SAMPLE.CONTROL.QUEUE**.

Il client di registrazione passa una struttura dati contenente i nomi di tutti e tre i gestori callback registrati a CSQ4CVCT, insieme agli handle dell'oggetto associati ai due consumer di messaggi.

Avendo registrato i gestori di callback, il client di registrazione emette un MQCTL START_WAIT per avviare il consumo asincrono e sospendere fino a quando non viene restituito il controllo (ad esempio, da uno dei gestori di callback che emette un MQCTL STOP).

Gestore eventi CSQ4CVEV

Quando è guidato, il Gestore eventi visualizza un messaggio che indica il tipo di chiamata (ad esempio, START). Quando viene eseguito per il codice motivo IBM MQ CONNECTION_QUIESCING, il gestore eventi emette un STOP MQCTL per terminare il consumo asincrono e restituire il controllo al client di registrazione.

Utente messaggio semplice CSQ4CVCN

Quando gestito, questo utente del messaggio visualizza un messaggio che indica il tipo di chiamata (ad esempio, REGISTER). Quando viene utilizzato per il tipo di chiamata MSG_REMOVED, il Message Consumer richiama il messaggio in ingresso e lo invia al log del lavoro CICS .

Utente messaggio di controllo CSQ4CVCT

Quando gestito, questo utente del messaggio visualizza un messaggio che indica il tipo di chiamata (ad esempio, START). Quando viene utilizzato per il tipo di chiamata MSG_REMOVED, il Message Consumer

richiama il messaggio in entrata e la struttura di dati passati dal client di registrazione. In base al contenuto del messaggio, emette i comandi MQCB o MQCTL appropriati per uno dei seguenti:

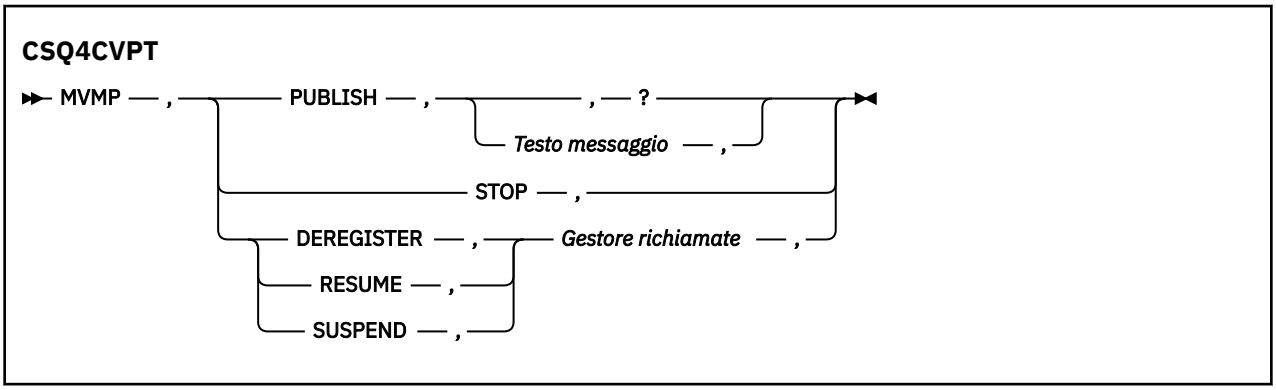
- Arrestare il consumo asincrono (restituendo il controllo al client di registrazione).
- SUSPEND, RESUME o DEREGISTER un gestore callback denominato (incluso se stesso).

Client di messaggistica CSQ4CVPT

Il client di messaggistica ha due funzioni:

- Pubblica un messaggio in un topic per l'utilizzo da parte del Message Consumer CSQ4CVCN.
- Inserisce un messaggio di controllo in una coda per l'utilizzo da parte del consumer del messaggio di controllo CSQ4CVCT, determinando una potenziale modifica del comportamento dell'esempio.

Il programma del client di messaggistica deve essere avviato da una console CICS in una transazione CICS e utilizza l'input della riga comandi con la sintassi seguente:



PUBLISH
Pubblicare il testo del messaggio (o un messaggio predefinito) come messaggio di conservazione per l'utilizzo da parte del Simple Message Consumer.

ARRESTA
Arresta consumo asincrono.

DERISTER
Annullare la registrazione del gestore callback denominato.

RIPRENDI
Riprendere il gestore callback denominato.

SUSPEND
Sospendere il gestore callback denominato.

I campi di input sono posizionali e separati da virgole. Le parole chiave e i nomi del gestore richiamate non sono sensibili al maiuscolo / minuscolo.

Esempi:

Tabella 186. Esempi di input

Esempio	**Descrizione**
MVMP, PUBBLICA,,	Pubblica un messaggio predefinito
MVMP, pubblica, `A short message`,	Pubblica il testo fornito
MVMP, STOP,	Arresta consumo asincrono
MVMP,DEREGISTER,CSQ4CVEV,	Annulla la registrazione del gestore eventi
MVMP,resume,csq4cvcn,	Riprendi il consumer del messaggio semplice
MVMP,SUSPEND,CSQ4CVEV,	Sospendi il gestore eventi

dove MVMP è la transazione CICS associata al programma client di messaggistica CSQ4CVPT.

Nota:

- La sospensione o l'annullamento della registrazione di tutti i gestori di callback termina il START_WAIT emesso dal client di registrazione, restituendo il controllo ad esso e terminando l'attività.
- La sospensione o l'annullamento della registrazione del gestore callback di controllo non è stata deliberatamente impedita, ma rimuove la possibilità di controllare ulteriormente il comportamento dell'esempio.

z/OS *L'esempio di pubblicazione / sottoscrizione su z/OS*

I programmi di esempio di pubblicazione / sottoscrizione dimostrano l'utilizzo delle funzioni di pubblicazione e sottoscrizione in IBM MQ.

Esistono quattro programmi di esempio di linguaggio di programmazione C e due COBOL che dimostrano come programmare l'interfaccia di pubblicazione / sottoscrizione IBM MQ . I programmi vengono forniti in linguaggio C e COBOL. Le applicazioni vengono eseguite nell'ambiente batch; consultare Esempi di pubblicazione / sottoscrizione per le applicazioni batch.

Esistono anche esempi COBOL eseguiti nell'ambiente CICS ; consultare “L'esempio CICS Utilizzo asincrono e pubblicazione / sottoscrizione su z/OS” a pagina 1218.

Questo argomento fornisce anche informazioni su come eseguire i programmi di esempio di pubblicazione / sottoscrizione. Questi programmi di esempio includono:

- “Esecuzione dell'esempio CSQ4BCP1” a pagina 1221
- “Esecuzione dell'esempio CSQ4BCP2” a pagina 1221
- “Esecuzione dell'esempio CSQ4BCP3” a pagina 1222
- “Esecuzione dell'esempio CSQ4BCP4” a pagina 1222
- “Esecuzione dell'esempio CSQ4BVP1” a pagina 1222
- “Esecuzione dell'esempio CSQ4BVP2” a pagina 1223

Esecuzione dell'esempio CSQ4BCP1

Questo programma è scritto in C; pubblica messaggi su un argomento. Avviare uno degli esempi del sottoscrittore prima di eseguire questo programma.

Questo programma utilizza fino a quattro parametri:

1. Il nome della stringa dell'argomento di destinazione (obbligatorio).
2. Il nome del gestore code (facoltativo).
3. Opzioni di apertura (facoltativo).
4. Opzioni di chiusura (facoltative).

Se un gestore code non è specificato, CSQ4BCP1 si connette a quello predefinito. Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BCPP.

Il contenuto del messaggio viene fornito tramite l'input standard (**SYSIN DD**).

Esecuzione dell'esempio CSQ4BCP2

Questo programma è scritto in C; si iscrive a un argomento e stampa i messaggi ricevuti.

Questo programma utilizza fino a tre parametri:

1. Il nome della stringa dell'argomento di destinazione (obbligatorio).
2. Il nome del gestore code (facoltativo).
3. Opzioni di sottoscrizione MQSD (facoltativo).

Se un gestore code non è specificato, CSQ4BCP2 si connette a quello predefinito. Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BCPS.

Esecuzione dell'esempio CSQ4BCP3

Questo programma è scritto in C; sottoscrive un argomento utilizzando una coda di destinazione specificata dall'utente e stampa i messaggi ricevuti.

Questo programma utilizza fino a quattro parametri:

1. Il nome della stringa dell'argomento di destinazione (obbligatorio).
2. Il nome della destinazione (obbligatorio).
3. Il nome del gestore code (facoltativo).
4. Opzioni di sottoscrizione MQSD (facoltativo).

Se un gestore code non è specificato, CSQ4BCP3 si connette a quello predefinito. Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BCPD.

Esecuzione dell'esempio CSQ4BCP4

Questo programma è scritto in C; esso sottoscrive e riceve i messaggi da un argomento che consente l'utilizzo delle opzioni estese sulla chiamata MQSUB, estendendo quelle disponibili sull'esempio MQSUB più semplice: CSQ4BCP2. Oltre al payload del messaggio, vengono ricevute e visualizzate le proprietà del messaggio per ciascun messaggio.

Questo programma utilizza una serie variabile di parametri:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  Importante: Uno tra **-t** o **-o**, o entrambi, è obbligatorio
- **-m** *Queue manager name* (facoltativo).
- **-b** *Connection binding type* (facoltativo), dove *tipo* può avere uno dei seguenti valori:
  - *standard* : MQCNO_STANDARD_BINDING, che è il valore predefinito
  - *condiviso*: MQCNO_SHARED_BINDING
  - *fastpath*: MQCNO_FASTPATH_BINDING
  - *isolato*: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (facoltativo).
- **-w** *Wait interval on MQGET in seconds* (facoltativo), dove *secondi* può avere uno dei seguenti valori:
  - unlimited: MQWI_UNLIMITED
  - none: nessuna attesa
  - *n*: intervallo di attesa in secondi
  - Nessun valore specificato: quando non viene specificato alcun valore, il valore predefinito è 30 secondi
- **-d** *Subscription name* (facoltativo). Crea o riprende la sottoscrizione durevole denominata.
- **-k** (facoltativo). Mantiene la sottoscrizione durevole su MQCLOSE.

Se non viene specificato un gestore code, CSQ4BCP4 si connette a quello predefinito. Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BCPE.

Esecuzione dell'esempio CSQ4BVP1

Questo programma è scritto in COBOL, pubblica i messaggi su un argomento. Avviare uno degli esempi del sottoscrittore prima di eseguire questo programma.

Questo programma non accetta parametri. **SYSIN DD** fornisce il nome dell'argomento di immissione, il nome del gestore code e il contenuto del messaggio.

Se non viene specificato un gestore code, CSQ4BVP1 si connette a quello predefinito. Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BVPP.

Esecuzione dell'esempio CSQ4BVP2

Questo programma è scritto in COBOL, si iscrive a un argomento e stampa i messaggi ricevuti.

Questo programma non accetta parametri. **SYSIN DD** fornisce l'input per il nome argomento e il nome gestore code.

Se non viene specificato un gestore code, CSQ4BVP1 si connette a quello predefinito. Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BVPP.

z/OS *L'esempio di proprietà del messaggio Set and Inquire su z/OS*

I programmi di esempio delle proprietà del messaggio dimostrano l'aggiunta di proprietà definite dall'utente a un handle del messaggio e l'interrogazione delle proprietà associate a tale messaggio.

Le applicazioni utilizzano queste chiamate MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

I programmi vengono forniti in linguaggio C. Le applicazioni vengono eseguite nell'ambiente batch. Consultare Altri esempi per le applicazioni batch.

Il programma CSQ4BCM1 viene utilizzato per analizzare le proprietà di un handle del messaggio da una coda messaggi ed è un esempio dell'utilizzo della chiamata API MQINQMP. L'esempio richiama un messaggio da una coda e quindi stampa tutte le proprietà dell'handle del messaggio.

Il programma CSQ4BCM2 è utilizzato per impostare le proprietà di un gestore messaggi su una coda messaggi ed è un esempio dell'utilizzo della chiamata API MQSETMP. L'esempio crea un handle del messaggio e lo inserisce nel campo `MsgHandle` della struttura MQGMO. Quindi inserisce il messaggio in una coda.

Altri esempi di analisi e stampa delle proprietà dei messaggi sono inclusi nei programmi di esempio CSQ4BCG1 e CSQ4BCP4 .

Questo argomento fornisce anche informazioni sull'esecuzione degli esempi di proprietà del messaggio Set and Inquire nelle intestazioni riportate di seguito:

- “Esecuzione dell'esempio CSQ4BCM1” a pagina 1223
- “Esecuzione dell'esempio CSQ4BCM2” a pagina 1224

Esecuzione dell'esempio CSQ4BCM1

Questo programma utilizza fino a quattro parametri:

1. Il nome della coda di destinazione (obbligatorio).
2. Il nome del gestore code (facoltativo).

3. Opzioni di apertura (facoltativo).
4. Opzioni di chiusura (facoltative).

Esecuzione dell'esempio CSQ4BCM2

Questo programma utilizza fino a sei parametri:

1. Il nome della coda di destinazione (obbligatorio).
2. Il nome del gestore code (facoltativo).
3. Opzioni di apertura (facoltativo).
4. Opzioni di chiusura (facoltative).
5. Il nome del gestore code di destinazione (facoltativo).
6. Il nome della coda dinamica (facoltativo).

I nomi proprietà, i valori e il contenuto del messaggio vengono forniti tramite l'input standard (**SYSIN DD**). Esiste un JCL di esempio per eseguire il programma, che risiede in CSQ4BCMP.

Sviluppo di applicazioni per Managed File Transfer

Specificare i programmi da eseguire con Managed File Transfer, utilizzare Apache Ant con Managed File Transfer, personalizzare Managed File Transfer con uscite utente e controllare Managed File Transfer inserendo i messaggi nella coda comandi dell'agent.

Specifica dei programmi da eseguire con MFT

È possibile eseguire programmi su un sistema in cui è in esecuzione un Managed File Transfer Agent . Come parte di una richiesta di trasferimento file, è possibile specificare un programma da eseguire prima o dopo l'inizio di un trasferimento. Inoltre, è possibile avviare un programma che non fa parte di una richiesta di trasferimento file inoltrando una chiamata gestita.

Informazioni su questa attività

Esistono cinque scenari in cui è possibile specificare un programma da eseguire:

- Come parte di una richiesta di trasferimento, presso l'agent di origine, prima dell'avvio del trasferimento.
- Come parte di una richiesta di trasferimento, nell'agente di destinazione, prima dell'avvio del trasferimento.
- Come parte di una richiesta di trasferimento, nell'agent di origine, dopo il completamento del trasferimento.
- Come parte di una richiesta di trasferimento, nell'agente di destinazione, dopo il completamento del trasferimento.
- Non come parte di una richiesta di trasferimento. È possibile inoltrare una richiesta a un agent per eseguire un programma. Questo scenario è a volte indicato come chiamata gestita.

Le uscite utente e le chiamate del programma vengono richiamate nel seguente ordine:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Note:

1. Il **DestinationTransferEndExits** viene eseguito solo quando il trasferimento viene completato, con esito positivo o parziale.
2. Il **postDestinationCall** viene eseguito solo quando il trasferimento viene completato, con esito positivo o parziale.
3. **SourceTransferEndExits** viene eseguito per trasferimenti riusciti, parzialmente riusciti o non riusciti.
4. **postSourceCall** viene richiamato solo se:
   - Il trasferimento non è stato annullato.
   - C'è un esito positivo o parzialmente positivo.
   - Tutti i programmi di trasferimento post - destinazione sono stati eseguiti correttamente.

Procedura

- Specificare il programma che si desidera eseguire utilizzando una delle seguenti opzioni:

  Utilizzare un'attività Apache Ant

  Utilizzare una delle attività `fte:filecopy`, `fte:filemovee` `fte:call` Ant per avviare un programma. Utilizzando un'attività Ant , è possibile specificare un programma in uno qualsiasi dei cinque scenari, utilizzando gli elementi nidificati `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrce` `fte:command` . Per ulteriori informazioni, vedere Elementi nidificati del richiamo del programma.

  Modifica il messaggio di richiesta di trasferimento file

  È possibile modificare l'XML generato da una richiesta di trasferimento. Utilizzando questo metodo, è possibile eseguire un programma in uno qualsiasi dei cinque scenari, aggiungendo elementi **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall**e **managedCall** al file XML. Utilizzare quindi questo file XML modificato come definizione di trasferimento per una nuova richiesta di trasferimento file, ad esempio con il parametro **fteCreateTransfer -td** . Per ulteriori informazioni, consultare Esempi di messaggi di richiesta chiamata agent MFT.

  Utilizzare il comando `fteCreateTransfer`

  È possibile utilizzare il comando **`fteCreateTransfer`** per specificare i programmi da avviare. È possibile utilizzare il comando per specificare i programmi da eseguire nei quattro primi scenari, come parte di una richiesta di trasferimento, ma non è possibile avviare una chiamata gestita. Per informazioni sui parametri da utilizzare, consultare **`fteCreateTransfer`**: avvio di un nuovo trasferimento file. Per esempi di utilizzo di questo comando, consultare Esempi di utilizzo di fteCreateTransfer to start programs.

Riferimenti correlati

proprietà commandPath MFT

Chiamate gestite

Gli agent Managed File Transfer (MFT) vengono generalmente utilizzati per trasferire file o messaggi. Questi sono noti come *Trasferimenti gestiti*. Gli agenti possono essere utilizzati anche per eseguire comandi, script o JCL senza la necessità di trasferire file o messaggi. Questa funzione è nota come *Chiamate gestite*.

Le richieste di chiamate gestite possono essere inoltrate a un agent in diversi modi:

- Utilizzo dell' attività fte: call Ant.
- Configurazione di un monitoraggio risorse con un XML di attività che esegue un comando o uno script. Per ulteriori informazioni, consultare Configurazione delle attività di monitoraggio per avviare comandi e script .

- Inserimento diretto di un messaggio XML nella coda comandi dell'agent. Consultare Formato del messaggio di richiesta di trasferimento file per ulteriori dettagli sullo schema XML della chiamata gestita.

Per le chiamate gestite, la directory contenente il comando o lo script in esecuzione deve essere specificata nella proprietà dell'agent **commandPath**.

Le chiamate gestite non possono eseguire comandi o script che si trovano in directory non specificate nel **commandPath**dell'agent. Ciò per garantire che l'agent non esegua alcun codice dannoso.

Importante: Per assicurarsi che questo sia il caso, per impostazione predefinita, quando si specifica **commandPath**:

- Qualsiasi sandbox dell'agent esistente viene configurato dall'agent quando viene avviato in modo che tutte le directory **commandPath** vengano aggiunte automaticamente all'elenco di directory che hanno negato l'accesso per un trasferimento.
- Tutte le sandbox utente esistenti vengono aggiornate quando l'agent viene avviato in modo che tutte le directory **commandPath** (e le relative sottodirectory) vengano aggiunte come elementi `<exclude>` agli elementi `<read>` e `<write>` .
- Se l'agente non è configurato per utilizzare una sandbox dell'agente o una sandbox dell'utente, una nuova sandbox dell'agente viene creata all'avvio dell'agente che ha le directory **commandPath** specificate come directory negate.

Inoltre, è anche possibile abilitare il controllo delle autorizzazioni su un agent per garantire che solo gli utenti autorizzati siano autorizzati a inoltrare richieste di chiamata gestite. Per ulteriori informazioni, consultare Limitazione delle autorizzazioni utente sulle azioni dell'agent MFT.

Il comando, lo script o il JCL richiamato come parte di una chiamata gestita viene eseguito come un processo esterno, monitorato dall'agent. Quando il processo termina, la chiamata gestita viene completata e il codice di ritorno del processo viene reso disponibile all'agente o allo script Ant che ha richiamato l'attività **fte:call** Ant .

Se la chiamata gestita è stata avviata dall'attività **fte:call** Ant , lo script Ant può controllare il valore del codice di ritorno per stabilire se la chiamata gestita ha avuto esito positivo o meno.

Per tutti gli altri tipi di chiamate gestite, è possibile specificare quali valori di codice di ritorno devono essere utilizzati per indicare che la chiamata gestita è stata completata correttamente. L'agent confronta il codice di ritorno del processo con questi codici di ritorno al termine del processo esterno.

Nota: Poiché le chiamate gestite vengono eseguite come processi esterni, non possono essere annullate una volta avviate.

Chiamate gestite e slot di trasferimento di origine

Un agente contiene un numero di slot di trasferimento di origine, come specificato dalla proprietà dell'agente **maxSourceTransfers**, descritto in Proprietà avanzate dell'agente: Limite di trasferimento.

Ogni volta che viene eseguita una chiamata gestita o un trasferimento gestito, occupano uno slot di trasferimento di origine. Lo slot viene rilasciato una volta completata la chiamata gestita o il trasferimento gestito.

Se tutti gli slot di trasferimento di origine sono in uso quando un agent riceve una nuova chiamata gestita o una richiesta di trasferimento gestito, la richiesta viene accodata dall'agent fino a quando non diventa disponibile uno slot.

Se una chiamata gestita avvia un trasferimento gestito (ad esempio, se una chiamata gestita esegue uno script Ant e tale script Ant utilizza l'attività fte: filecopy o fte: filemove per trasferire un file), sono necessari due slot di trasferimento di origine:

- Uno per il trasferimento gestito
- Uno per la chiamata gestita

In questa situazione, è importante notare che se il trasferimento gestito impiega molto tempo per essere completato o va in ripristino, i due slot di trasferimento di origine vengono occupati

fino a quando il trasferimento gestito non viene completato, annullato o in timeout a causa di **transferRecoveryTimeout**. Consultare Transfer recovery timeout concepts per i dettagli su **transferRecoveryTimeout**. Ciò può potenzialmente limitare il numero di altri trasferimenti gestiti o chiamate gestite che l'agent può elaborare.

Per questo motivo, è necessario considerare la progettazione di una chiamata gestita per garantire che non occupi slot di trasferimento di origine per un lungo periodo di tempo.

Utilizzo di REST API con chiamate gestite

V 9.3.0 V 9.3.0

I comandi HTTP GET e HTTP POST sono supportati per abilitare le chiamate gestite e funzionano solo sulla Versione 3 di REST API.

Altri verbi, ad esempio, HTTP DELETE e HTTP UPDATE non sono supportati e restituiscono il codice di errore HTTP 405 se si tenta di utilizzarli.

Attenzione: Una volta inoltrata, una chiamata gestita non può essere annullata utilizzando REST API.

Utilizzo di Apache Ant con MFT

Managed File Transfer fornisce attività che è possibile utilizzare per integrare la funzione di trasferimento file nello strumento Apache Ant .

È possibile utilizzare il comando **fteAnt** per eseguire attività Ant in un ambiente Managed File Transfer già configurato. È possibile utilizzare le attività di trasferimento file Ant dagli script Ant per coordinare operazioni di trasferimento file complesse da un linguaggio di script interpretato.

Per ulteriori informazioni su Apache Ant, consultare la Apache Ant pagina web del progetto: https://ant.apache.org/

Concetti correlati

“Introduzione all'utilizzo degli script Ant con MFT” a pagina 1227
L'uso di script Ant con Managed File Transfer consente di coordinare operazioni di trasferimento file complesse da un linguaggio di script interpretato.

fteAnt: eseguire attività Ant in MFT

Riferimenti correlati

“Attività Ant di esempio per MFT” a pagina 1228
Esistono diversi script Ant di esempio forniti con l'installazione di Managed File Transfer. Questi esempi si trovano nella directory *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Ogni script di esempio contiene una destinazione `init` , modificare le proprietà impostate nella destinazione `init` per eseguire questi script con la propria configurazione.

Introduzione all'utilizzo degli script Ant con MFT

L'uso di script Ant con Managed File Transfer consente di coordinare operazioni di trasferimento file complesse da un linguaggio di script interpretato.

Script Ant

Gli script Ant (o file di build) sono documenti XML che definiscono una o più destinazioni. Queste destinazioni contengono elementi attività da eseguire. Managed File Transfer fornisce attività che è possibile utilizzare per integrare la funzione di trasferimento file in Apache Ant. Per informazioni sugli script Ant , consultare la pagina Web del progetto Apache Ant : https://ant.apache.org/

Esempi di script Ant che utilizzano attività Managed File Transfer vengono forniti con l'installazione del prodotto nella directory *MQ_INSTALLATION_PATH*/mqft/samples/fteant

Sugli agent bridge di protocollo, gli script Ant vengono eseguiti sul sistema agent bridge di protocollo. Questi script Ant non hanno accesso diretto ai file sul server FTP o SFTP.

Spazio dei nomi

Uno spazio dei nomi viene utilizzato per differenziare le attività Ant di trasferimento file da altre attività Ant che potrebbero condividere lo stesso nome. Definire lo spazio dei nomi nella tag del progetto del proprio script Ant .

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

L'attributo `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` indica a Ant di cercare le definizioni delle attività con prefisso `fte` nella libreria `com.ibm.wmqfte.ant.taskdefs`.

Non è necessario utilizzare `fte` come prefisso dello spazio dei nomi; è possibile utilizzare qualsiasi valore. Il prefisso dello spazio dei nomi `fte` viene utilizzato in tutti gli esempi e gli script Ant di esempio.

Esecuzione di script Ant

Per eseguire gli script Ant che contengono le attività di trasferimento file Ant utilizzare il comando **fteAnt** . Ad esempio:

```
fteAnt -file ant_script_location/ant_script_name
```

Per ulteriori informazioni, consultare **fteAnt**: eseguire le attività Ant in MFT.

Codici di ritorno

Le attività di trasferimento file Ant restituiscono gli stessi codici di ritorno dei comandi Managed File Transfer . Per ulteriori informazioni, consultare Codici di ritorno per MFT.

Riferimenti correlati

fteAnt: eseguire attività Ant in MFT

“Attività Ant di esempio per MFT” a pagina 1228

Esistono diversi script Ant di esempio forniti con l'installazione di Managed File Transfer. Questi esempi si trovano nella directory *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Ogni script di esempio contiene una destinazione `init` , modificare le proprietà impostate nella destinazione `init` per eseguire questi script con la propria configurazione.

Attività Ant di esempio per MFT

Esistono diversi script Ant di esempio forniti con l'installazione di Managed File Transfer. Questi esempi si trovano nella directory *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Ogni script di esempio contiene una destinazione `init` , modificare le proprietà impostate nella destinazione `init` per eseguire questi script con la propria configurazione.

Invia un'email a

L'esempio email dimostra come utilizzare le attività Ant per trasferire un file e inviare un'email a un indirizzo email specificato se il trasferimento non riesce. Lo script controlla che gli agent di origine e di destinazione siano attivi e in grado di elaborare i trasferimenti utilizzando l'attività Managed File Transfer ping . Se entrambi gli agent sono attivi, lo script utilizza l'attività Managed File Transfer fte: filecopy per trasferire un file tra gli agent di origine e di destinazione, senza eliminare il file originale. Se il

trasferimento ha esito negativo, lo script invia un'email contenente informazioni sull'errore utilizzando l'attività Ant email standard.

hub

L'esempio hub è costituito da due script: `hubcopy.xml` e `hubprocess.xml`. Lo script `hubcopy.xml` mostra come utilizzare lo script Ant per creare topologie di stile 'hub and spoke'. In questo esempio, due file vengono trasferiti da agent in esecuzione su macchine spoke a un agent in esecuzione sulla macchina hub. Entrambi i file vengono trasferiti contemporaneamente e, una volta completati i trasferimenti, lo script di `hubprocess.xml` Ant viene eseguito sulla macchina hub per elaborare i file. Se entrambi i file vengono trasferiti correttamente, lo script Ant concatena il contenuto dei file. Se i file non vengono trasferiti correttamente, lo script Ant esegue il cleanup eliminando i dati dei file che sono stati trasferiti. Per far funzionare correttamente questo esempio, è necessario inserire lo script `hubprocess.xml` nel percorso di comandi dell'agent hub. Per ulteriori informazioni sull'impostazione del percorso del comando di un agent, consultare commandPath MFT property.

librarytransfer (solo piattaformaIBM i)

IBM i

IBM i L'esempio librarytransfer illustra come utilizzare le attività Ant per trasferire una libreria IBM i su un IBM i sistema su un secondo sistema IBM i .

IBM i L'esempio librarytransfer utilizza il supporto del file di salvataggio nativo su IBM i con attività Ant predefinite disponibili in Managed File Transfer per trasferire gli oggetti della libreria nativa tra due sistemi IBM i . L'esempio utilizza un < presrc> elemento nidificato in un'attività Managed File Transfer filecopy per richiamare uno script eseguibile `librarysave.sh` che salva la libreria richiesta sul sistema dell'agente di origine in un file di salvataggio temporaneo. Il file di salvataggio viene spostato dall'attività ant filecopy nel sistema dell'agent di destinazione in cui viene utilizzato un elemento nidificato < postdst> per richiamare lo script eseguibile `libraryrestore.sh` per ripristinare la libreria salvata nel file di salvataggio sul sistema di destinazione.

IBM i Prima di eseguire questo esempio, è necessario completare una configurazione come descritto nel file `librarytransfer.xml` . È inoltre necessario avere un Managed File Transfer ambiente di lavoro su due computer IBM i . Il programma di installazione deve essere costituito da un agent di origine in esecuzione sulla prima macchina IBM i e da un agent di destinazione in esecuzione sulla seconda macchina IBM i . I due agenti devono essere in grado di comunicare tra loro.

IBM i L'esempio librarytransfer è costituito dai seguenti tre file:

- `librarytransfer.xml`
- `librarysave.sh` (< script eseguibile presrc>)
- `libraryrestore.sh` (< postdst> script eseguibile)

I file di esempio si trovano nella seguente directory: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`

IBM i Per eseguire questo esempio, l'utente deve completare la seguente procedura:

1. Avviare una sessione Qshell. In una finestra comandi IBM i immettere: STRQSH
2. Passare alla directory `bin` nel modo seguente:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Dopo aver completato la configurazione richiesta, eseguire l'esempio utilizzando il seguente comando:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer (solo piattaformaIBM i)

IBM i L'esempio physicalfiletransfer dimostra come utilizzare le attività Ant per trasferire un file di origine fisico o di database da una libreria su un sistema IBM i a una libreria su un secondo sistema IBM i .

IBM i L'esempio physicalfiletransfer utilizza il supporto del file di salvataggio nativo su IBM i con le attività Ant predefinite disponibili in Managed File Transfer per trasferire i file di origine fisici e database completi tra due sistemi IBM i . L'esempio utilizza un elemento nidificato < presrc> all'interno di un'attività di copia file Managed File Transfer per richiamare uno script eseguibile `physicalfilesave.sh` per salvare il file di database o fisico di origine richiesto da una libreria sul sistema dell'agent di origine in un file di salvataggio temporaneo. Il file di salvataggio viene spostato dall'attività ant di copia file nel sistema dell'agent di destinazione in cui viene utilizzato un elemento nidificato < postdst> per richiamare lo script eseguibile `physicalfilerestore.sh` , quindi ripristina l'oggetto file all'interno del file di salvataggio in una libreria specificata sul sistema di destinazione.

IBM i Prima di eseguire questo esempio, è necessario completare una configurazione come descritto nel file `physicalfiletransfer.xml` . È inoltre necessario disporre di un ambiente Managed File Transfer funzionante su due sistemi IBM i . L'impostazione deve essere composta da un agente di origine in esecuzione sul primo sistema IBM i e da un agente di destinazione in esecuzione sul secondo sistema IBM i . I due agenti devono essere in grado di comunicare tra loro.

IBM i Il campione physicalfiletransfer è composto dai seguenti tre file:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (< script eseguibile presrc>)
- `physicalfilerestore.sh` (< postdst> script eseguibile)

I file di esempio si trovano nella seguente directory: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i Per eseguire questo esempio, l'utente deve completare la seguente procedura:

1. Avviare una sessione Qshell. In una finestra comandi IBM i immettere: STRQSH
2. Passare alla directory `bin` nel modo seguente:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Dopo aver completato la configurazione richiesta, eseguire l'esempio utilizzando il seguente comando:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

tempo massimo

L'esempio timeout illustra come utilizzare le attività Ant per tentare un trasferimento di file e per annullare il trasferimento se impiega più tempo di un valore di timeout specificato. Lo script avvia un trasferimento file utilizzando l'attività Managed File Transfer fte: filecopy . Il risultato di questo trasferimento viene rinviato. Lo script utilizza l'attività Managed File Transfer fte: awaitoutcome Ant per attendere un determinato numero di secondi per il completamento del trasferimento. Se il trasferimento non viene completato nel tempo specificato, l'attività Managed File Transfer fte: cancel Ant viene utilizzata per annullare il trasferimento file.

Trasferimento vsam

z/OS

z/OS L'esempio vsamtransfer illustra come utilizzare le attività Ant per il trasferimento da un dataset VSAM a un'altra serie di dati VSAM utilizzando Managed File Transfer. Managed File Transfer

attualmente non supporta il trasferimento di dataset VSAM. Lo script di esempio scarica i record di dati VSAM in un dataset sequenziale utilizzando gli `presrc` elementi nidificati di richiamo programma per richiamare il file eseguibile `datasetcopy.sh`. Lo script utilizza l'attività Managed File Transfer fte: filemove per trasferire il dataset sequenziale dall'agent di origine all'agent di destinazione. Lo script utilizza quindi gli elementi nidificati del richiamo del programma `postdst` ../refdev/nested_params.dita per richiamare lo script `loadvsam.jcl` . Questo script JCL carica i record del dataset trasferito in un dataset VSAM di destinazione. Questo esempio utilizza JCL per la chiamata di destinazione per dimostrare questa opzione di lingua. Lo stesso risultato può essere ottenuto anche utilizzando un secondo script di shell.

z/OS Questo esempio non richiede che i dataset di origine e di destinazione siano VSAM. L'esempio funziona per tutti i dataset se i dataset di origine e di destinazione sono dello stesso tipo.

z/OS Per il corretto funzionamento di questo esempio, è necessario inserire lo script `datasetcopy.sh` nel percorso comandi dell'agent di origine e lo script `loadvsam.jcl` nel percorso comandi dell'agent di destinazione. Per ulteriori informazioni sull'impostazione del percorso del comando di un agent, consultare commandPath MFT property.

codice postale

L'esempio zip è composto da due script: `zip.xml` e `zipfiles.xml`. L'esempio illustra come utilizzare l'elemento nidificato `presrc` ../refdev/nested_params.dita all'interno dell'attività Managed File Transfer fte: filemove per eseguire uno script Ant prima di eseguire un'operazione di spostamento del trasferimento file. Lo script `zipfiles.xml` richiamato dall'elemento nidificato `presrc` nello script `zip.xml` comprime il contenuto di una directory. Lo script `zip.xml` trasferisce il file compresso. Questo esempio richiede che lo script `zipfiles.xml` Ant sia presente nel percorso comandi dell'agent di origine. Questo perché lo script `zipfiles.xml` Ant contiene la destinazione utilizzata per comprimere il contenuto della directory sull'agent di origine. Per ulteriori informazioni sull'impostazione del percorso del comando di un agent, consultare commandPath MFT property.

Concetti correlati

“Introduzione all'utilizzo degli script Ant con MFT” a pagina 1227
L'uso di script Ant con Managed File Transfer consente di coordinare operazioni di trasferimento file complesse da un linguaggio di script interpretato.

Riferimenti correlati

`fteAnt`: eseguire attività Ant in MFT

Personalizzare MFT con uscite utente

È possibile personalizzare le funzioni di Managed File Transfer utilizzando i propri programmi noti come routine di uscita utente.

Importante: Qualsiasi codice all'interno di un'uscita utente non è supportato da IBMe qualsiasi problema con tale codice deve essere esaminato inizialmente dall'azienda o dal fornitore che ha fornito l'uscita.

Managed File Transfer fornisce punti nel codice in cui Managed File Transfer può passare il controllo a un programma scritto dall'utente (una routine di uscita utente). Questi punti sono noti come punti di uscita utente. Managed File Transfer può quindi riprendere il controllo quando il programma ha terminato il suo lavoro. Non è necessario utilizzare alcuna delle uscite utente, ma sono utili se si desidera estendere e personalizzare la funzione del proprio sistema Managed File Transfer per soddisfare i requisiti specifici.

Esistono due punti durante l'elaborazione del trasferimento file dove è possibile richiamare un'uscita utente sul sistema di origine e due punti durante l'elaborazione del trasferimento file dove è possibile richiamare un'uscita utente sul sistema di destinazione. La seguente tabella riepiloga ciascuno di questi punti di uscita utente e l'interfaccia Java che è necessario implementare per utilizzare i punti di uscita.

Tabella 187. Riepilogo dei punti di uscita lato origine e lato destinazione e delle interfacce Java	
Punto d'uscita	**Interfaccia Java da implementare**
Punti di uscita lato origine:	
Prima che inizi l'intero trasferimento file	SourceTransferStartExit.java interface
Una volta completato l'intero trasferimento file	SourceTransferEndExitInterfaccia .java
Punti di uscita lato destinazione:	
Prima che inizi l'intero trasferimento file	DestinationTransferStartExit.java interface
Una volta completato l'intero trasferimento file	DestinationTransferEndExit.java

Le uscite utente vengono richiamate nell'ordine riportato di seguito:

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

Le modifiche apportate dalle uscite SourceTransferStartExit e DestinationTransferStartExit vengono propagate come input alle uscite successive. Ad esempio, se l'uscitaStartExit di SourceTransfermodifica i metadati di trasferimento, le modifiche si riflettono nei metadati di trasferimento di input per le altre uscite.

Le uscite utente e le chiamate del programma vengono richiamate nel seguente ordine:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Note:

1. Il **`DestinationTransferEndExits`** viene eseguito solo quando il trasferimento viene completato, con esito positivo o parziale.
2. Il **`postDestinationCall`** viene eseguito solo quando il trasferimento viene completato, con esito positivo o parziale.
3. **`SourceTransferEndExits`** viene eseguito per trasferimenti riusciti, parzialmente riusciti o non riusciti.
4. **`postSourceCall`** viene richiamato solo se:
   - Il trasferimento non è stato annullato.
   - C'è un esito positivo o parzialmente positivo.
   - Tutti i programmi di trasferimento post - destinazione sono stati eseguiti correttamente.

Creazione dell'uscita utente

Le interfacce per creare un'uscita utente sono contenute in *`MQ_INSTALL_DIRECTORY`*`/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar`. È necessario includere questo file .jar nel percorso di classe quando si crea l'uscita. Per eseguire l'uscita, estrarre l'uscita come file .jar e collocare questo file .jar in una directory come descritto nella seguente sezione.

Ubicazioni uscita utente

È possibile memorizzare le routine di uscita utente in due possibili ubicazioni:

- La directory `exits` . Esiste una directory di uscite in ciascuna directory dell'agente. Ad esempio: `var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits`
- È possibile impostare la proprietà exitClassPath per specificare un'ubicazione alternativa. Se ci sono classi di uscita sia nella directory `exits` che nel percorso classe impostato da exitClassPath, le classi nella directory `exits` hanno la priorità, il che significa che se ci sono classi in entrambe le ubicazioni con lo stesso nome, le classi nella directory `exits` hanno la priorità.

Configurazione di un agent per utilizzare le uscite utente

È possibile impostare quattro proprietà agent per specificare le uscite utente richiamate da un agent. Queste proprietà dell'agent sono `sourceTransferStartExitClasses`, `sourceTransferEndExitClasses`, `destinationTransferStartExitClassese` `destinationTransferEndExitClasses`. Per informazioni su come utilizzare queste proprietà, consultare MFT Proprietà agent per uscite utente.

Uscite utente in esecuzione su agent bridge di protocollo

Quando l'agente di origine richiama l'uscita, passa all'uscita un elenco degli elementi di origine per il trasferimento. Per gli agent normali, questo è un elenco di nomi file completi. Poiché i file devono essere locali (o accessibili tramite un montaggio), l'uscita è in grado di accedervi e crittografarli.

Tuttavia, per un agent bridge di protocollo, le voci nell'elenco hanno il seguente formato:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

Per ogni voce nell'elenco, l'exit deve connettersi prima al file server (utilizzando l'FTP. FTPS o protocolli SFTP), scaricare il file, crittografarlo localmente e quindi caricare di nuovo il file crittografato sul server di file.

Esecuzione delle uscite utente sugli agent bridge Connect:Direct

Non è possibile eseguire uscite utente su agent bridge Connect:Direct .

Concetti correlati

“Uscite utente di origine e destinazione MFT” a pagina 1233

Metadati per uscite utente MFT

Interfacce Java per uscite utente MFT

Riferimenti correlati

“Abilitazione del debug remoto per le uscite utente MFT” a pagina 1238

Durante lo sviluppo delle uscite utente, è possibile utilizzare un programma di debug per individuare i problemi nel codice.

“User exit di trasferimento di origine MFT di esempio” a pagina 1239

“User exit delle credenziali del bridge di protocollo di esempio” a pagina 1240

Uscite utente del monitoraggio risorse MFT

MFT Proprietà dell'agent per uscite utente

Uscite utente di origine e destinazione MFT

Separatori di directory

I separatori di directory nelle specifiche del file di origine sono sempre rappresentati utilizzando i caratteri barra (/), indipendentemente da come sono stati specificati i separatori di directory nel comando **`fteCreateTransfer`** o in IBM MQ Explorer. È necessario tenerne conto quando si scrive un'uscita. Ad

esempio, se si desidera controllare che il seguente file di origine esista: `c:\a\b.txt` e si è specificato questo file di origine utilizzando il comando **fteCreateTransfer** o IBM MQ Explorer, tenere presente che il nome file è effettivamente memorizzato come: `c:/a/b.txt` Quindi, se si cerca la stringa originale di `c:\a\b.txt`, non si troverà una corrispondenza.

Punti di uscita lato origine

Prima che inizi l'intero trasferimento file

Questa uscita viene richiamata dall'agent di origine quando una richiesta di trasferimento è la successiva nell'elenco di trasferimenti in sospeso e il trasferimento sta per essere avviato.

Gli utilizzi di esempio di questo punto di uscita sono l'invio di file in fasi a una directory a cui l'agent ha accesso in lettura / scrittura utilizzando un comando esterno o la ridenominazione dei file sul sistema di destinazione.

Passare i seguenti argomenti a questa exit:

- Nome agent di origine
- Nome agent di destinazione
- Metadati di ambiente
- Metadati di trasferimento
- Specifiche file (inclusi metadati di file)

I dati restituiti da questa uscita sono i seguenti:

- Metadati trasferimento aggiornati. Le voci possono essere aggiunte, modificate ed eliminate.
- Elenco aggiornato di specifiche file, costituito da coppie di nomi file di origine e di destinazione. Le voci possono essere aggiunte, modificate ed eliminate
- Indicatore che specifica se continuare il trasferimento
- Stringa da inserire nel log di trasferimento.

Implementare SourceTransferStartExit.java interface per richiamare il codice di uscita utente in questo punto.

Una volta completato l'intero trasferimento file

Questa uscita viene richiamata dall'agent di origine una volta completato l'intero trasferimento file.

Un esempio di utilizzo di questo punto di uscita è quello di eseguire alcune attività di completamento, come l'invio di un'email o di un messaggio IBM MQ per indicare che il trasferimento è stato completato.

Passare i seguenti argomenti a questa exit:

- Risultato uscita trasferimento
- Nome agent di origine
- Nome agent di destinazione
- Metadati di ambiente
- Metadati di trasferimento
- Risultati file

I dati restituiti da questa uscita sono i seguenti:

- Stringa aggiornata da inserire nel log di trasferimento.

Implementare SourceTransferEndExit.java interface per richiamare il codice di uscita utente in questo punto di uscita.

Punti di uscita lato destinazione

Prima che inizi l'intero trasferimento file

Un esempio di utilizzo di questo punto di uscita è la convalida delle autorizzazioni nella destinazione.

Passare i seguenti argomenti a questa exit:

- Nome agent di origine
- Nome agent di destinazione
- Metadati di ambiente
- Metadati di trasferimento
- Specifiche di file

I dati restituiti da questa uscita sono i seguenti:

- Serie aggiornata di nomi file di destinazione. Le voci possono essere modificate ma non aggiunte o eliminate.
- Indicatore che specifica se continuare il trasferimento
- Stringa da inserire nel log di trasferimento.

Implementare DestinationTransferStartExit.java interface per chiamare il codice di user exit in questo punto di uscita.

Una volta completato l'intero trasferimento file

Un esempio di utilizzo di questa uscita utente è quello di avviare un processo batch che utilizza i file trasferiti o di inviare una e-mail se il trasferimento non è riuscito.

Passare i seguenti argomenti a questa exit:

- Risultato uscita trasferimento
- Nome agent di origine
- Nome agent di destinazione
- Metadati di ambiente
- Metadati di trasferimento
- Risultati file

I dati restituiti da questa uscita sono i seguenti:

- Stringa aggiornata da inserire nel log di trasferimento.

Implementare DestinationTransferEndExit.java interface per richiamare il codice di uscita utente in questo punto di uscita.

Concetti correlati

Interfacce Java per uscite utente MFT

Riferimenti correlati

“Abilitazione del debug remoto per le uscite utente MFT” a pagina 1238
Durante lo sviluppo delle uscite utente, è possibile utilizzare un programma di debug per individuare i problemi nel codice.

“User exit di trasferimento di origine MFT di esempio” a pagina 1239

Uscite utente del monitoraggio risorse MFT

Utilizzo delle uscite utente I/O di trasferimento MFT

È possibile utilizzare le uscite utente I/O di trasferimento Managed File Transfer per configurare il codice personalizzato per eseguire il lavoro I/O del file system sottostante per i trasferimenti Managed File Transfer .

Generalmente, per i trasferimenti MFT , un agent effettua una selezione da uno dei provider I/O integrati per interagire con i file system appropriati per il trasferimento. I provider I/O integrati supportano i seguenti tipi di file system:

- File system UNIX-type e Windows-type regolari
- z/OS Dataset sequenziali e partizionati z/OS (solo su z/OS)
- IBM i File di salvataggio nativi IBM i (solo su IBM i)
- Code IBM MQ
- Server di protocollo FTP e SFTP remoti (solo per agent bridge di protocollo)
- Nodi Connect:Direct remoti (solo per agent bridge Connect:Direct)

Per i file system non supportati o per i quali è necessario un funzionamento I/O personalizzato, è possibile scrivere un'uscita utente I/O di trasferimento.

Le uscite utente I/O di trasferimento utilizzano l'infrastruttura esistente per le uscite utente. Tuttavia, queste uscite utente I/O di trasferimento differiscono da altre uscite utente perché la loro funzione è accessibile più volte durante il trasferimento per ogni file.

Utilizzare la proprietà dell'agent IOExitClasses (nel file `agent.properties`) per specificare quali classi di uscita I/O caricare. Separare ciascuna classe di uscita con una virgola, ad esempio:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

Le interfacce Java per le uscite utente I/O di trasferimento sono le seguenti:

Uscita IO
Il punto di ingresso principale utilizzato per determinare se viene utilizzata l'uscita I/O. Questa istanza è responsabile della creazione di istanze IOExitPath .

È necessario specificare solo l'interfaccia di uscita I/O IOExit per la proprietà agent IOExitClasses.

IOExitPath
Rappresenta un'interfaccia astratta; ad esempio, un contenitore di dati o un carattere jolly che rappresentano una serie di contenitori di dati. Non è possibile creare un'istanza di classe che implementa questa interfaccia. L'interfaccia consente di esaminare il percorso e di elencare i percorsi derivati. Il percorso IOExitResourcee le interfacce di percorso IOExitWildcardestendono IOExitPath.

IOExitChannel
Abilita la lettura o la scrittura dei dati in una risorsa IOExitPath .

Canale IOExitRecord
Estende l'interfaccia IOExitChannel per le risorse IOExitPath orientate ai record, che consente la lettura o la scrittura dei dati in una risorsa IOExitPath in multipli di record.

IOExitLock
Rappresenta un blocco su una risorsa IOExitPath per l'accesso condiviso o esclusivo.

z/OS **IOExitRecordResourcePath**
Estende l'interfaccia del percorso IOExitResourceper rappresentare un contenitore di dati per un file orientato ai record; ad esempio, un dataset z/OS . È possibile utilizzare l'interfaccia per individuare i dati e creare istanze del canale IOExitRecordper le operazioni di lettura o scrittura.

Percorso IOExitResource
Estende l'interfaccia IOExitPath per rappresentare un contenitore di dati; ad esempio, un file o una directory. È possibile utilizzare l'interfaccia per individuare i dati. Se l'interfaccia rappresenta una directory, è possibile utilizzare il metodo listPaths per restituire un elenco di percorsi.

Percorso IOExitWildcard
Estende l'interfaccia IOExitPath per rappresentare un percorso che denota un carattere jolly. È possibile utilizzare questa interfaccia per corrispondere a più percorsi IOExitResource.

IOExitProperties
Specifica le proprietà che determinano in che modo Managed File Transfer gestisce IOExitPath per determinati aspetti di I/O. Ad esempio, se utilizzare file intermedi o se rileggere una risorsa dall'inizio se un trasferimento viene riavviato.

Concetti correlati

"Personalizzare MFT con uscite utente" a pagina 1231
È possibile personalizzare le funzioni di Managed File Transfer utilizzando i propri programmi noti come routine di uscita utente.

Riferimenti correlati

Interfaccia IOExit.java
Interfaccia IOExitChannel.java
Interfaccia IOExitLock.java
Interfaccia IOExitPath.java
Interfaccia IOExitProperties.java
Interfaccia IOExitRecordChannel.java
z/OS Interfaccia IOExitRecordResourcePath.java
Interfaccia IOExitResourcePath.java
Interfaccia IOExitWildcardPath.java
Il file MFT`agent.properties`

IBM i Uscite utente MFT on IBM i di esempio

Managed File Transfer fornisce uscite utente di esempio specifiche per IBM i con l'installazione. Gli esempi si trovano nelle directory *MQMFT_install_dir*`/samples/ioexit-IBMi` e *MQMFT_install_dir*`/samples/userexit-IBMi`.

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit
L'esempio com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit trasferisce i file nel filesystem QDLS su IBM i. Una volta installata l'uscita, tutti i trasferimenti ai file che iniziano con `/QDLS` utilizzano automaticamente l'uscita.

Per installare questa uscita, completare la seguente procedura:

1. Copiare il file `com.ibm.wmqfte.samples.ibmi.ioexits.jar` dalla directory *WMQFTE_install_dir*`/samples/ioexit-IBMi` alla directory `exits` dell'agent.
2. Aggiungere com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit alla proprietà IOExitClasses .
3. Riavviare l'agent.

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit
L'uscita utente di esempio com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit si comporta come un controllo file MFT e trasferisce automaticamente i membri file fisici da una libreria IBM i .

Per eseguire questa uscita, specificare un valore per il campo di metadati "library.qsys.monitor" (utilizzando il parametro **-md** , ad esempio). Questo parametro utilizza un percorso stile IFS per un membro file e può contenere caratteri jolly di file e membri. Ad esempio, /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.FILE/*.MBR.

Questa uscita di esempio ha anche un campo di metadati facoltativo "naming.scheme.qsys.monitor", che è possibile utilizzare per determinare lo schema di denominazione utilizzato durante il trasferimento. Per impostazione predefinita, questo campo è impostato su "unix", che determina la chiamata del file di destinazione `F00.MBR`. È anche possibile specificare il valore "ibmi" per utilizzare

il FILE IBM i FTP FILE.MEMBER , ad esempio /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR viene trasferito come BAR.BAZ.

Per installare questa uscita, completare la seguente procedura:

1. Copiare il file `com.ibm.wmqfte.samples.ibmi.userexits.jar` dalla directory *WMQFTE_install_dir*`/samples/userexit-IBMi` alla directory `exits` dell'agent.
2. Aggiungere com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit alla proprietà sourceTransferStartExitClasses nel file `agent.properties` .
3. Riavviare l'agent.

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

L'uscita utente di esempio com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit elimina un oggetto file vuoto quando il membro del file di origine viene eliminato come parte del trasferimento. Poiché gli oggetti file IBM i possono potenzialmente contenere molti membri, gli oggetti file vengono trattati come directory da MFT. Pertanto, non è possibile eseguire un'operazione di spostamento su un oggetto file utilizzando MFT; le operazioni di spostamento sono supportate solo a livello di membro. Di conseguenza, quando si esegue un'operazione di spostamento su un membro, il file ora vuoto viene lasciato. Utilizzare questa uscita di esempio se si desidera eliminare questi file vuoti come parte della richiesta di trasferimento.

Se si specifica "true" per i metadati "empty.file.delete" e si trasferisce un FTEFileMember, l'uscita di esempio elimina il file parent se il file è vuoto.

Per installare questa uscita, completare la seguente procedura:

1. Copiare il file com.ibm.wmqfte.samples.ibmi.userexits.jar da *WMQFTE_install_dir*`/samples/userexit-IBMi` alla directory `exits` dell'agente.
2. Aggiungere com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit nella proprietà delle classiStartExit sourceTransfernel file `agent.properties` .
3. Riavviare l'agent.

Riferimenti correlati

“Utilizzo delle uscite utente I/O di trasferimento MFT” a pagina 1235
È possibile utilizzare le uscite utente I/O di trasferimento Managed File Transfer per configurare il codice personalizzato per eseguire il lavoro I/O del file system sottostante per i trasferimenti Managed File Transfer .

MFT Proprietà dell'agent per uscite utente

Abilitazione del debug remoto per le uscite utente MFT

Durante lo sviluppo delle uscite utente, è possibile utilizzare un programma di debug per individuare i problemi nel codice.

Poiché le uscite vengono eseguite nella macchina virtuale Java che esegue l'agent, non è possibile utilizzare il supporto di debug diretto generalmente incluso in un ambiente di sviluppo integrato. Tuttavia, è possibile abilitare il debug remoto della JVM e quindi collegare un programma di debug remoto adatto.

Per abilitare il debug remoto, utilizzare i parametri JVM standard **-Xdebug** e **-Xrunjdwp**.
Queste proprietà vengono trasmesse alla JVM che esegue l'agent dalla variabile di ambiente **BFG_JVM_PROPERTIES** . Ad esempio, su AIX and Linux , i seguenti comandi avviano l'agent e fanno in modo che la JVM sia in ascolto per le connessioni del debugger sulla porta TCP 8765.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

L'agent non viene avviato fino a quando il debugger non si collega. Utilizzare il comando **set** su Windows invece del comando **export** .

È anche possibile utilizzare altri metodi di comunicazione tra il debugger e JVM. Ad esempio, la JVM può aprire la connessione al debugger invece che viceversa oppure è possibile utilizzare la memoria condivisa invece di TCP. Per ulteriori dettagli, consultare la documentazione Java Platform Debugger Architecture .

È necessario utilizzare il parametro **-F** (foreground) quando si avvia l'agente in modalità di debug remota.

Utilizzo del programma di debug Eclipse

La seguente procedura si applica alla funzionalità di debug remoto nell'ambiente di sviluppo Eclipse . È anche possibile utilizzare altri programmi di debug remoti compatibili con JPDA.

1. Fare clic su **Esegui** > **Apri finestra di dialogo di debug** (o **Esegui** > **Configurazioni di debug** o **Esegui** > **Finestra di dialogo di debug** a seconda della versione di Eclipse).
2. Fare doppio clic su **Applicazione Java remota** nell'elenco dei tipi di configurazione per creare una configurazione di debug.
3. Completare i campi di configurazione e salvare la configurazione di debug. Se la JVM dell'agent è già stata avviata in modalità debug, è ora possibile connettersi alla JVM.

User exit di trasferimento di origine MFT di esempio

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);
```

```
        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

User exit delle credenziali del bridge di protocollo di esempio

Per informazioni su come utilizzare questa user exit di esempio, consultare Associazione delle credenziali per un server di file utilizzando le classi exit.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();
```

```
    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
                 inputStream = new FileInputStream(propertiesFile);

                 // Load the properties from the file
                 mappingProperties.load(inputStream);
             }
             catch (FileNotFoundException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                 initialisationResult = false;
             }
             catch (IOException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                 initialisationResult = false;
             }
             finally {
                 // Close the inputStream
                 if (inputStream != null) {
                     try {
                         inputStream.close();
                     }
                     catch (IOException ex) {
                         System.err.println("Error initializing SampleCredentialExit.");
                          System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                         initialisationResult = false;
                     }
                 }
             }

             if (initialisationResult) {
                 // Populate the map of mqUserId to server credentials from the properties
                 final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                  while ( propertyNames.hasMoreElements()) {
                     final Object name = propertyNames.nextElement();
                     if (name instanceof String ) {
                         final String mqUserId = ((String)name).trim();
                          // Get the value and split into serverUserId and serverPassword
                        final String value = mappingProperties.getProperty(mqUserId);
                        final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                         String serverUserId = "";
                         String serverPassword = "";
                         if (valueTokenizer.hasMoreTokens()) {
                             serverUserId = valueTokenizer.nextToken().trim();
                         }
                         if (valueTokenizer.hasMoreTokens()) {
                             serverPassword = valueTokenizer.nextToken().trim();
                         }
```

```
                        // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                        // Insert the credentials into the map
                        credentialsMap.put(mqUserId, credentials);
                    }
                }

            }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

User exit delle proprietà del bridge di protocollo di esempio

Per informazioni su come utilizzare questa uscita utente di esempio, consultare ProtocolBridgePropertiesExit2: Ricerca delle proprietà del server di file di protocollo .

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
```

```
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
```

```
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
```

```
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

Controllo di MFT inserendo i messaggi nella coda comandi dell'agente

È possibile scrivere un'applicazione che controlli Managed File Transfer inserendo i messaggi nelle code comandi dell'agente.

È possibile inserire un messaggio nella coda comandi di un agente per richiedere all'agente di eseguire una delle azioni riportate di seguito:

- Crea un trasferimento file
- Crea un trasferimento file pianificato
- Annulla un trasferimento file
- Annulla un trasferimento file pianificato
- Richiama un comando
- Crea un monitoraggio
- Elimina un monitor
- Restituisce un ping per indicare che l'agente è attivo

Per richiedere che l'agent esegua una di queste azioni, il messaggio deve essere in formato XML conforme a uno dei seguenti schemi:

FileTransfer.xsd
I messaggi in questo formato possono essere utilizzati per creare un trasferimento file o un trasferimento file pianificato, per richiamare un comando o per annullare un trasferimento file o un trasferimento file pianificato. Per ulteriori informazioni, vedere Formato del messaggio di richiesta di trasferimento file.

Monitor.xsd
I messaggi in questo formato possono essere utilizzati per creare o eliminare un controllo risorse. Per ulteriori informazioni, consultare MFT monitor request message formats.

PingAgent.xsd
I messaggi in questo formato possono essere utilizzati per eseguire il ping di un agent per verificare che sia attivo. Per ulteriori informazioni, consultare Ping del formato del messaggio di richiesta dell'agent MFT.

L'agente restituisce una risposta ai messaggi di richiesta. Il messaggio di risposta viene inserito in una coda di risposta definita nel messaggio di richiesta. Il messaggio di risposta è in formato XML definito dal seguente schema:

`Reply.xsd`
Per ulteriori informazioni, consultare Formato del messaggio di risposta dell'agent MFT.

Sviluppo di applicazioni per MQ Telemetry

Le applicazioni di telemetria integrano i dispositivi di rilevamento e controllo con altre fonti di informazione disponibili su Internet e nelle imprese.

Sviluppare applicazioni per MQ Telemetry utilizzando modelli di progetto, esempi di lavoro, programmi di esempio, concetti di programmazione e informazioni di riferimento.

Concetti correlati

MQ Telemetry

Casi di utilizzo della telemetria

Attività correlate

InstallazioneMQ Telemetry

AmministrazioneMQ Telemetry

Risoluzione dei problemi di MQ Telemetry

Riferimenti correlati

Riferimento di MQ Telemetry

IBM MQ Telemetry Transport Programmi di esempio

Vengono forniti script di esempio che funzionano con un'applicazione client IBM MQ Telemetry Transport v3 di esempio (`mqttv3app.jar`). Per IBM MQ 8.0.0 e versioni successive, l'applicazione client di esempio non è più inclusa in MQ Telemetry. Faceva parte del IBM Messaging Telemetry Clients SupportPac(non più disponibile). Applicazioni di esempio simili continuano ad essere liberamente disponibili da Eclipse Paho e MQTT.org.

Per le ultime informazioni e i download, consultare le seguenti risorse:

- Il progetto Eclipse Paho, e MQTT.org, hanno dei download gratuiti dei client di telemetria più recenti e degli esempi per una gamma di linguaggi di programmazione. Utilizzare questi siti come ausilio nello sviluppo di programmi di esempio per la pubblicazione e la sottoscrizione di IBM MQ Telemetry Transport e per l'aggiunta di funzioni di sicurezza.
- IBM Messaging Telemetry Clients SupportPac non è più disponibile per il download. Se si dispone di una copia scaricata in precedenza, presenta il seguente contenuto:
  - La versione MA9B di IBM Messaging Telemetry Clients SupportPac includeva un'applicazione di esempio compilata (`mqttv3app.jar`) e una libreria client associata (`mqttv3.jar`). Venivano fornite nelle seguenti directory:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - Nella versione MA9C di questo SupportPac, la directory `/SDK/` e il contenuto sono stati rimossi:
    - Veniva fornita solo l'origine per l'applicazione di esempio (`mqttv3app.jar`). Si trovava in questa directory:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - La libreria client compilata continuava a essere fornita. Si trovava in questa directory:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Se si dispone ancora di una copia di IBM Messaging Telemetry Clients SupportPac(non più disponibile), le informazioni relative all'installazione e all'esecuzione dell'applicazione di esempio vengono fornite in Verifica dell'installazione di MQ Telemetry utilizzando la riga comandi.

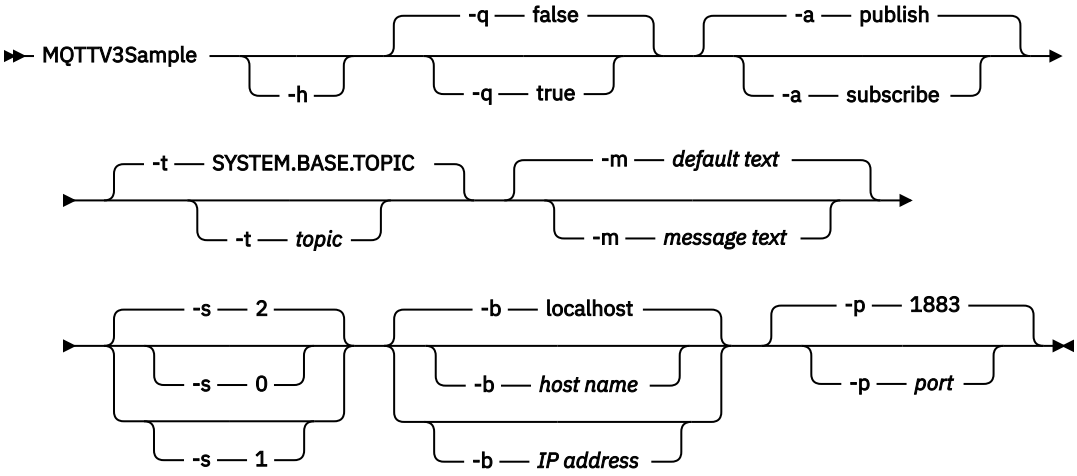
Programma MQTTV3Sample

Informazioni di riferimento sulla sintassi e sui parametri di esempio per il programma `MQTTV3Sample` .

Finalità

Il programma `MQTTV3Sample` può essere utilizzato per pubblicare un messaggio e sottoscrivere un argomento. Per informazioni su come ottenere questo programma di esempio, consultare “IBM MQ Telemetry Transport Programmi di esempio” a pagina 1246.

MQTTV3Sample syntax



Parametri

-h
Stampare questo testo di aiuto e uscire

-q
Impostare la modalità silenziosa, invece di utilizzare la modalità predefinita false.

-a
Impostare la pubblicazione o la sottoscrizione, invece di assumere l'azione predefinita di pubblicazione.

-t
Pubblica o sottoscrivi l'argomento, invece di pubblicare o sottoscrivere l'argomento predefinito

-m
Pubblicare il testo del messaggio invece di inviare il testo di pubblicazione predefinito, "Ciao da un'applicazione MQTT v3 ".

-s
Impostare QoS invece di utilizzare il valore predefinito QoS, 2.

-b
Connettersi a questo nome host o indirizzo IP invece di connettersi al nome host predefinito, localhost.

-p
Utilizzare questa porta invece di utilizzare il valore predefinito, 1883.

Eseguire il programma MQTTV3Sample

Per sottoscrivere un argomento su Windows, utilizzare il comando:

```
run MQTTV3Sample -a subscribe
```

Per pubblicare un messaggio su Windows, utilizzare il comando:

```
run MQTTV3Sample
```

Concetti di programmazione client MQTT

I concetti descritti in questa sezione consentono di comprendere le librerie client per MQTT protocol. I concetti completano la documentazione API che accompagna le librerie client.

Per le ultime informazioni e i download, consultare le seguenti risorse:

- Il progetto Eclipse Paho, e MQTT.org, hanno dei download gratuiti dei client di telemetria più recenti e degli esempi per una gamma di linguaggi di programmazione. Utilizzare questi siti come ausilio nello sviluppo di programmi di esempio per la pubblicazione e la sottoscrizione di IBM MQ Telemetry Transport e per l'aggiunta di funzioni di sicurezza.
- IBM Messaging Telemetry Clients SupportPac non è più disponibile per il download. Se si dispone di una copia scaricata in precedenza, presenta il seguente contenuto:
  - La versione MA9B di IBM Messaging Telemetry Clients SupportPac includeva un'applicazione di esempio compilata (`mqttv3app.jar`) e una libreria client associata (`mqttv3.jar`). Venivano fornite nelle seguenti directory:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - Nella versione MA9C di questo SupportPac, la directory `/SDK/` e il contenuto sono stati rimossi:
    - Veniva fornita solo l'origine per l'applicazione di esempio (`mqttv3app.jar`). Si trovava in questa directory:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - La libreria client compilata continuava a essere fornita. Si trovava in questa directory:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Per sviluppare ed eseguire un client MQTT è necessario copiare o installare queste risorse sul dispositivo client. Non è necessario installare un runtime client separato.

Le condizioni di licenza per client sono associate al server a cui si stanno collegando i client.

Le librerie client MQTT sono implementazioni di riferimento di MQTT protocol. È possibile implementare i propri client in diverse lingue adatte a diverse piattaforme di dispositivi. Vedere IBM MQ Telemetry Transport formato e protocollo.

La documentazione API non fa supposizioni su quale server MQTT è connesso al client. Il comportamento del client potrebbe differire leggermente quando si è connessi a server differenti. Le descrizioni che seguono descrivono il comportamento del client quando è connesso al servizio di telemetria IBM MQ .

Callback e sincronizzazione nelle applicazioni client MQTT

Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Callback

Nota: Consultare il sito Web Eclipse Paho per le ultime modifiche a `MqttCallback`. Ad esempio, `MqttCallback` è definito come un'Interfaccia nella versione Paho del client e i metodi asincroni vengono forniti dalla classe Paho `MqttAsyncClient` .

L'interfaccia di `MqttCallback` ha tre metodi di callback:

`connectionLost(java.lang.Throwable cause)`

`connectionLost` viene chiamato quando un errore di comunicazione porta al rilascio della connessione. Viene anche richiamato se il server elimina la connessione come risultato di un errore sul server dopo che la connessione è stata stabilita. Gli errori del server vengono registrati nel log degli errori del gestore code. Il server elimina la connessione al client e il client richiama `MqttCallback.connectionLost`.

Gli unici errori remoti generati come eccezioni sullo stesso thread dell'applicazione client sono eccezioni da `MqttClient.connect`. Gli errori rilevati dal server una volta stabilita la connessione vengono riportati al metodo di callback `MqttCallback.connectionLost` come `throwables`.

Gli errori tipici del server che risultano in `connectionLost` sono errori di autorizzazione. Ad esempio, il server di telemetria tenta di pubblicare su un argomento per conto di un client che non è autorizzato a pubblicare sull'argomento. Tutto ciò che risulta in un codice condizione `MQCC_FAIL` restituito al server di telemetria può causare l'eliminazione della connessione.

`deliveryComplete(IMqttDeliveryToken token)`

`deliveryComplete` viene richiamato dal client MQTT per passare un token di consegna all'applicazione client; consultare “Token di consegna” a pagina 1255. Utilizzando il token di consegna, il callback può accedere al messaggio pubblicato con il metodo `token.getMessage`.

Quando il callback dell'applicazione restituisce il controllo al client MQTT dopo essere stato richiamato dal metodo `deliveryComplete` , la consegna viene completata. Fino al completamento della consegna, i messaggi con `QoS` 1 o 2 vengono conservati dalla classe di persistenza.

La chiamata a `deliveryComplete` è un punto di sincronizzazione tra l'applicazione e la classe di persistenza. Il metodo `deliveryComplete` non viene mai richiamato due volte per lo stesso messaggio.

Quando il callback dell'applicazione viene restituito da `deliveryComplete` al client MQTT , il client richiama `MqttClientPersistence.remove` per i messaggi con `QoS` 1 o 2. `MqttClientPersistence.remove` elimina la copia memorizzata localmente del messaggio pubblicato.

Da una prospettiva di elaborazione della transazione, la chiamata a `deliveryComplete` è una transazione a fase singola che esegue il commit della consegna. Se l'elaborazione non riesce durante il callback, al riavvio del client `MqttClientPersistence.remove` viene richiamato di nuovo per eliminare la copia locale del messaggio pubblicato. Il callback non viene richiamato di nuovo. Se si utilizza il callback per memorizzare un log dei messaggi consegnati, non è possibile sincronizzare il log con il client MQTT . Se si desidera memorizzare un log in modo affidabile, aggiornare il log nella classe `MqttClientPersistence` .

Il token di consegna e il messaggio sono indicati dal thread dell'applicazione principale e dal client MQTT . Il client MQTT annulla il riferimento all'oggetto `MqttMessage` una volta completata la consegna e l'oggetto token di consegna quando il client si disconnette. L'oggetto `MqttMessage` può essere raccolto nel garbage collector dopo il completamento della consegna se l'applicazione client lo annulla. Il token di consegna può essere un raccoglitore dati inutilizzati dopo che la sessione è stata disconnessa.

È possibile ottenere gli attributi `IMqttDeliveryToken` e `MqttMessage` dopo la pubblicazione di un messaggio. Se si tenta di impostare gli attributi `MqttMessage` dopo che il messaggio è stato pubblicato, il risultato non è definito.

Il client MQTT continua a elaborare le conferme di consegna se il client si riconnette alla precedente sessione con lo stesso `ClientIdentifier` ; consultare “Ripulisci sessioni” a pagina 1252. L'applicazione client MQTT deve impostare `MqttClient.CleanSession` su `false` per la sessione precedente e impostarlo su `false` nella nuova sessione. Il client MQTT crea nuovi token

di consegna e oggetti messaggio nella nuova sessione per le consegne in sospeso. Recupera gli oggetti utilizzando la classe `MqttClientPersistence` . Se il client dell'applicazione ha ancora riferimenti ai vecchi token di consegna e messaggi, annullarli. La richiamata dell'applicazione viene richiamata nella nuova sessione per tutte le distribuzioni avviate nella sessione precedente e completate in questa sessione.

Il callback dell'applicazione viene richiamato dopo la connessione del client dell'applicazione, quando viene completata una consegna in sospeso. Prima che il client dell'applicazione si colleghi, è possibile richiamare le consegne in sospeso utilizzando il metodo `MqttClient.getPendingDeliveryTokens` .

Si noti che l'applicazione client ha originariamente creato l'oggetto del messaggio pubblicato e il relativo array di byte del payload. Il client MQTT fa riferimento a questi oggetti. L'oggetto messaggio restituito dal token di consegna nel metodo `token.getMessage` non è necessariamente lo stesso oggetto messaggio creato dal client. Se una nuova istanza client MQTT ricrea il token di consegna, la classe `MqttClientPersistence` ricrea l'oggetto `MqttMessage` . Per coerenza `token.getMessage` restituisce `null` se `token.isCompleted` è `true`, indipendentemente dal fatto che l'oggetto del messaggio sia stato creato dal client dell'applicazione o dalla classe `MqttClientPersistence` .

`messageArrived(String topic, MqttMessage message)`

`messageArrived` viene richiamato quando arriva una pubblicazione per il client che corrisponde a un argomento di sottoscrizione. `topic` è l'argomento di pubblicazione, non il filtro di sottoscrizione. I due possono essere diversi se il filtro contiene caratteri jolly.

Se l'argomento corrisponde a più sottoscrizioni create dal client, il client riceve più copie della pubblicazione. Se un client pubblica su un argomento a cui è anche sottoscrittore, riceve una copia della propria pubblicazione.

Se un messaggio viene inviato con un QoS di 1 o 2, il messaggio viene memorizzato dalla classe `MqttClientPersistence` prima che il client MQTT chiami `messageArrived`. `messageArrived` funziona come `deliveryComplete`: viene richiamato una sola volta per una pubblicazione e la copia locale della pubblicazione viene rimossa da `MqttClientPersistence.remove` quando `messageArrived` ritorna al client MQTT . Il client MQTT elimina i riferimenti all'argomento e al messaggio quando `messageArrived` ritorna al client MQTT . L'argomento e gli oggetti del messaggio vengono raccolti nel raccoglitore dati inutilizzati, se il client delle applicazioni non ha mantenuto un riferimento agli oggetti.

Callback, thread e sincronizzazione dell'applicazione client

Il client MQTT richiama un metodo di callback su un thread separato al thread dell'applicazione principale. L'applicazione client non crea un thread per il callback, ma viene creata dal client MQTT .

Il client MQTT sincronizza i metodi callback. Viene eseguita una sola istanza del metodo callback alla volta. La sincronizzazione facilita l'aggiornamento di un oggetto che indica quali pubblicazioni sono state consegnate. Un'istanza di `MqttCallback.deliveryComplete` viene eseguita alla volta, quindi è sicuro aggiornare il conteggio senza ulteriore sincronizzazione. È anche il caso di una sola pubblicazione alla volta. Il codice nel metodo `messageArrived` può aggiornare un oggetto senza sincronizzarlo. Se si fa riferimento al conteggio o all'oggetto che si sta aggiornando, in un altro thread, sincronizzare il conteggio o l'oggetto.

Il token di consegna fornisce un meccanismo di sincronizzazione tra il thread dell'applicazione principale e la distribuzione di una pubblicazione. Il metodo `token.waitForCompletion` attende il completamento della consegna di una pubblicazione specifica o la scadenza di un timeout facoltativo. È possibile utilizzare `token.waitForCompletion` nel modo seguente per elaborare una pubblicazione alla volta.

Per sincronizzarsi con il metodo `MqttCallback.deliveryComplete` . Solo quando `MqttCallback.deliveryComplete` ritorna al client MQTT `token.waitForCompletion` viene ripreso. Utilizzando questo meccanismo, è possibile sincronizzare il codice in esecuzione in `MqttCallback.deliveryComplete` prima che il codice venga eseguito nel thread dell'applicazione principale.

Cosa fare se si desidera pubblicare senza attendere la consegna di ogni pubblicazione, ma si desidera una conferma quando tutte le pubblicazioni sono state consegnate? Se si esegue la pubblicazione su un singolo thread, l'ultima pubblicazione da inviare è anche l'ultima da consegnare.

Sincronizzazione delle richieste inviate al server

Tabella 188 a pagina 1251 descrive i metodi nel client MQTT Java che inviano una richiesta al server. A meno che il client delle applicazioni non imposti un timeout indefinito, il client non attende mai indefinitamente il server. Se il client si blocca, si tratta di un problema di programmazione dell'applicazione o di un difetto nel client MQTT .

Tabella 188. Comportamento di sincronizzazione dei metodi che risultano in richieste al server

Metodo	Sincronizzazione	Intervallo di timeout
`MqttClient.Connect`	Attende che venga stabilita una connessione con il server.	Il valore predefinito è 30 secondi oppure, come impostato da un parametro, genera un'eccezione.
`MqttClient.Disconnect`	Attende che il client MQTT termini il lavoro che deve eseguire e che la sessione TCP/IP si scolleghi.	
`MqttClient.Subscribe`	Attende il completamento del metodo Sottoscrivi o UnSubscribe .	
`MqttClient.UnSubscribe`		
`MqttClient.Publish`	Ritorna immediatamente al thread dell'applicazione dopo aver passato la richiesta al client MQTT .	Nessuna.
`IMqttDeliveryToken.waitForCompletion`	Attende la restituzione del token di consegna.	Indefinito o impostato come parametro.

Concetti correlati

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT

I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Ripulisci sessioni

Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Quando si collega un'applicazione client MQTT utilizzando il metodo `MqttClient.connect` , il client identifica la connessione utilizzando l'identificativo client e l'indirizzo del server. Il server verifica se le informazioni sulla sessione sono state salvate da una precedente connessione al server. Se una sessione precedente esiste ancora e `cleanSession=true`, le informazioni sulla sessione precedente sul client e sul server vengono cancellate. Se `cleanSession=false` la sessione precedente viene ripresa. Se non esiste alcuna sessione precedente, viene avviata una nuova sessione.

Nota: L'amministratore IBM MQ può forzare la chiusura di una sessione aperta ed eliminarne tutte le informazioni. Se il client riapre la sessione con `cleanSession=false`, viene avviata una nuova sessione.

Pubblicazioni

Se si utilizza il valore predefinito `MqttConnectOptions`o si imposta `MqttConnectOptions.cleanSession` su `true` prima di collegare il client, tutte le distribuzioni di pubblicazione in sospeso per il client vengono rimosse quando il client si connette.

L'impostazione di ripulitura della sessione non ha alcun effetto sulle pubblicazioni inviate con `QoS=0`. Per `QoS=1` e `QoS=2`, l'utilizzo di `cleanSession=true` potrebbe comportare la perdita di una pubblicazione.

Sottoscrizioni

Se si utilizza il valore predefinito `MqttConnectOptions`o si imposta `MqttConnectOptions.cleanSession` su `true` prima di connettere il client, tutte le vecchie sottoscrizioni per il client vengono rimosse quando il client si connette. Qualsiasi nuova sottoscrizione effettuata dal client durante la sessione viene rimossa quando si disconnette.

Se si imposta `MqttConnectOptions.cleanSession` su `false` prima della connessione, tutte le sottoscrizioni create dal client vengono aggiunte a tutte le sottoscrizioni esistenti per il client prima della connessione. Tutte le sottoscrizioni restano attive alla disconnessione del client.

Un altro modo per comprendere il modo in cui l'attributo `cleanSession` influenza le sottoscrizioni è considerarlo come un attributo modale. Nella sua modalità predefinita, `cleanSession=true`, il client crea sottoscrizioni e riceve pubblicazioni solo nell'ambito della sessione. Nella modalità alternativa, `cleanSession=false`, le sottoscrizioni sono durevoli. Il client può connettersi e disconnettersi e le sue sottoscrizioni rimangono attive. Quando il client si riconnette, riceve tutte le pubblicazioni non consegnate. Mentre è connesso, può modificare l'insieme di sottoscrizioni attive per suo conto.

È necessario impostare la modalità `cleanSession`on prima della connessione; la modalità dura per l'intera sessione. Per modificarne l'impostazione, è necessario disconnettere e riconnettere il client. Se si modificano le modalità dall'utilizzo di `cleanSession=false` a `cleanSession=true`, tutte le sottoscrizioni precedenti per il client e tutte le pubblicazioni non ricevute vengono eliminate.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT

Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Identificativo client

L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

L'identificativo client viene utilizzato nella gestione di un sistema MQTT . Con potenzialmente centinaia di migliaia di clienti da amministrare, è necessario essere in grado di identificare rapidamente un particolare cliente. Ad esempio, si supponga che un dispositivo non abbia funzionato correttamente e che l'utente venga avvisato, forse da un cliente che squilla un help desk. Il cliente deve essere in grado di identificare la periferica ed è necessario essere in grado di correlare tale identificazione con il server generalmente connesso al client.

Quando si sfogliano le connessioni client MQTT , ogni connessione viene etichettata con l'identificativo client. Per decidere come associare al meglio questo identificativo al dispositivo e al server, porsi le seguenti domande:

- Sarebbe conveniente gestire e utilizzare un database che associ ogni dispositivo a un identificativo client e a un server?
- Il nome della periferica può identificare il server a cui è collegata?
- È necessaria una tabella di ricerca che associ un identificativo client ad una periferica fisica?
- L'identificativo del client identifica una particolare periferica, un utente o un'applicazione in esecuzione sul client?
- Se un cliente sostituisce un dispositivo difettoso con uno nuovo, il nuovo dispositivo ha lo stesso identificativo del vecchio dispositivo o si assegna un nuovo identificativo? (Se si modifica un dispositivo fisico e si mantiene lo stesso identificativo, le pubblicazioni in sospeso e le sottoscrizioni attive vengono automaticamente trasferite al nuovo dispositivo.)

È inoltre necessario un sistema per garantire che gli identificativi client siano univoci ed è necessario disporre di un processo affidabile per l'impostazione dell'identificativo sul client. Se il dispositivo client è una "black-box", senza interfaccia utente, è possibile produrre il dispositivo con un identificativo client, oppure è possibile avere un processo di installazione e configurazione del software che configura il dispositivo prima che venga attivato.

Per mantenere l'identificativo breve e univoco, è possibile creare un identificativo client dall'indirizzo MAC della periferica a 48 bit. Se la dimensione di trasmissione non è un problema critico, è possibile utilizzare i restanti 17 byte per semplificare l'amministrazione dell'indirizzo.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso

il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Token di consegna

Quando un client pubblica un argomento, viene creato un nuovo token di consegna. Utilizzare il token di consegna per monitorare la consegna di una pubblicazione o per bloccare l'applicazione client fino al completamento della consegna.

Il token è un oggetto `MqttDeliveryToken` . Viene creato richiamando il metodo MqttTopic.publish () e viene conservato dal client MQTT fino a quando la sessione client non viene disconnessa e la consegna non viene completata.

Il normale utilizzo del token è quello di verificare se la consegna è completa. Blocca l'applicazione client fino al completamento della consegna utilizzando il token restituito per richiamare

`token.waitForCompletion`. In alternativa, fornire un handler `MqttCallBack` . Quando il client MQTT ha ricevuto tutti i riconoscimenti previsti come parte della consegna della pubblicazione, richiama `MqttCallBack.deliveryComplete` passando il token di consegna come parametro.

Fino a quando la consegna non è completa, puoi ispezionare la pubblicazione utilizzando il token di consegna restituito chiamando `token.getMessage`.

Consegne completate

Il completamento delle consegne è asincrono e dipende dalla qualità del servizio associato alla pubblicazione.

Al massimo una volta

`QoS=0`
La consegna è completa immediatamente al ritorno da `MqttTopic.publish`. `MqttCallback.deliveryComplete` viene richiamato immediatamente.

Almeno una volta

`QoS=1`
Il recapito è completo quando è stato ricevuto un riconoscimento alla pubblicazione dal gestore code. `MqttCallback.deliveryComplete` viene richiamato quando viene ricevuto il riconoscimento. Il messaggio potrebbe essere consegnato più di una volta prima che venga richiamato `MqttCallback.deliveryComplete` , se le comunicazioni sono lente o inaffidabili.

Esattamente una volta

`QoS=2`
La consegna è completa quando il client riceve un messaggio di completamento che indica che la pubblicazione è stata pubblicata per i sottoscrittori. `MqttCallback.deliveryComplete` viene richiamato non appena viene ricevuto il messaggio di pubblicazione. Non attende il messaggio di completamento.

In rari casi, l'applicazione client potrebbe non tornare al client MQTT da `MqttCallback.deliveryComplete` normalmente. Sai che la consegna è stata completata, perché `MqttCallback.deliveryComplete` è stato chiamato. Se il client riavvia la stessa sessione, `MqttCallback.deliveryComplete` non viene richiamato di nuovo.

Consegne incomplete

Se la distribuzione non è completa dopo la disconnessione della sessione client, è possibile connettere nuovamente il client e completare la distribuzione. È possibile completare la consegna di un messaggio solo se il messaggio è stato pubblicato in una sessione con l'attributo `MqttConnectionOptions` impostato su `false`.

Creare il client utilizzando lo stesso identificativo client e lo stesso indirizzo server, quindi connettersi, impostando nuovamente l'attributo `cleanSession` `MqttConnectionOptions` su `false` . Se si imposta `cleanSession` su `true`, i token di consegna in sospeso vengono eliminati.

È possibile verificare se sono presenti recapiti in sospeso chiamando `MqttClient.getPendingDeliveryTokens`. È possibile chiamare `MqttClient.getPendingDeliveryTokens` prima di collegare il client.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta"

la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Ultimo testamento e pubblicazione testamentaria

Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Creare un argomento per l'ultimo testamento. È possibile creare un argomento come `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Impostare un "ultimo testamento" utilizzando il metodo `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Considerare la creazione di una data / ora nel messaggio `lastWillPayload` . Includere altre informazioni sul client che consentono di identificare il client e le circostanze della connessione. Passare l'oggetto `MqttConnectionOptions` al costruttore `MqttClient` .

Impostare `lastWillQos` su 1 o 2, per rendere il messaggio persistente in IBM MQe per garantire la consegna. Per conservare le informazioni sull'ultima connessione persa, impostare `lastWillRetained` su `true`.

La pubblicazione "Last Will and Testament" viene inviata ai sottoscrittori se la connessione termina in modo imprevisto. Viene inviato se la connessione termina senza che il client chiami il metodo `MqttClient.disconnect` .

Per monitorare le connessioni, completare la pubblicazione "Last Will and Testament" con altre pubblicazioni per registrare le connessioni e le disconnessioni programmate.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni

Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Persistenza del messaggio nei client MQTT

I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

In MQTT, la persistenza del messaggio ha due aspetti: il modo in cui il messaggio viene trasferito e se viene accodato in IBM MQ come messaggio persistente.

1. Il client MQTT accoppia la persistenza del messaggio con la QoS (quality of service). A seconda della qualità del servizio scelta per un messaggio, il messaggio viene reso persistente. La persistenza del messaggio è necessaria per implementare la QoS (quality of service) richiesta.

   Se si specifica "al massimo una volta", `QoS=0`, il client elimina il messaggio non appena viene pubblicato. Se si verifica un errore nell'elaborazione upstream del messaggio, il messaggio non viene inviato di nuovo. Anche se il client rimane attivo, il messaggio non viene inviato di nuovo. Il funzionamento dei messaggi `QoS=0` è lo stesso dei messaggi non persistenti veloci IBM MQ .

   Se un messaggio viene pubblicato da un client con QoS di 1 o 2, viene reso persistente. Il messaggio viene memorizzato localmente e scartato dal client solo quando non è più necessario per garantire "almeno una volta", `QoS=1`o "esattamente una volta", `QoS=2`, la consegna.
2. Se un messaggio è contrassegnato come QoS 1 o 2, viene accodato in IBM MQ come un messaggio persistente. Se è contrassegnato come `QoS=0`, viene accodato in IBM MQ come messaggio non persistente. In IBM MQ i messaggi non persistenti vengono trasferiti tra gestori code "esattamente una volta", a meno che il canale dei messaggi non abbia l'attributo `NPMSPEED` impostato su `FAST`.

Una pubblicazione persistente viene memorizzata sul cliente fino a quando non viene ricevuta da un'applicazione client. Per `QoS=2`, la pubblicazione viene eliminata dal client quando il callback dell'applicazione restituisce il controllo. Per `QoS=1` l'applicazione potrebbe ricevere nuovamente la pubblicazione, se si verifica un errore. Per `QoS=0`, il callback riceve la pubblicazione non più di una volta. Potrebbe non ricevere la pubblicazione se si verifica un errore o se il client è disconnesso al momento della pubblicazione.

Quando si sottoscrive un topic, è possibile ridurre il QoS con cui il sottoscrittore riceve i messaggi in modo che corrisponda alle sue capacità di persistenza. Le pubblicazioni create in un QoS superiore vengono inviate con il QoS più elevato richiesto dal sottoscrittore (subscriber).

Memorizzare i messaggi

L'implementazione dell'archiviazione dei dati su piccoli dispositivi varia molto. Il modello di salvataggio temporaneo dei messaggi persistenti nella memoria gestita dal client MQTT potrebbe essere troppo lento o richiedere troppa memoria. Nei dispositivi mobili, il sistema operativo mobile potrebbe fornire un servizio di archiviazione ideale per i messaggi MQTT .

Per fornire flessibilità nel soddisfare i vincoli di piccole unità, il client MQTT ha due interfacce di persistenza. Le interfacce definiscono le operazioni coinvolte nella memorizzazione dei messaggi persistenti. Le interfacce sono descritte nella documentazione API per MQTT client for Java. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT. È possibile implementare le interfacce per adattarsi a una periferica. Il

client MQTT eseguito su Java SE ha un'implementazione predefinita delle interfacce che memorizzano i messaggi persistenti nel file system. Utilizza il pacchetto `java.io` .

Classi di persistenza

`MqttClientPersistence`
Passare un'istanza dell'implementazione di `MqttClientPersistence` al client MQTT come parametro del costruttore `MqttClient` . Se si omette il parametro `MqttClientPersistence` dal costruttore `MqttClient` , il client MQTT memorizza i messaggi persistenti utilizzando la classe `MqttDefaultFilePersistence`.

`MqttPersistable`
`MqttClientPersistence` ottiene e inserisce `MqttPersistable` oggetti utilizzando una chiave di memoria. È necessario fornire un'implementazione di `MqttPersistable` e l'implementazione di `MqttClientPersistence` se non si utilizza `MqttDefaultFilePersistence`.

`MqttDefaultFilePersistence`

Il client MQTT fornisce la classe `MqttDefaultFilePersistence` . Se si crea un'istanza di `MqttDefaultFilePersistence` nell'applicazione client, è possibile fornire la directory per memorizzare i messaggi persistenti come parametro del costruttore `MqttDefaultFilePersistence` .

In alternativa, il client MQTT può istanziare `MqttDefaultFilePersistence` e posizionare i file nella seguente directory predefinita:

`client identifier` `-tcp` *`hostname portnumber`*

I seguenti caratteri vengono rimossi dalla stringa del nome directory:

`"\", "\\", "/", ":" e " "`

Il percorso della directory è il valore della proprietà di sistema `rcp.data`; se `rcp.data` non è impostato, il percorso è il valore della proprietà di sistema `usr.data`, dove

- `rcp.data` è una proprietà associata all'installazione di OSGi o Eclipse Rich Client Platform (RCP).
- `usr.data` è la directory in cui è stato avviato il comando Java che ha avviato l'applicazione.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria

Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Pubblicazioni

Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Un `MqttMessage` ha un array di byte come payload. Cercare di mantenere i messaggi il più piccoli possibile. La lunghezza massima del messaggio consentita da MQTT protocol è 250 MB.

Di solito, un programma client MQTT utilizza `java.lang.String` o `java.lang.StringBuffer` per manipolare il contenuto del messaggio. Per comodità, la classe `MqttMessage` ha un metodo `toString` per convertire il suo payload in una stringa. Per creare il payload dell'array di byte da un `java.lang.String` o `java.lang.StringBuffer`, utilizzare il metodo `getBytes` .

Il metodo `getBytes` converte una stringa nella serie di caratteri predefinita per la piattaforma. La serie di caratteri predefinita è generalmente `UTF-8`. Le pubblicazioni MQTT che contengono solo testo sono generalmente codificate in `UTF-8`. Utilizzare il metodo `getBytes("UTF8")` per sovrascrivere la serie di caratteri predefinita.

In IBM MQ, una pubblicazione MQTT viene ricevuta come messaggio `jms - bytes` . Il messaggio include una cartella `MQRFH2` contenente una cartella `<mqtt>`e una `<mqps>` . La cartella `<mqtt>` contiene `clientId`, `msgId`e `qos`, ma questo contenuto potrebbe cambiare in futuro.

Un `MqttMessage` ha tre attributi aggiuntivi: QoS (quality of service), se è conservato e se è un duplicato. L'indicatore duplicato viene impostato solo se la qualità del servizio è "almeno una volta" o "esattamente una volta". Se il messaggio è stato inviato in precedenza e non è stato riconosciuto abbastanza rapidamente dal client MQTT , il messaggio viene inviato di nuovo, con l'attributo duplicato impostato su `true`.

Editoriale

Per creare una pubblicazione in un'applicazione client MQTT , creare un `MqttMessage`. Imposta il payload, la qualità del servizio e se viene conservato e richiama il metodo `MqttTopic.publish(MqttMessage message)` ; viene restituito `MqttDeliveryToken` e il completamento della pubblicazione è asincrono.

In alternativa, il client MQTT può creare un oggetto messaggio temporaneo dai parametri sul metodo `MqttTopic.publish(byte [] payload, int qos, boolean retained)` quando crea una pubblicazione.

Se la pubblicazione ha una qualità del servizio "almeno una volta" o "esattamente una volta", `QoS=1` o `QoS=2`, il client MQTT richiama l'interfaccia `MqttClientPersistence` . Richiama `MqttClientPersistence` per memorizzare il messaggio prima di restituire un token di consegna all'applicazione.

L'applicazione può scegliere di bloccare fino a quando il messaggio non viene consegnato al server, utilizzando il metodo `MqttDeliveryToken.waitForCompletion` . In alternativa, l'applicazione può continuare senza bloccare. Se si desidera controllare se le pubblicazioni vengono consegnate, senza bloccare, registrare un'istanza di una classe di callback che implementa `MqttCallback` con il client MQTT . Il client MQTT richiama il metodo `MqttCallback.deliveryComplete` non appena la pubblicazione è stata consegnata. A seconda della qualità del servizio, la distribuzione potrebbe essere quasi immediata per `QoS=0`o potrebbe richiedere del tempo per `QoS=2`.

Utilizzare il metodo `MqttDeliveryToken.isComplete` per eseguire il polling se la consegna è completa. Mentre il valore di `MqttDeliveryToken.isComplete` è `false`, è possibile richiamare `MqttDeliveryToken.getMessage` per ottenere il contenuto del messaggio. Se il risultato della chiamata `MqttDeliveryToken.isComplete` è `true`, il messaggio è stato eliminato e la chiamata `MqttDeliveryToken.getMessage` genera un'eccezione di puntatore null. Non esiste alcuna sincronizzazione integrata tra `MqttDeliveryToken.getMessage` e `MqttDeliveryToken.isComplete`.

Se il client si disconnette prima di ricevere tutti i token di consegna in sospeso, una nuova istanza del client può interrogare i token di consegna in sospeso prima della connessione. Fino a quando il client non si connette, non vengono completate nuove consegne ed è sicuro chiamare `MqttDeliveryToken.getMessage`. Utilizzare il metodo `MqttDeliveryToken.getMessage` per scoprire quali pubblicazioni non sono state consegnate. I token di consegna in sospeso vengono eliminati se ci si connette con `MqttConnectOptions.cleanSession` impostato sul valore predefinito, `true`.

Sottoscrizione

Un gestore code è responsabile della creazione di pubblicazioni da inviare a una sottoscrizione MQTT . Il gestore code verifica se il filtro argomenti in una sottoscrizione creata da un client MQTT corrisponde alla stringa argomenti in una pubblicazione. La corrispondenza può essere una corrispondenza esatta oppure può includere caratteri jolly. Prima che la pubblicazione venga inoltrata al sottoscrittore dal gestore code, il gestore code controlla gli attributi argomento associati alla pubblicazione. Segue la procedura di ricerca descritta in Sottoscrizione mediante una stringa di argomenti contenente caratteri jolly per identificare se un oggetto argomento di gestione concede all'utente l'autorizzazione alla sottoscrizione.

Quando il client MQTT riceve una pubblicazione con QoS (quality of service) "almeno una volta", richiama il metodo `MqttCallback.messageArrived` per elaborare la pubblicazione. Se la qualità del servizio della pubblicazione è "esattamente una volta", `QoS=2`, il client MQTT richiama l'interfaccia `MqttClientPersistence` per memorizzare il messaggio quando viene ricevuto. Viene quindi chiamato `MqttCallback.messageArrived`.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Qualità del servizio fornito da un client MQTT

Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

La qualità del servizio di una pubblicazione è un attributo di `MqttMessage`. È impostato dal metodo `MqttMessage.setQos`.

Il metodo `MqttClient.subscribe` può ridurre la qualità del servizio applicato alle pubblicazioni inviate a un client su un argomento. La qualità del servizio di una pubblicazione inoltrata a un sottoscrittore

potrebbe essere diversa dalla qualità del servizio della pubblicazione. Il valore inferiore dei due valori viene utilizzato per inoltrare una pubblicazione.

Al massimo una volta
QoS=0

Il messaggio viene consegnato al massimo una volta oppure non viene consegnato. Il suo recapito sulla rete non è riconosciuto.

Il messaggio non è memorizzato. Se il client è disconnesso o se si verifica un errore nel server, il messaggio potrebbe andare perso.

QoS=0 è la modalità di trasferimento più veloce. A volte si chiama "fuoco e dimenticate".

MQTT protocol non richiede che i server inoltrino le pubblicazioni in QoS=0 a un client. Se il client è disconnesso nel momento in cui il server riceve la pubblicazione, la pubblicazione potrebbe essere eliminata, a seconda del server. Il servizio di telemetria (MQXR) non elimina i messaggi inviati con QoS=0. Vengono memorizzati come messaggi non persistenti e vengono eliminati solo se il gestore code viene arrestato.

Almeno una volta
QoS=1

QoS=1 è la modalità predefinita di trasferimento.

Il messaggio viene sempre consegnato almeno una volta. Se il mittente non riceve un riconoscimento, il messaggio viene inviato di nuovo con l'indicatore DUP impostato fino a quando non viene ricevuto un riconoscimento. Di conseguenza, il destinatario può essere inviato lo stesso messaggio più volte e potrebbe elaborarlo più volte.

Il messaggio deve essere memorizzato localmente sul mittente e sul destinatario fino a quando non viene elaborato.

Il messaggio viene eliminato dal destinatario dopo che questo ha elaborato il messaggio. Se il destinatario è un broker, il messaggio viene pubblicato per i relativi sottoscrittori. Se il destinatario è un client, il messaggio viene consegnato all'applicazione del sottoscrittore. Una volta eliminato il messaggio, il destinatario invia un riconoscimento al mittente.

Il messaggio viene eliminato dal mittente dopo aver ricevuto un riconoscimento dal destinatario.

Esattamente una volta
QoS=2

Il messaggio viene sempre consegnato esattamente una volta.

Il messaggio deve essere memorizzato localmente sul mittente e sul destinatario fino a quando non viene elaborato.

QoS=2 è la modalità di trasferimento più sicura ma più lenta. Richiede almeno due coppie di trasmissioni tra il mittente e il ricevente prima che il messaggio venga eliminato dal mittente. Il messaggio può essere elaborato dal destinatario dopo la prima trasmissione.

Nella prima coppia di trasmissioni, il mittente trasmette il messaggio e riceve il riconoscimento dal destinatario che ha memorizzato il messaggio. Se il mittente non riceve un riconoscimento, il messaggio viene inviato di nuovo con l'indicatore DUP impostato fino a quando non viene ricevuto un riconoscimento.

Nella seconda coppia di trasmissioni, il mittente comunica al ricevente che può completare l'elaborazione del messaggio, "PUBREL". Se il mittente non riceve un riconoscimento del messaggio "PUBREL" , il messaggio "PUBREL" viene inviato di nuovo fino a quando non viene ricevuto un riconoscimento. Il mittente elimina il messaggio salvato quando riceve il riconoscimento al messaggio "PUBREL"

Il destinatario può elaborare il messaggio nella prima o nella seconda fase, purché non rielabori il messaggio. Se il destinatario è un broker, pubblica il messaggio ai sottoscrittori. Se il destinatario è un client, consegna il messaggio all'applicazione del sottoscrittore. Il destinatario invia un messaggio di completamento al mittente che ha terminato l'elaborazione del messaggio.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT

Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Pubblicazioni conservate e client MQTT

Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Utilizzare il metodo `MqttMessage.setRetained` per specificare se viene conservata una pubblicazione su un argomento.

Quando si crea o si aggiorna una pubblicazione conservata, inviarla con un QoS di 1 o 2. Se lo si invia con un QoS pari a 0, IBM MQ crea una pubblicazione conservata non persistente. La pubblicazione non viene conservata se il gestore code viene arrestato.

Se si pubblica una pubblicazione non conservata in un argomento che ha una pubblicazione conservata, la pubblicazione conservata non viene influenzata. I sottoscrittori correnti ricevono la nuova pubblicazione. I nuovi sottoscrittori ricevono prima la pubblicazione conservata, quindi le nuove pubblicazioni.

È possibile utilizzare una pubblicazione conservata per registrare l'ultimo valore di una misura. I nuovi sottoscrittori di un argomento ricevono immediatamente il valore più recente della misura. Se non viene eseguita alcuna nuova misurazione dall'ultima sottoscrizione del sottoscrittore all'argomento della pubblicazione e se il sottoscrittore effettua nuovamente la sottoscrizione, il sottoscrittore riceve nuovamente la pubblicazione conservata più recente sull'argomento.

Per eliminare una pubblicazione conservata, sono disponibili due opzioni:

- Eseguire il comando MQSC **CLEAR TOPICSTR** .
- Creare una pubblicazione conservata di lunghezza zero. Come specificato nella specifica MQTT 3.1.1 , se un messaggio conservato di lunghezza zero viene pubblicato in un argomento, tutti i messaggi conservati per tale argomento vengono cancellati.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT

Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Sottoscrizioni

Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Creare sottoscrizioni utilizzando i metodi `MqttClient.subscribe` , passando uno o più filtri argomento e parametri QoS (quality of service). Il parametro QoS (quality of service) imposta la qualità massima del servizio che il sottoscrittore è pronto a utilizzare per ricevere un messaggio. I messaggi inviati a questo client non possono essere consegnati con una QoS (quality of service) superiore. La QoS (quality of service) è impostata sul valore inferiore del valore originale quando il messaggio è stato pubblicato e sul livello specificato per la sottoscrizione. La qualità del servizio predefinita per la ricezione dei messaggi è `QoS=1`, almeno una volta.

La richiesta di sottoscrizione viene inviata con `QoS=1`.

Le pubblicazioni vengono ricevute da un sottoscrittore quando il client MQTT richiama il metodo `MqttCallback.messageArrived` . Il metodo `messageArrived` inoltra anche la stringa di argomenti con cui il messaggio è stato pubblicato al sottoscrittore.

È possibile rimuovere una sottoscrizione o una serie o sottoscrizioni utilizzando i metodi `MqttClient.unsubscribe` .

Un comando IBM MQ può rimuovere una sottoscrizione. Elencare le sottoscrizioni utilizzando IBM MQ Explorero utilizzando i comandi **runmqsc** o PCF. Tutte le sottoscrizioni dei client MQTT vengono denominate. Viene fornito un nome del formato: *`ClientIdentifier:Topic name`*

Se si utilizza il valore predefinito `MqttConnectOptions`o si imposta `MqttConnectOptions.cleanSession` su `true` prima di connettere il client, tutte le vecchie sottoscrizioni per il client vengono rimosse quando il client si connette. Qualsiasi nuova sottoscrizione effettuata dal client durante la sessione viene rimossa quando si disconnette.

Se si imposta `MqttConnectOptions.cleanSession` su `false` prima della connessione, tutte le sottoscrizioni create dal client vengono aggiunte a tutte le sottoscrizioni esistenti per il client prima della connessione. Tutte le sottoscrizioni restano attive alla disconnessione del client.

Un altro modo per comprendere il modo in cui l'attributo `cleanSession` influenza le sottoscrizioni è considerarlo come un attributo modale. Nella sua modalità predefinita, `cleanSession=true`, il client crea sottoscrizioni e riceve pubblicazioni solo nell'ambito della sessione. Nella modalità alternativa, `cleanSession=false`, le sottoscrizioni sono durevoli. Il client può connettersi e disconnettersi e le sue sottoscrizioni rimangono attive. Quando il client si riconnette, riceve tutte le pubblicazioni non consegnate. Mentre è connesso, può modificare l'insieme di sottoscrizioni attive per suo conto.

È necessario impostare la modalità `cleanSession` prima della connessione; la modalità dura per l'intera sessione. Per modificarne l'impostazione, è necessario disconnettere e riconnettere il client. Se si modificano le modalità dall'utilizzo di `cleanSession=false` a `cleanSession=true`, tutte le sottoscrizioni precedenti per il client e tutte le pubblicazioni non ricevute vengono eliminate.

Le pubblicazioni che corrispondono alle sottoscrizioni attive vengono inviate al client non appena vengono pubblicate. Se il client è disconnesso, vengono inviati al client se si riconnette allo stesso server con lo stesso identificativo client e `MqttConnectOptions.cleanSession` impostato su `false`.

Le sottoscrizioni per un particolare cliente vengono identificate dall'identificativo client. È possibile riconnettere il client da un dispositivo client differente allo stesso server e continuare con la stessa sottoscrizione e ricevere pubblicazioni non consegnate.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni
Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Stringhe di argomento e filtri argomento nei client MQTT
Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Stringhe di argomento e filtri argomento nei client MQTT

Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM MQ.

Le stringhe di argomenti vengono utilizzate per inviare pubblicazioni ai sottoscrittori. Creare una stringa di argomenti utilizzando il metodo `MqttClient.getTopic(java.lang.String topicString)`.

I filtri argomento vengono utilizzati per sottoscrivere argomenti e ricevere pubblicazioni. I filtri argomento possono contenere caratteri jolly. Con i caratteri jolly, è possibile sottoscrivere più argomenti. Creare un filtro argomenti utilizzando un metodo di sottoscrizione; ad esempio, `MqttClient.subscribe(java.lang.String topicFilter)`.

Stringhe argomento

La sintassi di una stringa di argomenti IBM MQ è descritta in Stringhe di argomenti. La sintassi delle stringhe di argomenti MQTT è descritta nella classe `MqttClient` nella documentazione API per MQTT client for Java. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT.

La sintassi di ciascun tipo di stringa argomento è quasi identica. Ci sono quattro differenze minori:

1. Le stringhe degli argomenti inviate a IBM MQ dai client MQTT devono seguire le convenzioni per i nomi dei gestori code.
2. Le lunghezze massime differiscono. Le stringhe di argomenti IBM MQ sono limitate a 10.240 caratteri. Un client MQTT può creare stringhe di argomenti fino a 65535 byte.
3. Una stringa argomento creata da un client MQTT non può contenere un carattere null.
4. In IBM Integration Bus, un livello di argomento null, `'...//...'` non è valido. I livelli di argomento null sono supportati da IBM MQ.

A differenza della pubblicazione / sottoscrizione IBM MQ , il protocollo `mqttv3` non ha un concetto di oggetto argomento di gestione. Non è possibile costruire una stringa argomento da un oggetto argomento e una stringa argomento. Tuttavia, una stringa di argomenti viene associata a un argomento di amministrazione in IBM MQ. Il controllo accessi associato all'argomento di gestione determina se una pubblicazione viene pubblicata nell'argomento o scartata. Gli attributi che vengono applicati a una pubblicazione quando viene inoltrata ai sottoscrittori, sono influenzati dagli attributi dell'argomento di gestione.

Filtri argomento

La sintassi di un filtro argomento IBM MQ è descritta in Schema dei caratteri jolly basati sugli argomenti. La sintassi dei filtri argomento che puoi creare con un client MQTT è descritta nella classe `MqttClient` nella documentazione API per MQTT client for Java. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT.

Concetti correlati

Callback e sincronizzazione nelle applicazioni client MQTT
Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi da e verso il server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Ripulisci sessioni

Il client MQTT e il servizio di telemetria (MQXR) conservano le informazioni sullo stato della sessione. Le informazioni di stato vengono utilizzate per garantire "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT . È possibile scegliere di eseguire un client MQTT con o senza conservare le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Identificativo client
L'identificativo client è una stringa di 23 byte che identifica un client MQTT . Ogni ID deve essere univoco a un solo client connesso per volta. L'identificativo deve contenere solo caratteri validi in un nome gestore code. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione degli identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

Token di consegna

Ultimo testamento e pubblicazione testamentaria
Se una connessione client MQTT termina inaspettatamente, è possibile configurare MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Persistenza del messaggio nei client MQTT
I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta"o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

Pubblicazioni
Le pubblicazioni sono istanze di `MqttMessage` associate a una stringa di argomento. I client MQTT possono creare pubblicazioni da inviare a IBM MQe sottoscrivere argomenti su IBM MQ per ricevere pubblicazioni.

Qualità del servizio fornito da un client MQTT
Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a IBM MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

Pubblicazioni conservate e client MQTT
Un argomento può avere una sola pubblicazione conservata. Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione viene immediatamente inoltrata all'utente.

Sottoscrizioni
Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Sviluppo di applicazioni Microsoft Windows Communication Foundation con IBM MQ

Il canale personalizzato WCF (Microsoft Windows Communication Foundation) per IBM MQ invia e riceve messaggi tra i servizi e i client WCF.

Concetti correlati

“Introduzione al canale personalizzato IBM MQ per WCF con .NET” a pagina 1271
Il canale personalizzato per IBM MQ è un canale di trasporto che utilizza il modello di programmazione unificato WCF (Microsoft Windows Communication Foundation).

“Utilizzo di canali personalizzati IBM MQ per WCF” a pagina 1275

Panoramica delle informazioni disponibili per i programmatori che utilizzano i canali personalizzati IBM MQ per WCF (Windows Communication Foundation).

“Utilizzo degli esempi WCF” a pagina 1294
Gli esempi WCF (Windows Communication Foundation) forniscono alcuni semplici esempi di come può essere utilizzato il canale personalizzato IBM MQ .

FFST: WCF XMS First Failure Support Technology

Attività correlate

Traccia del canale personalizzato WCF per IBM MQ

Risoluzione dei problemi relativi al canale personalizzato WCF per i problemi IBM MQ

Introduzione al canale personalizzato IBM MQ per WCF con .NET

Il canale personalizzato per IBM MQ è un canale di trasporto che utilizza il modello di programmazione unificato WCF (Microsoft Windows Communication Foundation).

Il framework Microsoft Windows Communication Foundation, introdotto in Microsoft.NET 3, consente lo sviluppo di applicazioni e servizi .NET indipendentemente dal trasporto e dai protocolli utilizzati per connetterli, abilitando trasporti o configurazioni alternativi da utilizzare in base all'ambiente in cui viene distribuito il servizio o l'applicazione.

Le connessioni vengono gestite in fase di runtime da WCF creando uno stack del canale contenente la combinazione richiesta di:

- Elementi protocollo: una serie facoltativa di elementi in cui nessuno, uno o più possono essere aggiunti per supportare protocolli come gli standard WS - *.
- Codificatore del messaggio: un elemento obbligatorio nello stack che controlla la serializzazione del messaggio nel suo formato wire.
- Canale di trasporto: un elemento obbligatorio nello stack responsabile del trasferimento del messaggio serializzato al relativo endpoint.

Il canale personalizzato per IBM MQ è un canale di trasporto e, come tale, deve essere accoppiato con un codificatore di messaggi e protocolli facoltativi come richiesto dall'applicazione che utilizza un bind personalizzato WCF. In questo modo, le applicazioni che sono state sviluppate per utilizzare WCF possono utilizzare il canale personalizzato per IBM MQ per inviare e ricevere i dati nello stesso modo in cui utilizzano i trasporti integrati forniti da Microsoft, consentendo una semplice integrazione con le funzioni di messaggistica asincrona, scalabile e affidabile di IBM MQ. Per un elenco completo delle funzioni supportate, consultare “Funzionalità e caratteristiche del canale WCF Custom” a pagina 1275.

Quando e perché utilizzare il canale personalizzato IBM MQ per WCF?

È possibile utilizzare il canale personalizzato IBM MQ per inviare e ricevere messaggi tra client e servizi WCF nello stesso modo dei trasporti integrati forniti da Microsoft, consentendo alle applicazioni di accedere alle funzioni di IBM MQ all'interno del modello di programmazione unificato WCF.

Un tipico scenario di modello di utilizzo per il canale personalizzato IBM MQ per WCF è un'interfaccia non SOAP per la trasmissione di messaggi IBM MQ nativi.

Messaggi trasmessi utilizzando il formato del messaggio non SOAP/nonJMS (Pure MQMessage)

Quando si utilizza il canale personalizzato IBM MQ per WCF come interfaccia non - SOAP per la trasmissione di messaggi IBM MQ nativi, i messaggi vengono trasmessi utilizzando il formato messaggio Non - SOAP/Non-JMS (Pure MQMessage) IBM MQ.

Gli utenti WCF sono in grado di avviare il servizio o, in altre parole, gli utenti del servizio possono inviare un messaggio a una coda IBM MQ utilizzando MQMessages. Le applicazioni possono ottenere e impostare i campi MQMD e il payload. Quando il messaggio è disponibile nelle code IBM MQ , questo messaggio può essere elaborato da qualsiasi servizio WCF o da applicazioni non WCF come le applicazioni C o Java in esecuzione su AIX, Linux, Windowso z/OS.

Requisiti software per il canale personalizzato IBM MQ per WCF

Questo argomento illustra i requisiti software per il canale personalizzato IBM MQ per WCF. Il canale personalizzato IBM MQ per WCF può connettersi solo a gestori code IBM WebSphere MQ 7.0 o superiori.

Requisiti dell'ambiente di runtime

- Microsoft.NET Framework v4.7.2 o superiore deve essere installato sulla macchina host.
- *Java e .NET Messaging and Web Services* è installato per default come parte del programma di installazione di IBM MQ . Questo componente installa gli assembly .NET richiesti per il canale personalizzato nella cache di assembly globale.

Nota: Se Microsoft .NET Framework V4.7.2 o versioni successive non viene installato prima di installare IBM MQ, l'installazione del prodotto IBM MQ continua senza errori, ma IBM MQ classes for .NET non è disponibile. Se.NET Framework viene installato dopo l'installazione di IBM MQ, gli assemblaggi IBM MQ.NET devono essere registrati eseguendo lo script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , dove *WMQInstallDir* è la directory in cui è installato IBM MQ . Questo script installa gli assembly richiesti nella GAC (Global Assembly Cache). Una serie di file `amqi*.log` che registrano le azioni eseguite vengono creati nella directory %TEMP% . Non è necessario eseguire nuovamente lo script `amqiRegisterdotNet.cmd` se .NET viene aggiornato a V4.7.2 o versione successiva da una versione precedente, ad esempio da .NET V3.5.

Requisiti dell'ambiente di sviluppo

- Microsoft Visual Studio 2015 o Windows Software Development Kit per .NET 4.7.2 o versioni successive.
- Microsoft.NET Framework V4.7.2 o successivo deve essere installato sulla macchina host per creare i file della soluzione di esempio.

Canale personalizzato IBM MQ per WCF: cosa è installato?

Il canale personalizzato per IBM MQ è un canale di trasporto che utilizza il modello di programmazione unificato WCF (Microsoft Windows Communication Foundation). Il canale personalizzato viene installato per impostazione predefinita come parte dell'installazione.

Canale personalizzato IBM MQ per WCF

Il canale personalizzato e le sue dipendenze si trovano nel componente `Java and .NET Messaging and Web Services` , installato per impostazione predefinita. Quando si aggiorna IBM MQ da una versione precedente rispetto aIBM MQ 8.0, l'aggiornamento installa il canale personalizzato IBM MQ per WCF per impostazione predefinita se il componente `Java and .NET Messaging and Web Services` è stato precedentemente installato in una precedente installazione.

Il componente `.NET Messaging and Web Services` contiene il file `IBM.XMS.WCF.dll` e il file `IBM.WMQ.WCF.dll` e questi file sono l'assemblaggio del canale personalizzato principale, che contiene le classi dell'interfaccia WCF. Questi file sono installati nella GAC (Global Assembly Cache) e sono disponibili anche nella seguente directory: *MQ_INSTALLATION_PATH* `\bin` dove *MQ_INSTALLATION_PATH* è la directory in cui è installato IBM MQ .

La seguente tabella riepiloga le classi di chiavi richieste per l'utilizzo del canale personalizzato.

Tabella 189. Classi chiave richieste per l'utilizzo del canale personalizzato

	Interfaccia SOAP/JMS (esistente)	**Interfaccia non - SOAP/Non-JMS (da IBM MQ 8.0)**
Assemblaggio canale personalizzato	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll

Tabella 189. Classi chiave richieste per l'utilizzo del canale personalizzato (Continua)		
	Interfaccia SOAP/JMS (esistente)	**Interfaccia non - SOAP/Non-JMS (da IBM MQ 8.0)**
Nome bind trasporto	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Programma di importazione binding di trasporto	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Configurazione binding di trasporto	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Esempi (Oneway)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client, MQMessaging_OneWay_Service
Esempi (RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	Client MQMessaging_RequestReply_Client, MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll supporta le interfacce SOAP/JMS e Non - SOAP/Non-JMS . Le nuove applicazioni sviluppate sono consigliate per utilizzare IBM.WMQ.WCF poiché supporta entrambe le interfacce.

Invio di messaggi formattati MQSTR

Se il messaggio di richiesta è di tipo MQSTR, è possibile scegliere di inviare il messaggio di risposta in formato MQSTR.

È necessario utilizzare un parametro URI aggiuntivo **replyMessageFormat** per modificare il formato del messaggio di risposta. I valori supportati sono:

""

"" è il valore predefinito.

Il messaggio di risposta è in formato byte (MQMFT_NONE). Ad esempio:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR

Il messaggio di risposta è in formato MQSTR (MQMFT_STRING). Ad esempio:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

Note:

1. Il valore per **replyMessageFormat** non è sensibile al maiuscolo / minuscolo.
2. L'utilizzo di un qualsiasi valore diverso da "" o *MQSTR*, causa un'eccezione di valore del parametro non valido.

Esempi di canale personalizzato IBM MQ

Gli esempi forniscono alcuni semplici esempi di come può essere utilizzato il canale personalizzato IBM MQ per WCF. Gli esempi e i relativi file associati si trovano nella directory *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf` , dove *MQ_INSTALLATION_PATH* è la directory di installazione per

IBM MQ. Per ulteriori informazioni sugli esempi di canale personalizzato IBM MQ , consultare “Utilizzo degli esempi WCF” a pagina 1294.

svcutil.exe.config

`svcutil.exe.config` è un esempio delle impostazioni di configurazione richieste per consentire allo strumento di creazione proxy del client Microsoft WCF `svcutil` di riconoscere il canale personalizzato. Il file `svcutil.exe.config` si trova nella directory *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\` , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ. Per ulteriori informazioni sull'utilizzo di `svcutil.exe.config`, consultare “Generazione di un proxy client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con metadati da un servizio in esecuzione” a pagina 1292.

Architettura WCF

Il canale personalizzato IBM MQ per WCF è integrato nella parte superiore dell'API `IBM Message Service Client for .NET (XMS .NET)`.

Interfaccia SOAP/JMS

L'architettura WCF è quella mostrata nel diagramma seguente:

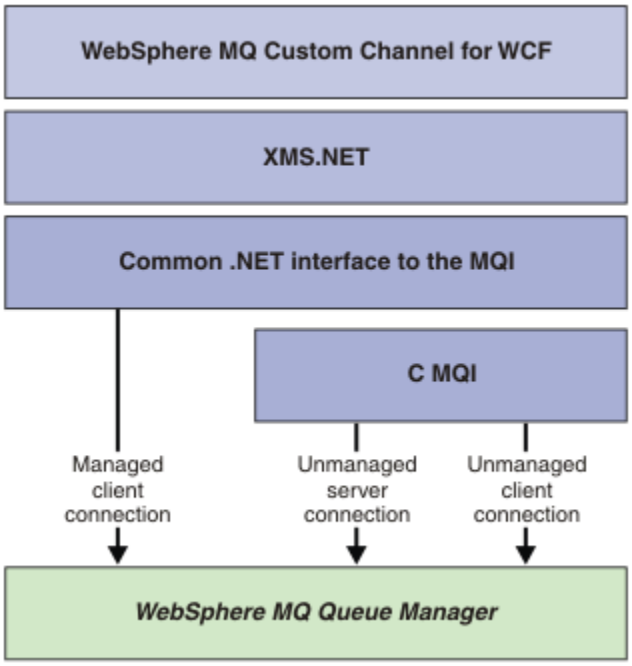


Figura 149. Architettura WCF per interfaccia SOAP/JMS

Tutti i componenti richiesti vengono installati per impostazione predefinita con l'installazione del prodotto.

Le tre connessioni sono:

- Connessioni client gestite
- Connessioni server non gestite
- Connessioni client non gestite

Per ulteriori informazioni su queste connessioni, consultare “Opzioni di connessione WCF” a pagina 1281.

Interfaccia non - SOAP/Non-JMS

Il canale personalizzato IBM MQ per WCF supporta sia l'interfaccia SOAP/JMS (disponibile da IBM WebSphere MQ 7.0.1) e l'interfaccia Non - SOAP/Non-JMS .

L'architettura WCF è quella mostrata nel diagramma seguente:

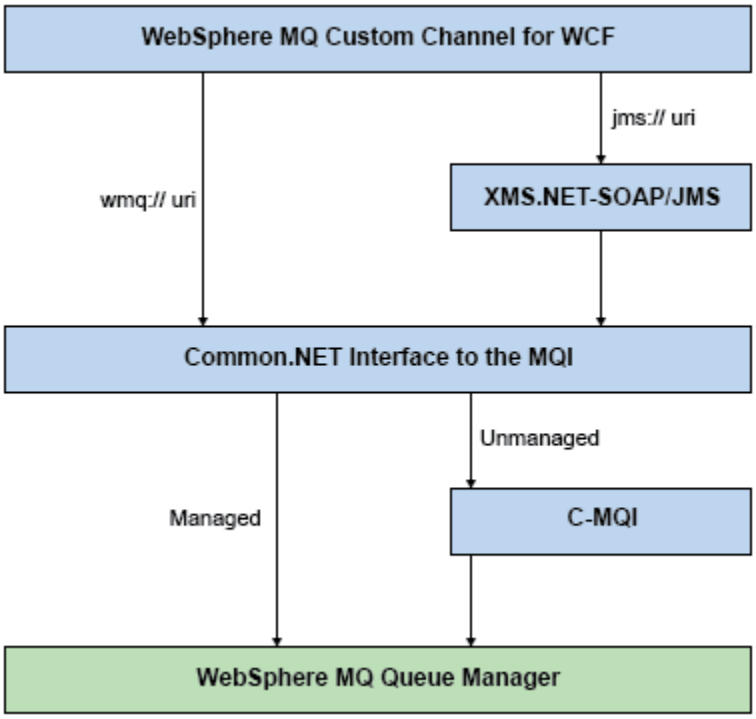


Figura 150. Architettura WCF per interfaccia Non - SOAP/Non-JMS

Utilizzo di canali personalizzati IBM MQ per WCF

Panoramica delle informazioni disponibili per i programmatori che utilizzano i canali personalizzati IBM MQ per WCF (Windows Communication Foundation).

Microsoft Windows Communication Foundation supporta i servizi Web e il supporto della messaggistica in Microsoft.NET Framework 3. IBM MQ può essere utilizzato come un canale personalizzato all'interno di WCF in .NET Framework 3 allo stesso modo dei canali integrati offerti da Microsoft.

I messaggi trasportati nel canale personalizzato vengono formattati in base all'implementazione SOAP su JMS di IBM MQ. Le applicazioni possono quindi comunicare con i servizi ospitati da WCF o dall'infrastruttura del servizio WebSphere SOAP over JMS .

Funzionalità e caratteristiche del canale WCF Custom

Utilizzare i seguenti argomenti per informazioni relative alle funzioni e alle funzioni del canale personalizzato WCF.

Forme canale personalizzate WCF

Panoramica delle forme di canale personalizzate che IBM MQ può utilizzare come all'interno dei canali personalizzati WCF (Microsoft Windows Communication Foundation).

Il canale personalizzato IBM MQ per WCF supporta due forme di canale:

- Unidirezionale
- Richiesta-Risposta

WCF seleziona automaticamente la forma del canale in base al contratto di servizio che si sta ospitando.

I contratti che includono metodi che utilizzano solo il parametro **IsOneWay** vengono serviti dalla forma del canale unidirezionale, ad esempio:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

I contratti che includono una combinazione di metodi unidirezionali e di richiesta - risposta, o tutti i metodi di richiesta - risposta, vengono serviti dalla forma del canale richiesta - risposta. Ad esempio:

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

Nota: Quando si combinano metodi unidirezionali e di richiesta - risposta nello stesso contratto, è necessario assicurarsi che il comportamento sia quello previsto, in particolare quando si lavora in un ambiente misto perché i metodi unidirezionali attendono che ricevano una risposta null dal servizio.

Canale unidirezionale

Il canale personalizzato unidirezionale IBM MQ per WCF viene utilizzato, ad esempio, per inviare messaggi da un client WCF utilizzando una forma di canale unidirezionale. Il canale può inviare messaggi solo in una direzione, ad esempio, da un gestore code client a un servizio WCF.

Canale di richiesta - risposta

Il canale personalizzato di richiesta - risposta IBM MQ per WCF viene utilizzato, ad esempio, per inviare messaggi in due direzioni in modo asincrono; la stessa istanza client deve essere utilizzata per la messaggistica asincrona. Il canale può inviare messaggi in una direzione, ad esempio, da un gestore code client a una coda su un servizio WCF e quindi inviare un messaggio di risposta da WCF a una coda sul gestore code client.

Valori e nomi parametro URI WCF

Nomi e valori dei parametri URI per l'interfaccia SOAP/JMS e l'interfaccia Non - SOAP/Non JMS .

Interfaccia SOAP/JMS

connectionFactory
Il parametro connectionFactory è richiesto.

initialContextFactory
Il parametro initialContextFactory è obbligatorio e deve essere impostato su "com.ibm.mq.jms.Nojndi" per la compatibilità con WebSphere Application Server e altri prodotti.

Interfaccia non SOAP/Non JMS

Il formato URI è quello delle specifiche MA93 . Consultare SupportPac - MA93 per ulteriori informazioni sulle specifiche IRI IBM MQ .

Sintassi URI IBM MQ

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
```

```
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

Esempio IRI IBM MQ

Il seguente IRI di esempio indica a un richiedente del servizio che può utilizzare una connessione IBM MQ TCP client - binding a una macchina denominata example.com sulla porta 1414 e inserire i messaggi di richiesta persistenti in una coda denominata SampleQ sul gestore code QM1. L'IRI specifica che il provider del servizio inserirà le risposte in una coda denominata SampleReplyQ.

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

Per connessioni abilitate TLS

Per effettuare connessioni protette (TLS) utilizzando il servizio / client WCF, impostare le seguenti proprietà con i valori appropriati nell'URI. Tutte le proprietà che hanno come prefisso "*" sono obbligatorie per stabilire una connessione protetta.

- **sslKeyRepository**: *SYSTEM o *USER
- * **sslCipherSpec**: un CipherSpecvalido, ad esempio TLS_RSA_WITH_AES_128_CBC_SHA256.
- **sslCertRevocationCheck**: true o false.
- **sslKeyResetCount**: un valore maggiore di 32kb.
- **sslPeerName:** il DN del certificato server

Ad esempio:

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

Distribuzione garantita canale personalizzato WCF

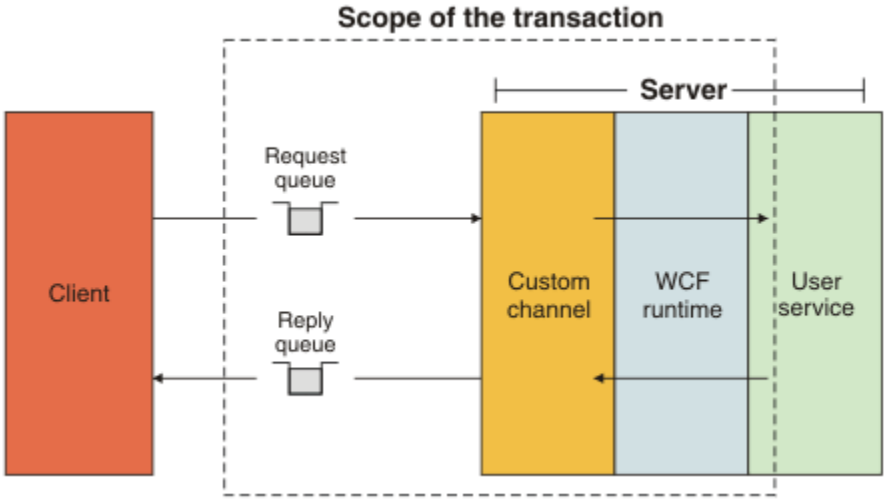
La consegna garantita garantisce che una richiesta di servizio o risposta è azionata e non persa.

Viene ricevuto un messaggio di richiesta e qualsiasi messaggio di risposta viene inviato in un punto di sincronizzazione della transazione locale, di cui è possibile eseguire il rollback in caso di errore di runtime. Esempi di questi errori sono: un'eccezione non gestita generata dal servizio, un errore di invio del messaggio al servizio o un errore di consegna del messaggio di risposta.

`AssuredDelivery` è l'attributo di consegna garantita che può essere specificato su un contratto di servizio per garantire che tutti i messaggi di richiesta ricevuti da un servizio e qualsiasi messaggio di risposta inviato da un servizio, non vadano persi in caso di errore di runtime.

Per garantire che i messaggi vengano conservati anche in caso di errore di sistema o di interruzione dell'alimentazione, i messaggi devono essere inviati come persistenti. Per utilizzare i messaggi persistenti, l'applicazione client deve avere questa opzione specificata sull'URI dell'endpoint.

Le transazioni distribuite non sono supportate e l'ambito della transazione non si estende oltre l'elaborazione del messaggio di richiesta e risposta eseguita da IBM MQ. Qualsiasi lavoro eseguito all'interno del servizio potrebbe essere rieseguito come risultato di un errore che causa la ricezione del messaggio. Il seguente diagramma mostra l'ambito della transazione:



La consegna garantita viene abilitata applicando l'attributo `AssuredDelivery` alla classe di servizio come mostrato nel seguente esempio:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

Quando si utilizza l'attributo `AssuredDelivery` , è necessario essere consapevoli dei punti seguenti:

- Quando un canale determina che è probabile che un errore si ripeta se un messaggio è stato sottoposto a rollback e ricevuto di nuovo, il messaggio viene trattato come un messaggio non elaborabile e non viene restituito alla coda di richieste per la rielaborazione. Ad esempio: se il messaggio ricevuto non è formattato correttamente o non può essere inviato ad un servizio. Le eccezioni non gestite generate da un'operazione di servizio vengono sempre reinviate fino a quando il messaggio non viene riconsegnato il numero massimo di volte specificato dalla proprietà della soglia di backout della coda di richieste. Per ulteriori informazioni, consultare “Messaggi non elaborabili del canale personalizzato WCF” a pagina 1279
- Il canale esegue la lettura, l'elaborazione e la risposta di ogni messaggio di richiesta come un'operazione atomica utilizzando un singolo thread di esecuzione per applicare l'integrità transazionale. Per abilitare l'esecuzione simultanea delle operazioni di servizio, il canale consente a WCF di creare più istanze del canale. Il numero di istanze del canale disponibili per l'elaborazione delle richieste è controllato dalla proprietà di bind `MaxConcurrentCalls`. Per ulteriori informazioni, consultare “Opzioni di configurazione del bind WCF” a pagina 1287
- La funzione di distribuzione garantita utilizza sia i punti di estendibilità IOperationInvoker che IErrorHandler WCF. Se questi punti di estendibilità vengono utilizzati esternamente da un'applicazione, l'applicazione deve garantire che vengano richiamati tutti i punti di estendibilità precedentemente registrati. La mancata esecuzione di questa operazione per IErrorHandler può causare errori non notificati. Se non si riesce a eseguire questa operazione per IOperationInvoker , WCF potrebbe non rispondere più.

Sicurezza del canale personalizzato WCF

Il canale personalizzato IBM MQ per WCF supporta l'utilizzo di TLS solo per le connessioni client non gestite al gestore code.

Specificare TLS utilizzando una voce nella CCDT (client channel definition table). Per ulteriori informazioni sui CCDT, consultare Tabella di definizione di canale client.

Tabelle di definizione del canale client WCF (CCDT)

Il canale personalizzato IBM MQ per WCF supporta l'utilizzo di CCDT (client channel definition tables) per configurare le informazioni di collegamento per le connessioni client.

I CCDT sono controllati tramite queste due variabili di ambiente:

- *MQCHLLIB* specifica la directory in cui si trova la tabella.
- *MQCHLTAB* specifica il nome file della tabella.

Se queste variabili di ambiente sono definite, hanno la priorità sui dettagli di connessione client specificati nell'URI.

Per ulteriori informazioni sulle tabelle di definizione del canale client, consultare: Tabella di definizione del canale client.

Messaggi non elaborabili del canale personalizzato WCF

Quando un servizio non riesce a elaborare un messaggio di richiesta o a consegnare un messaggio di risposta a una coda di risposta, il messaggio viene trattato come un messaggio non elaborabile.

Messaggi di richiesta poison

Se un messaggio di richiesta non può essere elaborato, viene trattato come un messaggio non elaborabile. Questa azione impedisce al servizio di ricevere nuovamente lo stesso messaggio non elaborabile. Affinché un messaggio di richiesta non elaborabile venga trattato come un messaggio non elaborabile, una delle seguenti situazioni deve essere true:

- Il conteggio di backout dei messaggi ha superato la soglia di backout specificata sulla coda di richieste, che si verifica solo se è stata specificata la consegna garantita per il servizio. Per ulteriori informazioni sulla consegna sicura, consultare: “Distribuzione garantita canale personalizzato WCF” a pagina 1277
- Il messaggio non è stato formattato correttamente e non può essere interpretato come messaggio SOAP su JMS .

Messaggi di risposta non elaborabili

Se un servizio non riesce a consegnare un messaggio di risposta alla coda di risposta, il messaggio di risposta viene trattato come un messaggio non elaborabile. Per i messaggi di risposta, questa azione consente ai messaggi di risposta di essere richiamati in un secondo momento per facilitare la determinazione dei problemi.

Gestione messaggi non elaborabili

L'azione intrapresa per un messaggio non elaborabile dipende dalla configurazione del gestore code e dai valori impostati nelle opzioni di report del messaggio. Per SOAP su JMS, le seguenti opzioni di report sono impostate sui messaggi di richiesta per impostazione predefinita e non sono configurabili:

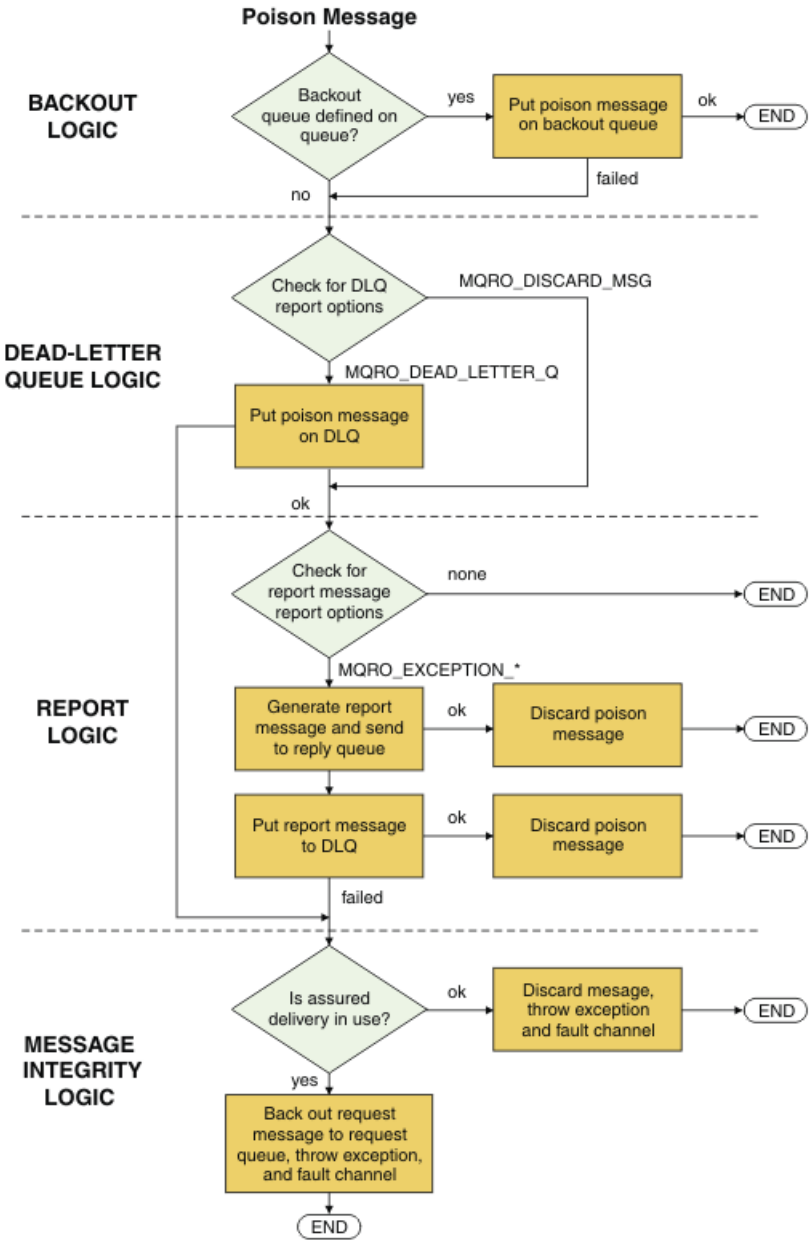
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Per SOAP su JMS, la seguente opzione di report è impostata sui messaggi di risposta per impostazione predefinita e non è configurabile:

- MQRO_DEAD_LETTER_Q

Se i messaggi provengono da un'origine non WCF, fare riferimento alla documentazione per tale origine.

Il seguente diagramma mostra le azioni possibili e le operazioni eseguite se la gestione dei messaggi non elaborabili ha esito negativo:



Funzionalità dei messaggi di IBM MQ per le applicazioni WCF

Non - SOAP/Non -JMS (ovvero, IBM MQ) capacità dei messaggi per applicazioni WCF.

Per l'interfaccia Non - SOAP/Non-JMS , le funzionalità dei messaggi IBM MQ per applicazioni WCF sono le seguenti:

- Le applicazioni WCF possono inviare e ricevere i messaggi IBM MQ di base che possono essere elaborati da qualsiasi applicazione IBM MQ .
- Le applicazioni WCF hanno il controllo completo per aggiornare MQMD e payload.
- Il client WCF può inviare messaggi IBM MQ che possono essere utilizzati da qualsiasi client IBM MQ , ad esempio i client C, Java, JMSe .NET .

L'interfaccia WCF per Non - SOAP/Non-JMS deve utilizzare le seguenti classi per impostare il payload del messaggio e MQMD per il messaggio:

- WmqStringMessaggio per un payload di tipo String
- Messaggio WmqBytesper un payload di tipo Bytes
- WmqXmlMessaggio per un payload di tipo XML

Per impostare il payload del messaggio, utilizzare la proprietà **Data** per il messaggio WmqString, WmqBytesMessage o la classe di messaggi WmqXml, in base al tipo di payload. Ad esempio, utilizzare il seguente codice per impostare un payload di tipo String:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

Opzioni di connessione WCF

Esistono tre modi per connettere un canale personalizzato IBM MQ per WCF a un gestore code. Considerare il tipo di connessione più adatto alle proprie esigenze.

Per ulteriori informazioni sulle opzioni di connessione, consultare: “Differenze di connessione” a pagina 580

Per ulteriori informazioni sull'architettura WCF, consultare: “Architettura WCF” a pagina 1274

Connessione client non gestito

Una connessione effettuata in questa modalità si connette come un client IBM MQ a un server IBM MQ in esecuzione sulla macchina locale o su una macchina remota.

Per utilizzare il canale personalizzato IBM MQ per WCF come un IBM MQ client, è possibile installarlo, con IBM MQ MQI client, sul server IBM MQ o su una macchina separata.

Connessione server non gestita

Quando viene utilizzato in modalità bind del server, il canale personalizzato IBM MQ per WCF utilizza l'API del gestore code, piuttosto che comunicare attraverso una rete. L'utilizzo delle connessioni dei collegamenti fornisce prestazioni migliori per applicazioni IBM MQ rispetto all'utilizzo delle connessioni di rete.

Per utilizzare la connessione dei bind, è necessario installare il canale personalizzato IBM MQ per WCF sul server IBM MQ .

Connessione client gestito

Una connessione effettuata in questa modalità si connette come un client IBM MQ a un server IBM MQ in esecuzione sulla macchina locale o su una macchina remota.

Le classi del canale personalizzato IBM MQ per .NET 3 che si collegano in questa modalità rimangono nel codice gestito .NET e non effettuano chiamate a servizi nativi. Per ulteriori informazioni sul codice gestito, consultare la documentazione Microsoft .

Esistono diverse limitazioni all'utilizzo del client gestito. Per ulteriori informazioni su queste limitazioni, consultare “Connessioni client gestite” a pagina 580.

Creazione e configurazione del canale personalizzato IBM MQ per WCF

I canali personalizzati IBM MQ per WCF funzionano allo stesso modo dei canali WCF di trasporto offerti da Microsoft. Il canale personalizzato IBM MQ per WCF può essere creato in uno dei due modi.

Informazioni su questa attività

Il canale personalizzato IBM MQ si integra con WCF come un canale di trasporto WCF e come tale deve essere accoppiato con un codificatore di messaggi e canali di protocollo facoltativi, in modo da poter creare uno stack di canali completo che può essere utilizzato da un'applicazione. Sono necessari due elementi per creare correttamente uno stack del canale completo:

1. Una definizione di bind: specifica quali elementi sono richiesti per creare lo stack del canale delle applicazioni, incluso il canale di trasporto, il codificatore dei messaggi e i protocolli, oltre a tutte le impostazioni di configurazione generali. Per il canale personalizzato, la definizione di bind deve essere creata sotto forma di un bind personalizzato WCF.
2. Una definizione di endpoint: collega il contratto di servizi con la definizione di bind e fornisce anche l'URI di connessione effettivo che descrive dove l'applicazione può connettersi. Per il canale personalizzato, l'URI è nel formato SOAP su URI JMS .

Queste definizioni possono essere create in due modi diversi:

- Amministrativamente; le definizioni vengono create fornendo i dettagli in un file di configurazione dell'applicazione (ad esempio: `app.config`).
- In modo programmatico; le definizioni vengono create direttamente dal codice dell'applicazione.

La decisione su quale metodo utilizzare per creare le definizioni deve essere basata sui requisiti della domanda come segue:

- Il metodo di gestione per la configurazione fornisce la flessibilità per modificare i dettagli del servizio e del client dopo la distribuzione senza ricreare l'applicazione.
- Il metodo programmatico per la configurazione fornisce una maggiore protezione dagli errori di configurazione e la possibilità di creare dinamicamente una configurazione in fase di runtime.

Creazione amministrativa di un canale personalizzato WCF fornendo informazioni sul bind e sull'endpoint in un file di configurazione dell'applicazione

Il canale personalizzato IBM MQ per WCF è un canale WCF a livello di trasporto. Un endpoint e un bind devono essere definiti per utilizzare il canale personalizzato e queste definizioni possono essere eseguite fornendo le informazioni sul bind e sull'endpoint in un file di configurazione dell'applicazione.

Per configurare e utilizzare il canale personalizzato IBM MQ per WCF, che è un canale WCF a livello di trasporto, è necessario definire un bind e una definizione di endpoint. Il bind contiene le informazioni di configurazione per il canale e la definizione endpoint contiene i dettagli di connessione. Queste definizioni possono essere create in due modi:

- Programmaticamente direttamente dal codice dell'applicazione, come descritto di seguito: “Creazione di un canale personalizzato WCF mediante fornitura programmatica di informazioni di binding ed endpoint” a pagina 1284
- Amministrativamente, fornendo i dettagli in un file di configurazione dell'applicazione, come descritto nella seguente procedura.

Il file di configurazione dell'applicazione del servizio o del client è comunemente denominato *yourappname*`.exe.config` dove *yourappname* è il nome della tua applicazione. Il file di configurazione dell'applicazione viene modificato più facilmente utilizzando lo strumento dell'editor di configurazione servizio Microsoft denominato `SvcConfigEditor.exe` nel seguente modo:

- Avviare lo strumento dell'editor di configurazione `SvcConfigEditor.exe` . Il percorso di installazione predefinito per lo strumento è: *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe` dove *Unità:* è il nome dell'unità di installazione.

Passo 1: aggiungi un'estensione dell'elemento di bind per abilitare WCF a individuare il canale personalizzato

1. Fare clic con il tasto destro del mouse su **Avanzate** > **Estensione** > **elemento di collegamento** per aprire il menu e selezionare **Nuovo**
2. Completare i campi come mostrato in questa tabella:

Tabella 190. Nuovi campi dell'elemento di collegamento

Campo	**Valore**
Nome	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Tabella 190. Nuovi campi dell'elemento di collegamento (Continua)	
Campo	**Valore**
Tipo	Passare a `IBM.XMS.WCF.dll` nella GAC (Global Assembly Cache) e selezionare `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig`

Passo 2: crea una definizione di bind personalizzato che accoppia il canale personalizzato con un codificatore di messaggi WCF

1. Fare clic con il tasto destro del mouse su **Bind** per aprire il menu e selezionare **Nuova configurazione bind**
2. Completare i campi come mostrato in questa tabella:

Tabella 191. Nuovi campi di configurazione del collegamento	
Campo	**Valore**
Nome	`CustomBinding_WMQ`
BindingElement 1	`textMessageEncoding (MessageVersion: Soap11)`
BindingElement 2	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Passo 3: Specificare le proprietà di binding

1. Selezionare *IBM.XMS.WCF.SoapJmsIbmTransportChannel* dal collegamento creato in “Passo 2: crea una definizione di bind personalizzato che accoppia il canale personalizzato con un codificatore di messaggi WCF” a pagina 1283
2. Apportare le modifiche richieste ai valori predefiniti delle proprietà come descritto in: “Opzioni di configurazione del bind WCF” a pagina 1287

Passo 4: crea una definizione di endpoint

Crea una definizione di endpoint che fa riferimento al bind personalizzato che hai creato in “Passo 2: crea una definizione di bind personalizzato che accoppia il canale personalizzato con un codificatore di messaggi WCF” a pagina 1283 e fornisce i dettagli di connessione del servizio. Il modo in cui queste informazioni vengono specificate dipende dal fatto che la definizione sia per un'applicazione client o per un'applicazione di servizio.

Per un'applicazione client, aggiungere una definizione di endpoint alla sezione client nel modo seguente:

1. Fare clic col pulsante destro del mouse su **Client** > **Endpoint** per aprire il menu e selezionare **Nuovo endpoint client**
2. Completare i campi come mostrato in questa tabella:

Tabella 192. Nuovi campi endpoint client	
Campo	**Valore**
Nome	`Endpoint_WMQ`
Indirizzo	*L'URI SOAP/JMS che descrive i dettagli di connessione WMQ richiesti per accedere al servizio.* Per ulteriori dettagli, consultare: “Canale personalizzato IBM MQ per il formato dell'indirizzo URI dell'endpoint WCF” a pagina 1286
Binding	`customBinding`

Tabella 192. Nuovi campi endpoint client (Continua)	
Campo	**Valore**
BindingConfiguration	`CustomBinding_WMQ`
Contratto	*Il nome dell'interfaccia del contratto di servizio*

Per un'applicazione di servizio, aggiungere una definizione di servizio alla sezione dei servizi nel modo seguente:

1. Fare clic con il tasto destro del mouse su **Servizi** per aprire il menu e selezionare **Nuovo servizio**, quindi selezionare la classe di servizio da ospitare.
2. Aggiungere una definizione di endpoint alla sezione **Endpoint** per il nuovo servizio e completare i campi come mostrato in questa tabella:

Tabella 193. Nuovi campi endpoint servizio	
Campo	**Valore**
Nome	`Endpoint_WMQ`
Indirizzo	*L'URI SOAP/JMS che descrive i dettagli di connessione WMQ richiesti per accedere al servizio.* Per ulteriori dettagli, consultare: "Canale personalizzato IBM MQ per il formato dell'indirizzo URI dell'endpoint WCF" a pagina 1286
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contratto	*Il nome della classe di implementazione del servizio*

Creazione di un canale personalizzato WCF mediante fornitura programmatica di informazioni di binding ed endpoint

Il canale personalizzato IBM MQ per WCF è un canale WCF a livello di trasporto. È necessario definire un endpoint e un bind per utilizzare il canale personalizzato e queste definizioni possono essere eseguite in modo programmatico direttamente dal codice dell'applicazione.

Per configurare e utilizzare il canale personalizzato IBM MQ per WCF, che è un canale WCF a livello di trasporto, è necessario definire un bind e una definizione di endpoint. Il bind contiene le informazioni di configurazione per il canale e la definizione endpoint contiene i dettagli di connessione. Per ulteriori informazioni, fare riferimento a "Utilizzo degli esempi WCF" a pagina 1294.

Queste definizioni possono essere create in due modi:

- Amministrativamente, fornendo i dettagli in un file di configurazione dell'applicazione, come descritto in "Creazione amministrativa di un canale personalizzato WCF fornendo informazioni sul bind e sull'endpoint in un file di configurazione dell'applicazione" a pagina 1282.
- Programmaticamente direttamente dal codice dell'applicazione, come descritto nei seguenti topic secondari.

Definizione programmatica delle informazioni di binding ed endpoint: interfaccia SOAP/JMS

Per l'interfaccia SOAP/JMS , è possibile definire un endpoint e il bind in modo programmatico direttamente dal codice applicazione.

Informazioni su questa attività

Per fornire le informazioni sul bind e sull'endpoint in modo programmatico, aggiungi il codice richiesto alla tua applicazione completando la seguente procedura.

Procedura

1. Creare un'istanza dell'elemento di bind del trasporto del canale aggiungendo il seguente codice alla propria applicazione:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. Impostare le proprietà di collegamento richieste, ad esempio, aggiungendo il seguente codice all'applicazione per impostare `ClientConnectionMode`:

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. Creare un bind personalizzato che accoppia il canale di trasporto con un codificatore di messaggi aggiungendo il codice seguente all'applicazione:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. Creare l'URI SOAP/JMS .

   L'URI SOAP/JMS che descrive i dettagli di connessione IBM MQ richiesti per accedere al servizio, deve essere fornito come indirizzo endpoint. L'indirizzo specificato dipende dal fatto che il canale venga utilizzato per un'applicazione di servizio o per un'applicazione client.

   - Per le applicazioni client, l'URI SOAP/JMS deve essere creato come `EndpointAddress` come segue:

```
EndpointAddress address = new EndpointAddress("jms:/queue?
destination=SampleQ@QM1&connectionFactory
=connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

   - Per le applicazioni di servizi, l'URI SOAP/JMS deve essere creato come URI come segue:

```
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

   Per ulteriori informazioni sugli indirizzi endpoint, consultare “Canale personalizzato IBM MQ per il formato dell'indirizzo URI dell'endpoint WCF” a pagina 1286.

Definizione programmatica delle informazioni di binding ed endpoint: interfaccia Non - SOAP/Non-JMS
Per l'interfaccia Non - SOAP/Non-JMS , è possibile definire un endpoint e il bind in modo programmatico direttamente dal codice dell'applicazione.

Informazioni su questa attività

Per fornire le informazioni sul bind e sull'endpoint in modo programmatico, aggiungi il codice richiesto alla tua applicazione completando la seguente procedura.

Procedura

1. Crea un WmqBinding aggiungendo il seguente codice all'applicazione:

```
WmqBinding binding = new WmqBinding();
```

   Questo codice crea un bind che accoppia l'elemento WmqMsgEncodingElement e l'elemento WmqIbmTransportBindingrichiesto per l'interfaccia Non - SOAP/Non-JMS .
2. Fornisci l'URI wmq: // che descrive i dettagli di connessione IBM MQ richiesti per accedere al servizio.

Il modo in cui si fornisce l'URI wmq: // dipende dal fatto che il canale venga utilizzato per un'applicazione del servizio o per un'applicazione del client.

- Per le applicazioni client, l'URI wmq: // deve essere creato come EndpointAddress come segue:

```
EndpointAddress address = new EndpointAddress
("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

- Per le applicazioni di servizio, l'URI wmq: // deve essere creato come URI come segue:

```
Uri sampleAddress = new Uri(
"wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

Canale personalizzato IBM MQ per il formato dell'indirizzo URI dell'endpoint WCF

Un servizio Web viene specificato utilizzando un URI (Universal Resource Identifier) che fornisce dettagli di connessione e ubicazione. Il formato dell'URI dipende dal fatto che si stia utilizzando l'interfaccia SOAP/JMS o l'interfaccia Non - SOAP/Non-JMS .

Interfaccia SOAP/JMS

Il formato URI supportato nel trasporto IBM MQ per SOAP consente un grado completo di controllo su opzioni e parametri specifici SOAP/ IBM MQ durante l'accesso ai servizi di destinazione. Questo formato è compatibile con WebSphere Application Server e con CICS, facilitando l'integrazione di IBM MQ con entrambi questi prodotti.

La sintassi URI è la seguente:

```
jms:/queue? name=value&name=value...
```

dove name è un nome parametro e *value* è un valore appropriato e l'elemento name = *value* può essere ripetuto un qualsiasi numero di volte con la seconda e le successive ricorrenze precedute da una e commerciale (&).

I nomi dei parametri sono sensibili al maiuscolo / minuscolo, così come i nomi degli oggetti IBM MQ . Se un parametro viene specificato più di una volta, la ricorrenza finale del parametro diventa effettiva, il che significa che le applicazioni client possono sovrascrivere i valori del parametro accodando all'URI. Se vengono inclusi ulteriori parametri non riconosciuti, vengono ignorati.

Se si memorizza un URI in una stringa XML, è necessario rappresentare il carattere e commerciale come "&". Allo stesso modo, se un URI è codificato in uno script, fare attenzione a eseguire l'escape di caratteri come **&** che altrimenti sarebbero interpretati dalla shell.

Questo è un esempio di un URI semplice per un servizio Axis:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Di seguito è riportato un esempio di URI semplice per un servizio .NET :

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Vengono forniti solo i parametri richiesti (`targetService` è obbligatorio solo per i servizi .NET) e a `connectionFactory` non viene fornita alcuna opzione.

In questo esempio di asse, `connectionFactory` contiene una serie di opzioni:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

In questo esempio di asse, è stata specificata anche l'opzione `sslPeerName` di `connectionFactory` . Il valore di sslPeerName stesso contiene coppie nome - valore e significativi spazi incorporati:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Interfaccia NON - SOAP/Non-JMS

Il formato URI per l'interfaccia NON - SOAP/Non-JMS consente un grado completo di controllo su opzioni e parametri specifici di IBM MQ quando si accede ai servizi di destinazione.

La sintassi URI è la seguente:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

Questo IRI indica a un richiedente del servizio che può utilizzare un collegamento client IBM MQ TCP a una macchina denominata example.com sulla porta 1415 e inserire i messaggi di richiesta persistenti in una coda denominata INS.QUOTE.REQUEST sul gestore code MOTOR.INS. L'IRI specifica che il provider del servizio inserisce le risposte a una coda denominata INS.QUOTE.REPLY sul gestore code BRANCH452. Il formato URI è quello specificato per SupportPac MA93. Consultare SupportPac MA93: IBM MQ - Definizione servizio per ulteriori dettagli sulle specifiche IRI IBM MQ .

Opzioni di configurazione del bind WCF

Esistono due modi per applicare le opzioni di configurazione alle informazioni di bind dei canali personalizzati. È possibile impostare le proprietà in modo amministrativo o programmatico.

Le opzioni di configurazione del collegamento possono essere impostate in due diversi modi:

1. Amministrativamente: le impostazioni della proprietà di collegamento devono essere specificate nella sezione di trasporto della definizione di collegamento personalizzato nel file di configurazione delle applicazioni, ad esempio `app.config`.
2. Programmaticamente: il codice applicazione deve essere modificato per specificare la proprietà durante l'inizializzazione del bind personalizzato.

Impostazione delle proprietà di collegamento in modo amministrativo

Le impostazioni della proprietà di binding possono essere specificate nel file di configurazione dell'applicazione, ad esempio `app.config`. Il file di configurazione viene generato da **`svcutil`**, come mostrato nei seguenti esempi.

Interfaccia SOAP/JMS

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

Interfaccia non - SOAP/Non-JMS

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

Impostazione programmatica delle proprietà di collegamento

Per aggiungere una proprietà di collegamento WCF per specificare la modalità di connessione client, è necessario modificare il codice servizio per specificare la proprietà durante l'inizializzazione del collegamento personalizzato.

Utilizzare il seguente esempio per specificare la modalità di connessione client non gestita:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                        transportBindingElement);
```

Proprietà bind WCF

Tabella 194. Valori delle proprietà di collegamento durante l'impostazione amministrativamente o programmaticamente

Nome proprietà	Applicazione client o servizio	Valore amministrativo	Valore programmatico	Descrizione
maxBufferPoolSize	Entrambi	Numero intero con segno da 0 a 64 bit	Numero intero con segno da 0 a 64 bit	Specifica la dimensione massima della memoria che può essere utilizzata per memorizzare i buffer di messaggi WCF per una istanza del canale.
maxMessageSize	Entrambi	Numero intero con segno da 1 a 32 bit	Numero intero con segno da 1 a 32 bit	Specifica la memoria massima che può essere utilizzata per un singolo messaggio WCF.
Modalità clientConnection	Entrambi	0 (Valore predefinito) 1	AS_URI (valore predefinito) CLIENT_NON gestito	Specifica la modalità di connessione client del canale di trasporto. 0 indica che la modalità di collegamento client è quella specificata nell'URI. Utilizzato solo se viene utilizzata la connessione client. Specifica che la modalità di connessione client è quella specificata nell'URI. 0 è il valore predefinito se non è impostata alcuna modalità di connessione client. 1 indica che la modalità di connessione client è un client non gestito. Utilizzato solo se viene utilizzata la connessione client.

Tabella 194. Valori delle proprietà di collegamento durante l'impostazione amministrativamente o programmaticamente (Continua)

Nome proprietà	Applicazione client o servizio	Valore amministrativo	Valore programmatico	Descrizione
Chiamate MaxConcurrent	Client	L'intervallo è 0-2 147 483 647 16 è il valore predefinito	L'intervallo è 0-2 147 483 647 16 è il valore predefinito	Questa proprietà definisce il numero massimo di operazioni simultanee che possono essere eseguite su un singolo proxy client in qualsiasi momento. Se vengono avviate più operazioni, queste vengono accodate fino al completamento o al timeout di un'operazione in corso. Questa impostazione può essere utilizzata per controllare il numero massimo di thread e risorse che possono essere utilizzati da un singolo proxy. `0` rimuove questo limite, consentendo il tentativo simultaneo di tutte le operazioni.
Chiamate MaxConcurrent	Servizio	L'intervallo è 1-2 147 483 647 16 è il valore predefinito	L'intervallo è 1-2 147 483 647 16 è il valore predefinito	Questa proprietà viene utilizzata solo se la funzione di consegna garantita è abilitata (per ulteriori informazioni sulla consegna garantita, consultare “Distribuzione garantita canale personalizzato WCF” a pagina 1277). Specifica il massimo numero di operazioni simultanee che possono essere in corso contemporaneamente per l'endpoint specificato. È necessario prestare attenzione quando si modifica questa impostazione. Ogni operazione simultanea richiede ulteriori risorse, in particolare una nuova istanza del canale personalizzato e i thread associati dal pool di thread per eseguire le richieste. L'allocazione eccessiva può essere controproducente e influire gravemente sulle prestazioni. È necessario eseguire la configurazione appropriata del pool di thread per supportare questa proprietà.

Servizi di costruzione e hosting per WCF

Panoramica dei servizi WCF (Microsoft Windows Communication Foundation) che spiega come creare e configurare i servizi WCF.

Il canale personalizzato IBM MQ per WCF e i servizi WCF che lo utilizzano possono essere ospitati dai seguenti metodi:

- Self - hosting
- Servizio Windows

Il canale personalizzato IBM MQ per WCF non può essere ospitato in Windows Process Activation Service.

I seguenti argomenti forniscono alcuni semplici esempi di auto - hosting per dimostrare le fasi coinvolte. La documentazione in linea di Microsoft WCF, che contiene ulteriori informazioni e gli ultimi dettagli, è disponibile sul sito Web di Microsoft MSDN all'indirizzo https://msdn.microsoft.com.

Creazione di applicazioni di servizio WCF utilizzando il metodo 1: hosting autonomo mediante un file di configurazione dell'applicazione

Dopo aver creato un file di configurazione dell'applicazione, apri un'istanza del servizio e aggiungi il codice specificato alla tua applicazione.

Prima di iniziare

Creare o modificare un file di configurazione dell'applicazione per il servizio, come descritto in: "Creazione amministrativa di un canale personalizzato WCF fornendo informazioni sul bind e sull'endpoint in un file di configurazione dell'applicazione" a pagina 1282

Informazioni su questa attività

1. Istanziare e aprire un'istanza del servizio nell'host del servizio. Il tipo di servizio deve essere uguale al tipo di servizio specificato nel file di configurazione del servizio.
2. Aggiungere il codice seguente alla propria applicazione:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

Creazione di applicazioni di servizio WCF utilizzando il metodo 2: Self - hosting programmaticamente direttamente dall'applicazione

Aggiungere le proprietà di bind, creare l'host di servizio con un'istanza della classe di servizio richiesta e aprire il servizio.

Prima di iniziare

1. Aggiungere un riferimento al canale personalizzato `IBM.XMS.WCF.dll` al progetto. `IBM.XMS.WCF.dll` si trova in *`WMQInstallDir`*`\bin` dove *WMQInstallDir* è la directory in cui è installato IBM MQ .
2. Aggiungere un'istruzione *using* al namespace `IBM.XMS.WCF` , ad esempio: `using IBM.XMS.WCF`
3. Creare un'istanza dell'elemento bind dei canali e dell'endpoint come descritto in "Creazione di un canale personalizzato WCF mediante fornitura programmatica di informazioni di binding ed endpoint" a pagina 1284

Informazioni su questa attività

Se sono richieste delle modifiche alle proprietà di bind del canale, completare la seguente procedura:

1. Aggiungere le proprietà di collegamento a `transportBindingElement` come mostrato nel seguente esempio:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Creare l'host di servizio con un'istanza della classe di servizio richiesta:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. Apri il servizio:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

Esposizione di metadati utilizzando un endpoint HTTP

Istruzioni per esporre i metadati di un servizio configurato per utilizzare il canale personalizzato IBM MQ per WCF.

Informazioni su questa attività

Se i metadati dei servizi devono essere esposti (in modo che gli strumenti come `svcutil` possano accedervi direttamente dal servizio in esecuzione piuttosto che da un file WSDL non in linea, ad esempio), devono essere esposti i metadati dei servizi con un endpoint HTTP . I seguenti passi possono essere utilizzati per aggiungere questo endpoint aggiuntivo.

1. Aggiungere l'indirizzo di base in cui i metadati devono essere esposti a ServiceHost, ad esempio:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Aggiungere il seguente codice a ServiceHost prima che il servizio venga aperto:

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

Risultati

I metadati sono ora disponibili al seguente indirizzo: `http://localhost:8000/MyService`

Creazione di applicazioni client per WCF

Panoramica sulla generazione e la generazione di applicazioni client Microsoft Windows Communication Foundation (WCF).

Un'applicazione client può essere creata per un servizio WCF; le applicazioni client vengono generalmente generate utilizzando lo Strumento di utilità metadati Microsoft ServiceModel (Svcutil.exe) per creare la configurazione richiesta e i file proxy che possono essere utilizzati direttamente dall'applicazione.

Generazione di un proxy client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con metadati da un servizio in esecuzione

Istruzioni per utilizzare lo strumento Microsoft svcutil.exe per generare un client per un servizio configurato per utilizzare il canale personalizzato IBM MQ per WCF.

Prima di iniziare

Esistono tre prerequisiti per l'utilizzo dello strumento svcutil per creare i file di configurazione e proxy richiesti che possono essere utilizzati direttamente dall'applicazione:

- Il servizio WCF deve essere in esecuzione prima dell'avvio dello strumento svcutil.
- Il servizio WCF deve esporre i suoi metadati utilizzando una porta HTTP in aggiunta ai riferimenti endpoint del canale personalizzato IBM MQ per generare direttamente un client da un servizio in esecuzione.
- Il canale personalizzato deve essere registrato nei dati di configurazione per svcutil.

Informazioni su questa attività

La seguente procedura spiega come generare un client per un servizio configurato per utilizzare il canale personalizzato IBM MQ , ma espone anche i relativi metadati al runtime tramite una porta HTTP separata:

1. Avviare il Servizio WCF (il Servizio deve essere in esecuzione prima dell'avvio dello strumento svcutil).
2. Aggiungere i dettagli dal file di configurazione `svcutil.exe` dalla root dell'installazione al file di configurazione svcutil attivo, in genere `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` in modo che svcutil riconosca il canale personalizzato IBM MQ .
3. Eseguire svcutil da un prompt dei comandi, ad esempio:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. Copiare i file `app.config` e `YourService.cs` generati nel progetto client Microsoft Visual Studio.

Operazioni successive

Se i metadati dei servizi non possono essere richiamati direttamente, svcutil può essere utilizzato per generare i file del client da wsdl. Per ulteriori informazioni, consultare: “Generazione di un proxy del client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con WSDL” a pagina 1292

Generazione di un proxy del client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con WSDL

Istruzioni per la generazione di client WCF da WSDL se i metadati del servizio non sono disponibili.

Se i metadati del servizio non possono essere richiamati direttamente per generare un client dai metadati di un servizio in esecuzione, è possibile utilizzare svcutil per generare i file client da WSDL. Le seguenti modifiche devono essere apportate al WSDL per specificare che deve essere utilizzato il canale personalizzato IBM MQ :

1. Aggiungere le seguenti definizioni di spazio dei nomi e informazioni sulla normativa:

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>
```

```
...

</wsdl:definitions>
```

2. Modificare la sezione dei collegamenti per fare riferimento alla nuova sezione della politica e rimuovere qualsiasi definizione `transport` dall'elemento di collegamento sottostante:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Eseguire svcutil da un prompt dei comandi, ad esempio:

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

Creazione di applicazioni del client WCF utilizzando un proxy client con un file di configurazione dell'applicazione

Prima di iniziare

Creare o modificare un file di configurazione dell'applicazione per il client, come descritto in: “Creazione amministrativa di un canale personalizzato WCF fornendo informazioni sul bind e sull'endpoint in un file di configurazione dell'applicazione” a pagina 1282

Informazioni su questa attività

Istanziare e aprire un'istanza del proxy client. Il parametro passato al proxy generato deve corrispondere al nome endpoint specificato nel file di configurazione client, ad esempio `Endpoint_WMQ`:

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

Creazione di applicazioni client WCF utilizzando un proxy client con configurazione programmatica

Prima di iniziare

1. Aggiungere un riferimento al canale personalizzato `IBM.XMS.WCF.dll` al progetto. `IBM.XMS.WCF.dll` si trova nella directory *WMQInstallDir*`\bin` dove *WMQInstallDir* è la directory in cui è installato IBM MQ .
2. Aggiungere un'istruzione *using* al namespace `IBM.XMS.WCF` , ad esempio: `using IBM.XMS.WCF`

3. Creare un'istanza dell'elemento di collegamento e dell'endpoint del canale come descritto in “Creazione di un canale personalizzato WCF mediante fornitura programmatica di informazioni di binding ed endpoint” a pagina 1284

Informazioni su questa attività

Se sono richieste delle modifiche alle proprietà di bind del canale, completare la seguente procedura.

1. Aggiungere le proprietà di collegamento a `transportBindingElement` come mostrato nella seguente figura:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Creare il proxy client come mostrato nella seguente figura, dove *binding* e *indirizzo endpoint* sono il bind e l'indirizzo endpoint configurati nel passo 1 e passati:

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
           try {
                  myClient.myMethod("HelloWorld!");
                  myClient.Close();
           }
           catch (TimeoutException e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
           catch (CommunicationException e) {
               Console.Out.WriteLine(e);
                              myClient.Abort();
           }
           catch (Exception e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
```

Utilizzo degli esempi WCF

Gli esempi WCF (Windows Communication Foundation) forniscono alcuni semplici esempi di come può essere utilizzato il canale personalizzato IBM MQ .

Per creare progetti di esempio, è necessario l'SDK Microsoft.NET 3.5 o Microsoft Visual Studio 2008.

Esempio WCF del server e del client a una via semplice

Questo esempio dimostra il canale personalizzato IBM MQ utilizzato per avviare un servizio WCF (Windows Communication foundation) da un client WCF utilizzando una forma di canale unidirezionale.

Informazioni su questa attività

Il servizio implementa un singolo metodo che restituisce una stringa alla console. Il client è stato generato utilizzando lo strumento `svcutil` per richiamare i metadati del servizio da un endpoint HTTP esposto separatamente, come descritto in “Generazione di un proxy client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con metadati da un servizio in esecuzione” a pagina 1292

L'esempio è stato configurato con nomi risorsa specifici come descritto nella seguente procedura. Se è necessario modificare i nomi delle risorse, è necessario modificare anche il valore corrispondente sull'applicazione client nel file *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` e sull'applicazione di servizio nel file *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` , dove

MQ_INSTALLATION_PATH è la directory di installazione per IBM MQ. Per ulteriori informazioni sulla formattazione dell'URI dell'endpoint JMS , consultare *IBM MQ Transport for SOAP* nella documentazione del prodotto IBM MQ . Se è necessario modificare l'origine e la soluzione di esempio, è necessario un IDE, ad esempio Microsoft Visual Studio 8 o superiore.

Procedura

1. Creare un gestore code denominato *QM1*
2. Creare una destinazione code denominata *SampleQ*
3. Avviare il servizio in modo che il listener sia in attesa di messaggi: eseguire il file *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ.
4. Eseguire il client una sola volta: eseguire il file *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ.

   L'applicazione client esegue un loop cinque volte inviando cinque messaggi a *SampleQ*

Risultati

L'applicazione del servizio riceve i messaggi da *SampleQ* e visualizza `Hello World` sullo schermo cinque volte.

Operazioni successive

Esempio WCF del server e del client di richiesta - risposta semplice

Questo esempio illustra il canale personalizzato di IBM MQ utilizzato per avviare un servizio Windows Communication foundation (WCF) da un client WCF utilizzando una forma di canale richiesta - risposta.

Informazioni su questa attività

Questo servizio fornisce alcuni semplici metodi di calcolo per aggiungere e sottrarre due numeri, quindi restituire il risultato. Il client è stato generato utilizzando lo strumento `svcutil` per richiamare i metadati del servizio da un endpoint HTTP esposto separatamente, come descritto in "Generazione di un proxy client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con metadati da un servizio in esecuzione" a pagina 1292

L'esempio è stato configurato con nomi risorsa specifici come descritto nella seguente procedura. Se è necessario modificare i nomi delle risorse, è anche necessario modificare il valore corrispondente sull'applicazione client nel file *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config` e sull'applicazione di servizio nel file *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ. Per ulteriori informazioni sulla formattazione dell'URI dell'endpoint JMS , consultare *IBM MQ Transport for SOAP* nella documentazione del prodotto IBM MQ . Se è necessario modificare l'origine e la soluzione di esempio, è necessario un IDE, ad esempio Microsoft Visual Studio 8 o superiore.

Procedura

1. Creare un gestore code denominato *QM1*
2. Creare una destinazione code denominata *SampleQ*
3. Creare una destinazione code denominata *SampleReplyQ*
4. Avviare il servizio in modo che il listener sia in attesa di messaggi: eseguire il file *MQ_INSTALLATION_PATH*

\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ.

5. Eseguire il client una sola volta: eseguire il file *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ.

Risultati

Quando il client è stato eseguito, viene avviato il seguente processo e si ripete quattro volte in modo che un totale di cinque messaggi venga inviato in ogni modo:

1. Il client inserisce un messaggio di richiesta su *SampleQ* e attende una risposta.
2. Il servizio riceve il messaggio di richiesta da *SampleQ*.
3. Il servizio aggiunge e sottrae alcuni valori utilizzando il contenuto del messaggio.
4. Il servizio inserisce i risultati in un messaggio su *SampleReplyQ*e attende che il client inserisci un nuovo messaggio.
5. Il client riceve il messaggio da *SampleReplyQ* e visualizza i risultati sullo schermo.

Operazioni successive

Client WCF a un servizio .NET ospitato dall'esempio IBM MQ

Le applicazioni client di esempio e le applicazioni proxy di servizio di esempio sono fornite sia per .NET che per Java. Gli esempi si basano su un servizio Stock Quote che acquisisce una richiesta per una quotazione in borsa e quindi fornisce la quotazione in borsa.

Prima di iniziare

L'esempio richiede che l'ambiente di hosting del servizio .NET SOAP over JMS sia correttamente installato e configurato in IBM MQ e sia accessibile da un gestore code locale.

Quando l'ambiente host del servizio .NET SOAP su JMS viene correttamente installato e configurato in IBM MQ ed è accessibile da un gestore code locale, è necessario completare ulteriori fasi di configurazione.

1. Impostare la variabile di ambiente **WMQSOAP_HOME** sulla directory di installazione IBM MQ , ad esempio: `C:\Program Files\IBM\MQ`
2. Verificare che il Java compilatore `javac` sia disponibile e su PATH.
3. Copiare il file `axis.jar` dalla directory `prereqs/axis` dell'immagine di installazione alla directory di produzione IBM MQ , ad esempio: `C:\Program Files\IBM\MQ\java\lib\soap`
4. Aggiungere al percorso: *MQ_INSTALLATION_PATH*`\Java\lib` dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ , ad esempio: `C:\Program Files\IBM\MQ`
5. Verificare che l'ubicazione di .NET sia specificata correttamente in *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ , ad esempio: `C:\Program Files\IBM\MQ`. La posizione di .NET può essere specificata ad esempio: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Una volta completati i passi precedenti, verificare ed eseguire il servizio:

1. Passare alla directory di lavoro SOAP over JMS .
2. Immettere uno dei seguenti comandi per eseguire il test di verifica e lasciare in esecuzione il listener di servizio:
   - Per .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ .
   - Per AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ .

L'argomento `hold` mantiene i listener in esecuzione una volta completato il test.

Se vengono riportati degli errori durante questa configurazione, è possibile rimuovere tutte le modifiche in modo che la procedura possa essere riavviata nel seguente modo:

1. Eliminare la directory SOAP over JMS generata.
2. Eliminare il gestore code.

Informazioni su questa attività

Questo esempio illustra una connessione da un client WCF al servizio di esempio .NET SOAP over JMS fornito in IBM MQ utilizzando una forma di canale unidirezionale. Il servizio implementa un semplice esempio StockQuote , che restituisce una stringa di testo alla console.

Il client è stato generato utilizzando WSDL per generare i file client come descritto in "Generazione di un proxy del client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con WSDL" a pagina 1292

L'esempio è stato configurato con nomi risorsa specifici come descritto nella seguente procedura. Se è necessario modificare i nomi delle risorse, è necessario modificare anche il valore corrispondente nell'applicazione client nel file *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\app.config` e nell'applicazione di servizio nel file *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` , dove *MQ_INSTALLATION_PATH* rappresenta la directory di installazione per IBM MQ. Per ulteriori informazioni sulla formattazione dell'URI dell'endpoint JMS , consultare *IBM MQ Transport for SOAP* nella documentazione del prodotto IBM MQ .

Procedura

Eseguire il client una volta: eseguire il file *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe` , dove *MQ_INSTALLATION_PATH* rappresenta la directory di installazione per IBM MQ.

L'applicazione client esegue un loop cinque volte inviando cinque messaggi alla coda di esempio.

Risultati

L'applicazione di servizio richiama i messaggi dalla coda di esempio e visualizza `Hello World` cinque volte sullo schermo.

Client WCF per un servizio Axis Java ospitato dall'esempio IBM MQ

Le applicazioni client di esempio e le applicazioni proxy di servizio di esempio sono fornite sia per Java che per .NET. Gli esempi si basano su un servizio Stock Quote che acquisisce una richiesta per una quotazione in borsa e quindi fornisce la quotazione in borsa.

Prima di iniziare

Questo esempio richiede che l'ambiente di hosting del servizio .NET SOAP over JMS sia installato e configurato correttamente in IBM MQ e sia accessibile da un gestore code locale.

Quando l'ambiente host del servizio .NET SOAP su JMS viene correttamente installato e configurato in IBM MQ ed è accessibile da un gestore code locale, è necessario completare ulteriori fasi di configurazione.

1. Impostare la variabile di ambiente **WMQSOAP_HOME** sulla directory di installazione IBM MQ , ad esempio: `C:\Program Files\IBM\MQ`
2. Verificare che il Java compilatore `javac` sia disponibile e su PATH.
3. Copiare il file `axis.jar` dalla directory `prereqs/axis` dell'immagine di installazione alla directory di installazione IBM MQ .

4. Aggiungere al percorso: *MQ_INSTALLATION_PATH*\Java\lib dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ , ad esempio: C:\Program Files\IBM\MQ
5. Verificare che l'ubicazione di .NET sia specificata correttamente in *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd , dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ , ad esempio: C:\Program Files\IBM\MQ. La posizione di .NET può essere specificata ad esempio: set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin

Una volta completati i passi precedenti, verificare ed eseguire il servizio:

1. Passare alla directory di lavoro SOAP over JMS .
2. Immettere uno dei seguenti comandi per eseguire il test di verifica e lasciare in esecuzione il listener di servizio:
   - Per .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ .
   - Per AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold dove *MQ_INSTALLATION_PATH* rappresenta la directory in cui è installato IBM MQ .

L'argomento hold mantiene i listener in esecuzione una volta completato il test.

Se durante questa configurazione vengono riportati degli errori, è possibile rimuovere tutte le modifiche in modo che la procedura venga riavviata nel modo seguente:

1. Eliminare la directory SOAP over JMS generata.
2. Eliminare il gestore code.

Informazioni su questa attività

L'esempio mostra una connessione da un client WCF al servizio di esempio Axis Java SOAP over JMS fornito in IBM MQ utilizzando una forma di canale unidirezionale. Il servizio implementa un esempio StockQuote semplice, che emette una stringa di testo in un file salvato nella directory corrente.

Il client è stato generato utilizzando WSDL per generare i file client come descritto in "Generazione di un proxy del client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con WSDL" a pagina 1292

L'esempio è stato configurato con nomi di risorsa specifici, come descritto in questo paragrafo. Se è necessario modificare i nomi delle risorse, è necessario modificare anche il valore corrispondente sull'applicazione client nel file *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\app.config e sull'applicazione del servizio nel file *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl , dove *MQ_INSTALLATION_PATH* rappresenta la directory di installazione per IBM MQ.

Procedura

Eseguire il client una volta: eseguire il file *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe , dove *MQ_INSTALLATION_PATH* rappresenta la directory di installazione per IBM MQ.

L'applicazione client esegue un loop cinque volte inviando cinque messaggi alla coda di esempio.

Risultati

L'applicazione del servizio riceve i messaggi dalla coda di esempio e aggiunge Hello World cinque volte a un file nella directory corrente.

Client WCF per il servizio Java ospitato dall'esempio WebSphere Application Server

Le applicazioni client di esempio e le applicazioni proxy del servizio di esempio vengono fornite per WebSphere Application Server 6. Viene fornito anche un servizio richiesta - risposta.

Prima di iniziare

Questo esempio richiede che venga utilizzata la seguente configurazione IBM MQ :

Tabella 195. IBM MQ Configurazione richiesta

Oggetto	**Nome richiesto**
Gestore code	QM1
Coda locale	HelloWorld
Coda locale	Risposta HelloWorld

Questo esempio richiede anche che un ambiente host WebSphere Application Server 6 sia installato e configurato correttamente. WebSphere Application Server 6 utilizza una connessione in modalità bind per connettersi a IBM MQ per impostazione predefinita. Pertanto, WebSphere Application Server 6 deve essere installato sulla stessa macchina del gestore code.

Una volta configurato l'ambiente WAS, è necessario completare le seguenti operazioni di configurazione aggiuntive:

1. Creare i seguenti oggetti JNDI nel repository JNDI WebSphere Application Server :
   a. Una destinazione coda JMS denominata `HelloWorld`
      - Impostare il nome JNDI su `jms/HelloWorld`
      - Impostare il nome della coda su `HelloWorld`
   b. Un factory di connessione code JMS denominato `HelloWorldQCF`
      - Impostare il nome JNDI su `jms/HelloWorldQCF`
      - Impostare il nome del gestore code su `QM1`
   c. Un factory di connessione code JMS denominato `WebServicesReplyQCF`
      - Impostare il nome JNDI su `jms/WebServicesReplyQCF`
      - Impostare il nome del gestore code su `QM1`
2. Creare una porta listener dei messaggi denominata `HelloWorldPort` in WebSphere Application Server con la configurazione seguente:
   - Impostare il nome JNDI del factory di connessione su `jms/HelloWorldQCF`
   - Impostare il nome JNDI di destinazione su `jms/HelloWorld`
3. Installare l'applicazione `HelloWorldEJBEAR.ear` del servizio Web su WebSphere Application Server nel modo seguente:
   a. Fare clic su **Applicazioni** > **Nuova applicazione** > **Nuova applicazione enterprise**.
   b. Passare a *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` , dove *MQ_INSTALLATION_PATH* è la directory di installazione di IBM MQ.
   c. Non modificare alcuna opzione predefinita nella procedura guidata e riavviare il server delle applicazioni dopo che l'applicazione è stata installata.

Una volta completata la configurazione di WAS, verificare il servizio eseguendolo una volta:

1. Passare alla directory di lavoro Soap over JMS .

2. Immettere questo comando per eseguire l'esempio: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe` , dove *MQ_INSTALLATION_PATH* è la directory di installazione di IBM MQ.

Informazioni su questa attività

L'esempio illustra una connessione da un client WCF al servizio di esempio WebSphere Application Server SOAP over JMS fornito negli esempi WCF inclusi in IBM MQ, utilizzando una forma di canale richiesta - risposta. Il flusso di messaggi tra WCF e WebSphere Application Server che utilizza code IBM MQ . Il servizio implementa il metodo `HelloWorld(...)` , che prende una stringa e restituisce un messaggio di saluto al client.

Il client è stato generato utilizzando lo strumento svcutil per richiamare i metadati del servizio da un endpoint HTTP esposto separatamente, come descritto in "Generazione di un proxy client WCF e di file di configurazione dell'applicazione utilizzando lo strumento svcutil con metadati da un servizio in esecuzione" a pagina 1292

L'esempio è stato configurato con nomi risorsa specifici come descritto nella seguente procedura. Se è necessario modificare i nomi delle risorse, è necessario modificare anche il valore corrispondente sull'applicazione client nel file *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\app.config` e sull'applicazione del servizio nel *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` , dove *MQ_INSTALLATION_PATH* è la directory di installazione di IBM MQ.

Il servizio e il client si basano sul servizio e sul client descritti nell'articolo IBM Developer *Building a JMS web service using SOAP over JMS e WebSphere Studio*. Per ulteriori informazioni sullo sviluppo di servizi Web SOAP su JMS compatibili con il canale personalizzato IBM MQ WCF, consultare https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

Procedura

Eseguire il client una sola volta: eseguire il file *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe` , dove *MQ_INSTALLATION_PATH* è la directory di installazione per IBM MQ.

L'applicazione client avvia entrambi i metodi di servizio contemporaneamente, inviando due messaggi alla coda di esempio.

Risultati

L'applicazione del servizio ottiene i messaggi dalla coda di esempio e fornisce una risposta alla chiamata del metodo `HelloWorld(...)` che l'applicazione del client emette sulla console.

Informazioni particolari

Queste informazioni sono state sviluppate per prodotti e servizi offerti negli Stati Uniti.

IBM potrebbe non offrire i prodotti, i servizi o le funzioni descritti in questo documento in altri paesi. Consultare il rappresentante IBM locale per informazioni sui prodotti e sui servizi disponibili nel proprio paese. Ogni riferimento relativo a prodotti, programmi o servizi IBM non implica che solo quei prodotti, programmi o servizi IBM possano essere utilizzati. In sostituzione a quelli forniti da IBM possono essere usati prodotti, programmi o servizi funzionalmente equivalenti che non comportino la violazione dei diritti di proprietà intellettuale o di altri diritti dell'IBM. Tuttavia, è responsabilità dell'utente valutare e verificare il funzionamento di qualsiasi prodotto, programma o servizio non IBM.

IBM potrebbe disporre di applicazioni di brevetti o brevetti in corso relativi all'argomento descritto in questo documento. La fornitura di tale documento non concede alcuna licenza a tali brevetti. Chi desiderasse ricevere informazioni relative a licenze può rivolgersi per iscritto a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Per richieste di licenze relative ad informazioni double-byte (DBCS), contattare il Dipartimento di Proprietà Intellettuale IBM nel proprio paese o inviare richieste per iscritto a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

Il seguente paragrafo non si applica al Regno Unito o a qualunque altro paese in cui tali dichiarazioni sono incompatibili con le norme locali: INTERNATIONAL BUSINESS MACHINES CORPORATION FORNISCE LA PRESENTE PUBBLICAZIONE "NELLO STATO IN CUI SI TROVA" SENZA GARANZIE DI ALCUN TIPO, ESPRESSE O IMPLICITE, IVI INCLUSE, A TITOLO DI ESEMPIO, GARANZIE IMPLICITE DI NON VIOLAZIONE, DI COMMERCIABILITÀ E DI IDONEITÀ PER UNO SCOPO PARTICOLARE. Alcuni stati non consentono la rinuncia a garanzie esplicite o implicite in determinate transazioni; quindi la presente dichiarazione potrebbe non essere applicabile.

Questa pubblicazione potrebbe contenere imprecisioni tecniche o errori tipografici. Le informazioni incluse in questo documento vengono modificate su base periodica; tali modifiche vengono incorporate nelle nuove edizioni della pubblicazione. IBM si riserva il diritto di apportare miglioramenti o modifiche al prodotto/i e/o al programma/i descritti nella pubblicazione in qualsiasi momento e senza preavviso.

Qualsiasi riferimento a siti Web non IBM contenuto nelle presenti informazioni è fornito per consultazione e non vuole in alcun modo promuovere i suddetti siti Web. I materiali presenti in tali siti Web non sono parte dei materiali per questo prodotto IBM e l'utilizzo di tali siti Web è a proprio rischio.

Tutti i commenti e i suggerimenti inviati potranno essere utilizzati liberamente da IBM e diventeranno esclusiva della stessa.

Coloro che detengono la licenza su questo programma e desiderano avere informazioni su di esso allo scopo di consentire (i) uno scambio di informazioni tra programmi indipendenti ed altri (compreso questo) e (ii) l'uso reciproco di tali informazioni, dovrebbero rivolgersi a:

IBM Corporation
Coordinatore interoperabilità software, Dipartimento 49XA
Autostrada 3605 52 N

© Copyright IBM Corp. 2007, 2024

Rochester, MN 55901
U.S.A.

Queste informazioni possono essere rese disponibili secondo condizioni contrattuali appropriate, compreso, in alcuni casi, il pagamento di un addebito.

Il programma su licenza descritto in queste informazioni e tutto il materiale su licenza disponibile per esso sono forniti da IBM in base ai termini dell' IBM Customer Agreement, IBM International Program License Agreement o qualsiasi altro accordo equivalente tra le parti.

Tutti i dati relativi alle prestazioni contenuti in questo documento sono stati determinati in un ambiente controllato. Pertanto, i risultati ottenuti in altri ambienti operativi possono variare in modo significativo. Alcune misurazioni potrebbero essere state fatte su sistemi a livello di sviluppo e non vi è alcuna garanzia che queste misurazioni saranno le stesse sui sistemi generalmente disponibili. Inoltre, alcune misurazioni potrebbero essere state stimate mediante estrapolazione. I risultati quindi possono variare. Gli utenti di questo documento dovrebbero verificare i dati applicabili per il loro ambiente specifico.

Le informazioni relative a prodotti non IBM provengono dai fornitori di tali prodotti, dagli annunci pubblicati o da altre fonti pubblicamente disponibili. IBM non ha verificato tali prodotti e, pertanto, non può garantirne l'accuratezza delle prestazioni. Eventuali commenti relativi alle prestazioni dei prodotti non IBM devono essere indirizzati ai fornitori di tali prodotti.

Tutte le dichiarazioni riguardanti la direzione o l'intento futuro di IBM sono soggette a modifica o ritiro senza preavviso e rappresentano solo scopi e obiettivi.

Questa pubblicazione contiene esempi di dati e prospetti utilizzati quotidianamente nelle operazioni aziendali, Per poterli illustrare nel modo più completo possibile, gli esempi riportano nomi di persone, società, marchi e prodotti. Tutti questi nomi sono fittizi e qualsiasi somiglianza con nomi ed indirizzi adoperati da imprese realmente esistenti sono una mera coincidenza.

LICENZA SUL COPYRIGHT:

Queste informazioni contengono programmi applicativi di esempio in lingua originale, che illustrano le tecniche di programmazione su diverse piattaforme operative. È possibile copiare, modificare e distribuire questi programmi di esempio sotto qualsiasi forma senza alcun pagamento alla IBM, allo scopo di sviluppare, utilizzare, commercializzare o distribuire i programmi applicativi in conformità alle API (application programming interface) a seconda della piattaforma operativa per cui i programmi di esempio sono stati scritti. Questi esempi non sono stati testati approfonditamente tenendo conto di tutte le condizioni possibili. IBM, quindi, non può garantire o sottintendere l'affidabilità, l'utilità o il funzionamento di questi programmi.

Se si sta visualizzando queste informazioni in formato elettronico, le fotografie e le illustrazioni a colori potrebbero non apparire.

Informazioni sull'interfaccia di programmazione

Le informazioni sull'interfaccia di programmazione, se fornite, consentono di creare software applicativo da utilizzare con questo programma.

Questo manuale contiene informazioni sulle interfacce di programmazione che consentono al cliente di scrivere programmi per ottenere i servizi di WebSphere MQ.

Queste informazioni, tuttavia, possono contenere diagnosi, modifica e regolazione delle informazioni. La diagnosi, la modifica e la regolazione delle informazioni vengono fornite per consentire il debug del software applicativo.

Importante: Non utilizzare queste informazioni di diagnosi, modifica e ottimizzazione come interfaccia di programmazione poiché sono soggette a modifica.

Marchi

IBM, il logo IBM , ibm.com, sono marchi di IBM Corporation, registrati in molte giurisdizioni nel mondo. Un elenco aggiornato dei marchi IBM è disponibile sul web in "Copyright and trademark

information"www.ibm.com/legal/copytrade.shtml. Altri nomi di prodotti e servizi potrebbero essere marchi di IBM o altre società.

Microsoft e Windows sono marchi di Microsoft Corporation negli Stati Uniti, in altri paesi o entrambi.

UNIX è un marchio registrato di The Open Group negli Stati Uniti e/o in altri paesi.

Linux è un marchi registrato di Linus Torvalds negli Stati Uniti e/o in altri paesi.

Questo prodotto include il software sviluppato da Eclipse Project (https://www.eclipse.org/).

Java e tutti i marchi e i logo Java sono marchi registrati di Oracle e/o di società affiliate.

Numero parte:

(1P) P/N: