MQSeries

# Utilizzo di Java

SC13-2958-07

MQSeries

# Utilizzo di Java

SC13-2958-07

**Nota**

Prima di utilizzare queste informazioni ed il prodotto che esse supportano, leggere le informazioni contenute nella "Appendice H. Informazioni particolari" a pagina 413.

**Ottava edizione (giugno 2001)**

Questa edizione è valida per IBM MQSeries classi per Java Versione 5.2.0 e MQSeries classi per Java Message Service Versione 5.2 e per tutti i rilasci e le modifiche successivi se non diversamente indicato nelle nuove edizioni.

# Indice

# Figure

# Tabelle

# Informazioni su questo manuale

Questo manuale descrive:

- MQSeries classi per Java, che può essere utilizzato per accedere ai sistemi MQSeries
- MQSeries classi per Java Message Service, che possono utilizzate per accedere sia a JMS (Java Message Service) che alle applicazioni MQSeries

**Note:**

1. Questa documentazione è disponibile solo in formato elettronico (PDF e HTML) come parte del prodotto e dal sito Web della famiglia MQSeries all'indirizzo:

   `http://www.ibm.com/software/mqseries/`

   Questa documentazione **non può** essere ordinata in formato cartaceo.
2. Il file README può essere consultato per informazioni che spiegano e correggono le informazioni in questo manuale. Il file README viene installato con il codice MQ Java e si può trovare nella sottodirectory `doc`.

## Abbreviazioni utilizzate in questo manuale

In questo manuale sono utilizzate le seguenti abbreviazioni:

| | |
|---|---|
| **MQ Java** | Una combinazione delle MQSeries classi per Java e delle MQSeries classi per Java Message Service |
| **MQ base Java** | MQSeries classi per Java |
| **MQ JMS** | MQSeries classi per Java Message Service |

## A chi si rivolge questo manuale

Questo manuale si rivolge ai programmatori che hanno dimestichezza con l'API procedurale MQSeries descritta nel manule *MQSeries Application Programming Guide* e spiega come applicare le informazioni in esso contenute per utilizzare in modo corretto e produttivo le interfacce di programmazione MQ Java.

## Cosa bisogna conoscere per comprendere questo manuale

E' necessario:

- Conoscere il linguaggio di programmazione Java
- Comprendere le finalità di MQSI (Message Queue Interface) descritte nel capitolo relativo a MQI nel manuale *MQSeries Application Programming Guide* e nel capitolo relativo alla descrizione delle chiamate nel manuale *MQSeries Application Programming Reference*
- Avere esperoenza dei programmi MQSeries in generale, oppure conoscere già il contenuto delle altre pubblicazioni MQSeries

Gli utenti che desiderano utilizzare MQ base Java con CICS Transaction Server per OS/390 devono inoltre:

- Conoscere i concetti di CICS (Customer Information Control System)
- Utilizzare CICS API (Application Programming Interface)Java

- Eseguire i programmi Java dall'interno del CICS

Gli utenti che desiderano utilizzare VisualAge per Java per sviluppare applicazioni OS/390 UNIX System Services High Performance Java (HPJ) devono avere dimestichezza con Enterprise Toolkit per OS/390 (fornito con VisualAge per Java Enterprise Edition per OS/390, Versione 2).

## Come utilizzare questo manuale

La prima parte di questo manuale descrive l'utilizzo di MQ base Java e MQ JMS. La seconda parte è destinata ai programmatori che desiderano utilizzare MQ base Java e la terza parte ai programmatori che desiderano utilizzare MQ JMS.

Si consiglia di leggere prima i capitoli della prima parte che contengono un'introduzione a MQ base Java e MQ JMS. Passare quindi alle informazioni relative alla programmazione nella parte 2 o nella parte 3 per comprendere come utilizzare le classi per inviare e ricevere i messaggi MQSeries nell'ambiente che si desidera utilizzare.

In appendice a questo manuale sono disponibili un glossario ed una bibliografia.

Ricordarsi di verificare il file README installato con il codice MQ Java per ulteriori e specifiche informazioni per l'ambiente.

# Riepilogo delle modifiche

Questa sezione descrive le modifiche contenute in questa edizione di *MQSeries - Utilizzo di Java*. Le modifiche apportate dall'ultima edizione di questo manuale sono contrassegnate da righe verticali visualizzate sulla loro sinistra.

## Modifiche a questa edizione (SC13-2958-07)

Questa edizione:

- Include informazioni sul supporto aggiornato per z/OS & OS/390 e iSeries & AS/400.
- Contiene un'appendice rielaborata che descrive il supporto LDAP.

  Consultare l'"Appendice C. Definizione dello schema LDAP per la memorizzazione degli oggetti Java" a pagina 393.
- Contiene una nuova appendice in cui si descrive come eseguire le applet utilizzando MQ Java con Java 1.2 o successiva.

  Consultare l'"Appendice F. Utilizzo di MQ Java nelle applet con Java 1.2 o successiva" a pagina 407.
- Contiene un'appendice separata con informazioni per MA1G SupportPac.

  Consultare l'"Appendice G. Informazioni per SupportPac MA1G" a pagina 409.
- Contiene diverse modifiche per migliorare l'utilizzo e l'accesso al prodotto.

## Modifiche alla settima edizione (SC13-2958-06)

Questa edizione include gli aggiornamenti per le nuove funzioni introdotte da MQ Java V5.2. Queste informazioni comprendono:

- Gli aggiornamenti apportati alle procedure di installazione. Consultare il "Capitolo 2. Procedure di installazione" a pagina 9.
- Il supporto fornito per i pool di informazioni, che consentono di migliorare le prestazioni delle applicazioni e del middleware che utilizzano connessioni multiple per i gestori code MQSeries. Consultare:
  - "Pool di connessioni" a pagina 71
  - "MQEnvironment" a pagina 97
  - "MQPoolServices" a pagina 138
  - "MQPoolServicesEvent" a pagina 139
  - "MQPoolToken" a pagina 141
  - "MQQueueManager" a pagina 158
  - "MQSimpleConnectionManager" a pagina 169
  - "MQConnectionManager" a pagina 173
  - "MQPoolServicesEventListener" a pagina 172
  - "ManagedConnection" a pagina 180
  - "ManagedConnectionFactory" a pagina 183
  - "ManagedConnectionMetaData" a pagina 185
- Le nuove opzioni di configurazione introdotte per le code del sottoscrittore, che consentiranno di creare code multiple e code condivise per le applicazioni publish/subscribe. Consultare:
  - "Proprietà" a pagina 44

  - "Configurazione della coda del sottoscrittore di base" a pagina 208
  - "Topic" a pagina 353
  - "TopicConnectionFactory" a pagina 358
- Il nuovo programma di utilità per la rimozione dei dati per i sottoscrittori, che consentirà di evitare gli eventuali problemi causati da una chiusura non programmata di oggetti del sottoscrittore. Consultare l'"Programma di utilità per la pulizia dei sottoscrittori" a pagina 211.
- Il supporto per le funzioni del server applicazioni, che consente un'elaborazione simultanea dei messaggi. Consultare:
  - "Capitolo 13. Application Server Facilities MQ JMS" a pagina 233
  - "ConnectionConsumer" a pagina 270
  - "QueueConnection" a pagina 322
  - "Session" a pagina 336
  - "TopicConnection" a pagina 355
- Gli aggiornamenti apportati alle informazioni di configurazione del server LDAP. Consultare l'"Appendice C. Definizione dello schema LDAP per la memorizzazione degli oggetti Java" a pagina 393.
- Il supporto per le transazioni distribuite utilizzando il protocollo X/Open XA. Il programma di servizio MQ JMS comprende infatti delle classi XA che gli consentono di partecipare ad un commit in due fasi coordinato da un appropriato programma di gestione delle transazioni. Consultare:
  - "Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401
  - "XAConnection" a pagina 374
  - "XAConnectionFactory" a pagina 375
  - "XAQueueConnection" a pagina 376
  - "XAQueueConnectionFactory" a pagina 377
  - "XAQueueSession" a pagina 379
  - "XASession" a pagina 380
  - "XATopicConnection" a pagina 382
  - "XATopicConnectionFactory" a pagina 384
  - "XATopicSession" a pagina 386

## Modifiche alla sesta edizione (SC13-2958-05)

Questa edizione contiene le informazioni relative all'integrazione del supporto per Linux.

# Parte 1. Istruzioni per gli utenti

# Capitolo 1. Introduzione

Questocapitolo offre una panoramica delle MQSeries classi per Java e delle MQSeries classi per Java Message Service e delle loro possibili applicazioni.

## Descrizione delle MQSeries classi per Java

Grazie alle MQSeries classi per Java (MQ base Java), un programma scritto nel linguaggio di programmazione Java può:

- Stabilire una connessione con MQSeries come un client MQSeries
- Stabilire una connessione diretta con un server MQSeries

Queste Java abilitano le applet, le applicazioni e i servlet ad eseguire chiamate e query per MQSeries. Questo consente di accedere ad applicazioni già esistenti e ad applicazioni mainframe, di norma tramite Internet, senza dover necessariamente disporre di altro codice MQSeries sulla macchina client. Grazie a MQ base Java, l'utente di un terminale connesso via Internet può partecipare attivamente alle transazioni e non è più limitato ad essere semplicemente un fornitore o un fruitore di informazioni.

## Descrizione di MQSeries classi per Java Message Service

MQSeries classi per Java Message Service (MQ JMS) è un gruppo di classi Java che implementa le interfacce Java Message Service (JMS) per abilitare i programmi JMS ad accedere ai sistemi MQSeries. Sono supportati sia il modello point-to-point che quello publish-subscribe di JMS.

L'utilizzo di MQ JMS come API per scrivere le applicazioni MQSeries presenta diversi vantaggi. Alcuni di questi vantaggi sono dovuti al fatto che JMS è uno standard aperto ad implementazione multipla. Altri vantaggi sono invece dovuti alle funzioni aggiuntive disponibili in MQ JMS, non disponibili invece in MQ base Java.

Utilizzare uno standard aperto consente infatti, tra l'altro, di:
- Proteggere gli investimenti fatti sia in codice applicativo che in competenze
- Disporre di tecnici esperti nella programmazione di applicazioni JMS
- Poter integrare varie implementazioni JMS per rispondere a requisiti di diversa natura

Per ulteriori informazioni sui vantaggi offerti da JMS API sono disponibili nel sito Web della Sun all'indirizzo `http://java.sun.com`.

Le ulteriori funzioni fornite con MQ base Java comprendono:
- Recapito asincrono dei messaggi
- Selettori di messaggi
- Supporto del sistema di messaggistica di tipo publish/subscribe
- Classi di messaggio strutturate

## Possibili utenti di MQ Java

Utilizzare MQSeries classi per Java e MQSeries classi per Java Message Service può rivelarsi molto vantaggioso per:

### Possibili utenti di MQ Java

- Medie e grandi imprese che stanno introducendo delle soluzioni client/server basate sull'utilizzo di Intranet. In questo caso, la tecnologia Internet consente di fornire un accesso facile ed economico ad un sistema di comunicazione globale e la connettività di MQSeries fornisce trasmissioni dei dati sicure e protette.
- Medie e grandi imprese che desiderano implementare un affidabile sistema per le comunicazioni interaziendali con le proprie imprese partner. Anche in questo caso, la tecnologia Internet consente di fornire un accesso facile ed economico ad un sistema di comunicazione globale e la connettività di MQSeries fornisce trasmissioni dei dati sicure e protette.
- Medie e grandi imprese che desiderano fornire un accesso via Internet ad alcune delle proprie applicazioni aziendali. In questo caso, Internet fornisce un accesso globale ed economico e la connettività MQSeries fornisce un alto fattore di protezione dei dati grazie all'utilizzo delle code. Offrendo una disponibilità ininterrotta, tempi di risposta brevi e maggiore precisione, un'impresa può offrire un servizio di elevata qualità alla propria clientela ad un costo contenuto.
- Un Internet Service Provider oppure un altro Provider VAN (Value Added Network). Queste imprese possono trarre vantaggio dal sistema per le comunicazioni semplice ed economico offerto da Internet. Esse possono inoltre proporre un servizio a valore aggiunto offrendo l'elevato fattore di protezione dei dati fornito dalla connettività MQSeries. Un Internet Service Provider che utilizza MQSeries può ricevere un'immediata conferma della ricezione di dati di input da un browser Web, garantire l'affidabilità delle trasmissioni di dati e fornire agli utenti del browser Web un modo semplice di controllare lo stato dei messaggi.

MQSeries e MQSeries classi per Java Message Service offrono un'eccellente infrastruttura per accedere alle applicazioni aziendali e per sviluppare complesse applicazioni Web. Una richiesta di servizio da un browser Web può essere memorizzata in una coda per essere elaborata appena possibile, consentendo così di inviare una risposta tempestiva all'utente finale indipendentemente dagli effettivi tempi di caricamento del sistema. Grazie all'utilizzo di questo tipo di code, in termini di rete 'chiuso' all'utente, i tempi di caricamento sulla rete non influenzano la tempestività della risposta. Inoltre, la natura transazionale del sistema di messaggistica di MQSeries consente di espandere in modo sicuro una semplice richiesta dal browser in una sequenza di singoli processi di back-end in modo transazionale.

MQSeries classi per Java abilita inoltre gli sviluppatori di applicazioni a sfruttare le notevoli capacità del linguaggio di programmazione Java per creare applet ed applicazioni che possono essere eseguite su tutte le piattaforme che supportano l'ambiente di runtime Java. Questo consente di ridurre notevolmente i tempi necessari per sviluppare delle applicazioni MQSeries multipiattaforma. Inoltre, eventuali aggiornamenti apportati successivamente alle applet vengono automaticamente integrati nei sistemi degli utenti finali quando viene scaricato il codice applet.

## Opzioni di connessione

Le opzioni programmabili consentono a MQ Java di stabilire connessione con MQSeries in uno dei seguenti modi:

- Come un client MQSeries utilizza TCP/IP (Transmission Control Protocol/Internet Protocol)
- In modalità binding, stabilendo una connessione diretta con MQSeries

## Connessioni

MQ base Java su Windows NT può anche stabilire una connessione utilizzando VisiBroker per Java. La Tabella 1 a pagina 6 contiene le modalità di connessione che è possibile utilizzare sulle varie piattaforme.

*Tabella 1. Piattaforme e modalità di connessione*

| Piattaforma server | Client standard | Client VisiBroker | Bind |
|---|---|---|---|
| Windows NT | si | si | si |
| Windows 2000 | si | no | si |
| AIX | si | no | si |
| Sun OS (v4.1.4 e precedenti) | si | no | no |
| Sun Solaris (v2.6, v2.8, V7 o SunOS v5.6, v5.7) | si | no | si |
| OS/2 | si | no | si |
| OS/400 | si | no | si |
| HP-UX | si | no | si |
| AT&T GIS UNIX | si | no | no |
| SINIX e DC/OSx | si | no | no |
| OS/390 | no | no | si |
| Linux | si | no | no |

**Note:**

1. Il supporto binding HP-UX Java è disponibile soltanto per i sistemi HP-UXv11 su cui è in esecuzione la versione POSIX draft 10 pthreaded di MQSeries. E' richiesto anche il HP-UX Developer's Kit per Java 1.1.7 (JDK), Rilascio C.01.17.01 o successivo.
2. Su HP-UXv10.20, Linux, Windows 95 e Windows 98, è supportata solo la connettività client TCP/IP.

Nelle sezioni che seguono verranno descritte queste opzioni in modo più dettagliato.

## Connessione client

Per utilizzare MQ Java come client MQSeries, è possibile installarlo nella macchina del server MQSeries, che può contenere anche un server Web o in una macchina separata. Se si installa MQ Java nello stesso computer del server Web, uno dei vantaggi è che è possibile scaricare ed eseguire le applicazioni client MQSeries in computer in cui MQ Java non è installato localmente.

Ogni volta che si sceglie di installare il client, è possibile eseguirlo in tre diverse modalità:

**Dall'interno di qualsiasi browser Web Java abilitato**
In questa modalità, le ubicazioni dei gestori code MQSeries a cui è possibile accedere potrebbero essere limitate dalle restrizioni di sicurezza del browser utilizzato.

**Utilizzo di un appletviewer**
Per utilizzare questo metodo, è necessario aver installato il JDK (Java Development Kit) o il JRE (Java Runtime Environment) sulla macchina del client.

**Come programma standalone Java o in un server applicazioni Web**
Per utilizzare questo metodo, è necessario aver installato il JDK (Java Development Kit) o il JRE (Java Runtime Environment) sulla macchina del client.

## Utilizzo VisiBroker per Java

Sulla piattaforma Windows, la connessione mediante VisiBroker viene fornita come alternativa all'utilizzo dei protocolli del client MQSeries standard. Questo supporto viene fornito da VisiBroker per Java insieme a Netscape Navigator, e richiede VisiBroker per Java e un server oggetti MQSeries nel computer server MQSeries. Un apposito server oggetti viene fornito con MQ base Java.

## Connessione binding

Quando viene utilizzato in modalità binding, MQ Java utilizza l'interfaccia JNI (Java Native Interface) per eseguire chiamate direttamente nell'API del gestore code esistente anziché comunicare attraverso una rete. in questo modo le prestazioni per le applicazioni MQSeries che utilizzano le connessioni di rete risultano sicuramente migliori. A differenza della modalità client, le applicazioni scritte utilizzando la modalità binding non possono essere scaricate come applet.

Per utilizzare la connessione binding, è necessario che MQ Java sia installato nel server MQSeries.

# Prerequisiti

Per eseguire MQ base Java, è necessario disporre del seguente software:

- MQSeries per la piattaforma server che si desidera utilizzare.
- Java Development Kit per la piattaforma server.
- Java Development Kit o JRE (Java Runtime Environment), o un browser Web che abilita Java per piattaforme client. (Vedere "Connessione client" a pagina 6.)
- VisiBroker per Java (solo se in esecuzione su Windows con una connessione VisiBroker).
- Per z/OS & OS/390, OS/390 Versione 2 Release 9 o superiore, or z/OS, con USS(UNIX System Services).
- Per OS/400, l'AS/400 Developer Kit for Java, 5769-JV1 e il Qshell Interpreter, OS/400 (5769-SS1) Option 30.

Per utilizzare lo strumento di amministrazione MQ JMS (vedere "Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS" a pagina 35), è necessario disporre del seguente software aggiuntivo:

- Almeno uno dei seguenti pacchetti dei provider di servizi:
  - Lightweight Directory Access Protocol (LDAP) - `ldap.jar`, `providerutil.jar`.
  - File system - `fscontext.jar`, `providerutil.jar`.
- Un provider di servizi JNDI (Java Naming and Directory Service). Questa è la risorsa che memorizza la rappresentazione fisica degli oggetti amministrati. Gli utenti di MQ JMS probabilmente utilizzeranno un server LDAP a questo scopo, ma lo strumento supporta anche l'utilizzo di un provider di servizi del contesto del file system. Se viene utilizzato un server LDAP, è necessario configurarlo per la memorizzazione degli oggetti JMS. Per informazioni su come effettuare questa configurazione, consultare "Appendice C. Definizione dello schema LDAP per la memorizzazione degli oggetti Java" a pagina 393.

Per utilizzare le funzioni XOpen/XA di MQ JMS, è necessario disporre di MQSeries V5.2.

# Capitolo 2. Procedure di installazione

Questo capitolo descrive come installare MQSeries classi per Java e MQSeries classi per Java Message Service.

## Recupero di MQSeries classi per Java e di MQSeries classi per Java Message Service

Questo prodotto è disponibile per le piattaforme AIX, iSeries & AS/400, HP-UX, Linux, Sun Solaris, z/OS & OS/390, eWindows. Contiene:

- MQSeries classi per Java (MQ base Java) Versione 5.2.0
- MQSeries classi per Java Message Service (MQ JMS) Versione 5.2

per informazioni sulla connettività disponibile su ogni piattaforma specifica, fare riferimento alla sezione "Opzioni di connessione" a pagina 4.

Il prodotto viene fornito sotto forma di file compressi disponibili presso il sito Web MQSeries, `http://www.ibm.com/software/mqseries/`. I file vengono forniti come parte di SupportPac MA88. Seguire i collegamenti per effettuare il "download", poi, "SupportPacs" per trovare il codice MQ Java.

**Nota:** Prima di installare o di utilizzare MQ Java, ricordarsi di consultare il file README che si trova nella sottodirectory `doc` con il codice installato. Questa directory può contenere informazioni che potrebbero correggere ed integrare questo manuale.

## Installazione di MQSeries classi per Java e MQSeries classi per Java Message Service

Per le versioni più aggiornate delle sole classi di MQ base Java, è possibile installare solo MQ base Java Versione 5.2.0. Per utilizzare le applicazioni MQ JMS, è necessario installare sia MQ base Java che MQ JMS (conosciuti insieme come MQ Java).

MQ base Java è contenuto nei seguenti file Java `.jar`:

| | |
|---|---|
| **com.ibm.mq.jar** | Questo codice include il supporto per tutte le opzioni di connessione. |
| **com.ibm.mq.iiop.jar** | Questo codice supporta solo la connessione VisiBroker. Viene fornito solo sulla piattaforma Windows. |
| **com.ibm.mqbind.jar** | Questo codice supporta solo la connessione binding e non viene fornito o supportato su tutte le piattaforme. Si consiglia di non utilizzarlo in tutte le applicazioni nuove. |

MQ JMS è contenuto nel seguente file Java `.jar`:

**com.ibm.mqjms.jar**

### Installazione di MQ base Java e MQ JMS

Le seguenti librerie Java Sun Microsystems sono ridistribuite con il prodotto MQ JMS:

| | |
|---|---|
| **connector.jar** | Versione 1.0 Public Draft |
| **fscontext.jar** | Early Access 4 Release |
| **jms.jar** | Versione 1.0.2 |
| **jndi.jar** | Versione 1.1.2 (ad eccezione di z/OS & OS/390) |
| **ldap.jar** | Versione 1.0.3 (ad eccezione di z/OS & OS/390) |
| **providerutil.jar** | Versione 1.0 |

Per istruzioni sull'installazione, consultare la sezione relativa alla piattaforma richiesta:

| | |
|---|---|
| **AIX, HP-UX e Sun Solaris** | "Installazione in UNIX" |
| **z/OS e OS/390** | "Installazione su z/OS & OS/390" a pagina 11 |
| **iSeries & AS/400** | "Installazione su iSeries & AS/400" a pagina 11 |
| **Linux** | "Installazione in Linux" a pagina 12 |
| **Windows** | "Installazione in Windows" a pagina 12 |

Quando l'installazione è completa, i file e gli esempi risultano installati nei percorsi indicati nella sezione "Directory di installazione" a pagina 13.

Dopo l'installazione è necessario aggiornare le variabili di ambiente, come viene indicato nella sezione "Variabili di ambiente" a pagina 13.

**Nota:** Effettuare l'installazione del prodotto e la successiva installazione o reinstallazione della versione di MQSeries di base con cautela. Assicurarsi di non aver installato la versione 5.1 di MQ base Java, poiché il supporto MQSeries Java riporta indietro di un livello.

## Installazione in UNIX

In questa sezione verrà descritto come installare MQ Java in AIX, HP-UX e Sun Solaris. per informazioni sull'installazione di MQ base Java in Linux, consultare la sezione "Installazione in Linux" a pagina 12.

**Nota:** Se questa è un'installazione solo client (ossia, un server MQSeries *non* è stato installato), è necessario impostare il gruppo e l'ID utente mqm. Per ulteriori informazioni, consultare il manuale di informazioni preliminari MQSeries relativo alla propria piattaforma.

1. Accedere come utente root.
2. Copiare il file `ma88_xxx.tar.Z` in formato binario e memorizzarlo nella directory /tmp, dove *xxx* è l'identificativo della piattaforma appropriato:
   - `aix` AIX
   - `hp10` HP-UXv10
   - `hp11` HP-UXv11
   - `sol` Sun Solaris
3. Immettere i seguenti comandi (dove *xxx* è l'identificativo della piattaforma appropriato):

```
uncompress -fv /tmp/ma88_xxx.tar.Z
tar -xvf /tmp/ma88_xxx.tar
rm /tmp/ma88_xxx.tar
```

Questi comandi creano i file e le directory necessarie.

4. Utilizzare lo strumento di installazione appropriato per ciascuna piattaforma:
   - Per AIX, utilizzare `smitty` e:
     a. Disinstallare tutti i componenti che iniziano con mqm.java.
     b. Installare i componenti dalla directory /tmp.
   - Per HP-UX, utilizzare `sam` ed eseguire l'installazione dal file `ma88_hp10` o `ma88_hp11`, nel modo più appropriato.

     **Nota:** Java non supporta la code page 1051 (che è invece predefinita per HP-UX). Per eseguire il broker Publish/Subscribe in HP-UX, potrebbe essere necessario modificare il CCSID del gestore code del broker su un valore alternativo, ad esempio 819.
   - Per Sun Solaris, immettere il seguente comando e selezionare le opzioni richieste:

     ```
     pkgadd -d /tmp mqjava
     ```

     Quindi immettere il seguente comando:

     ```
     rm -R /tmp/mqjava
     ```

## Installazione su z/OS & OS/390

Questa sezione descrive come installare MQ base Java su z/OS & OS/390.

1. Selezionare una directory di installazione per il prodotto in una USS HFS (per esempio, `/usr/lpp`). Se tale directory non è nella home directory, potrebbe essere necessaria l'autorità appropriata.
2. Copiare il file `ma88_zos.tar.Z` nella home directory in USS.
3. Passare alla directory di installazione selezionata; per esempio: `cd /usr/lpp`
4. Immettere il seguente comando:

   ```
   tar -xozf ~/ma88_zos.tar.Z
   ```

   In questo modo viverrà creata e riempita una directory chiamata `mqm` nella directory corrente.

## Installazione su iSeries & AS/400

In questa sezione verrà descritto come installare MQ Java in AS/400.

1. Copiare il file `ma88_iSeries.zip` in una directory del PC.
2. Decomprimere il file utilizzando la funzione Unzip di InfoZip.

   In questo modo verrà creato il file `ma88_iSeries.savf`.
3. Creare un file di salvataggio chiamato MA88 in una libreria adatta su iSeries o su AS/400; per esempio, nella libreria QGPL:

   ```
   CRTSAVF FILE(QGPL/MA88)
   ```
4. Trasferire `ma88_iSeries.savf` in questo file di salvataggio come immagine binaria. Se per eseguire questa operazione si utilizza l'FTP, il comando put dovrebbe essere simile a quanto segue:

   ```
   PUT C:\TEMP\MA88_ISERIES.SAVF QGPL/MA88
   ```
5. Installare MQSeries classi per Java, numero prodotto 5648C60, utilizzando RSTLICPGM:

```
RSTLICPGM LICPGM(5648C60) DEV(*SAVF) SAVF(QGPL/MA88)
```

6. Eliminare il file di salvataggio creato nel passaggio 3 a pagina 11:
   ```
   DLTF FILE(QGPL/MA88)
   ```

## Installazione in Linux

In questa sezione verrà illustrato come installare MQ Java su Linux.

Per Linux, sono disponibili due file di installazione, `ma88_linux.tgz` e `MQSeriesJava-5.2.0-1.noarch.rpm`. Entrambi forniscono un'installazione identica.

Se si dispone dell'accesso utente root al sistema di destinazione o si utilizza un database RPM (Red Hat Package Manager) per installare i pacchetti, utilizzare `MQSeriesJava-5.2.0-1.noarch.rpm`.

Se non si dispone dell'accesso da utente root al sistema di destinazione, oppure nel sistema di destinazione non è installato il RPM, utilizzare `ma88_linux.tgz`.

Per installare utilizzando `ma88_linux.tgz`:

1. Selezionare una directory di installazione per il prodotto (ad esempio, /opt).

   Se questa directory non è nella home directory, potrebbe essere necessario accedere come utente root.
2. Copiare il file `ma88_linux.tgz` nella home directory.
3. Passare alla directory di installazione selezionata, ad esempio:
   ```
   cd /opt
   ```
4. Immettere il seguente comando:
   ```
   tar -xpzf ~/ma88_linux.tgz
   ```

   In questo modo verrà creata e riempita una directory chiamata mqm nella directory corrente, ad esempio /opt.

Per installare utilizzando `MQSeriesJava-5.2.0-1.noarch.rpm`:

1. Accedere come utente root.
2. Copiare `MQSeriesJava-5.2.0-1.noarch.rpm` in una directory di lavoro.
3. Immettere il seguente comando:
   ```
   rpm -i MQSeriesJava-5.2.0-1.noarch.rpm
   ```

   Il prodotto verrà installato in /opt/mqm/. E' possibile anche installarlo in un percorso differente. Per ulteriori dettagli, consultare la documentazione RPM.

## Installazione in Windows

In questa sezione verrà descritta l'installazione di MQ Java in Windows.

1. Creare una directory vuota chiamata tmp e renderla la directory corrente.
2. Copiare il file `ma88_win.zip` in questa directory.
3. Decomprimere `ma88_win.zip` utilizzando la funzione Unzip di InfoZip.
4. Eseguire `setup.exe` da questa directory e seguire le istruzioni riportate nelle finestra successive.

   **Nota:** Per installare solo MQ base Java, selezionare le relative opzioni in questa fase.

## Directory di installazione

I file di MQ Java V5.2 vengono installati nelle directory riportate nella Tabella 2.

*Tabella 2. Directory di installazione del prodotto*

| Piattaforma | Directory |
|---|---|
| AIX | usr/mqm/java/ |
| z/OS & OS/390 | *install_dir*/mqm/java/ |
| iSeries & AS/400 | /QIBM/ProdData/mqm/java/ |
| HP-UX e Sun Solaris | opt/mqm/java/ |
| Linux | *install_dir*/mqm/java/ |
| Windows 95, 98, 2000 e NT | *install_dir*\ |
| **Nota:** *install_dir* è la directory in cui è stato installato il prodotto. Su Linux, probabilmente è /opt e su z/OS & OS/390 probabilmente è /usr/lpp. | |

## Variabili di ambiente

Al termine dell'installazione, è necessario aggiornare la variabile di ambiente CLASSPATH per includere il codice MQ base Java e le directory di esempio. Nella Tabella 3 sono riportate le impostazioni tipiche di CLASSPATH per le varie piattaforme.

*Tabella 3. Istruzioni CLASSPATH di esempio per il prodotto*

| Piattaforma | CLASSPATH di esempio |
|---|---|
| AIX | CLASSPATH=/usr/mqm/java/lib/com.ibm.mq.jar:<br>/usr/mqm/java/lib/connector.jar:<br>/usr/mqm/java/lib:<br>/usr/mqm/java/samples/base: |
| HP-UX e Sun Solaris | CLASSPATH=/opt/mqm/java/lib/com.ibm.mq.jar:<br>/opt/mqm/java/lib/connector.jar:<br>/opt/mqm/java/lib:<br>/opt/mqm/java/samples/base: |
| Windows 95, 98, 2000 e NT | CLASSPATH=*install_dir*\lib\com.ibm.mq.jar;<br>*install_dir*\lib\com.ibm.mq.iiop.jar;<br>*install_dir*\lib\connector.jar;<br>*install_dir*\lib\;<br>*install_dir*\samples\base\; |
| z/OS & OS/390 | CLASSPATH=*install_dir*/mqm/java/lib/com.ibm.mq.jar:<br>*install_dir*/mqm/java/lib/connector.jar:<br>*install_dir*/mqm/java/lib:<br>*install_dir*/mqm/java/samples/base: |
| iSeries & AS/400 | CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar:<br>/QIBM/ProdData/mqm/java/lib/connector.jar:<br>/QIBM/ProdData/mqm/java/lib:<br>/QIBM/ProdData/mqm/java/samples/base: |
| Linux | CLASSPATH=*install_dir*/mqm/java/lib/com.ibm.mq.jar:<br>*install_dir*/mqm/java/lib/connector.jar:<br>*install_dir*/mqm/java/lib:<br>*install_dir*/mqm/java/samples/base: |
| **Note:**<br>1. *jdk_dir* è la directory in cui è installato il JDK<br>2. *install_dir* è la directory in cui è stato installato il prodotto | |

### directory di installazione

Per utilizzare MQ JMS, è necessario includere file jar aggiuntivi nel percorso di ricerca delle classi. Tali file sono indicati nella sezione “Configurazione post-installazione” a pagina 23.

Se esistono delle applicazioni con una dipendenza sul pacchetto binding considerato obsoleto `com.ibm.mqbind`, è necessario anche aggiungere il file `com.ibm.mqbind.jar` al percorso di ricerca delle classi.

Occorre anche aggiornare ulteriori variabili di ambiente su alcune piattaforme, come viene illustrato nella Tabella 4.

*Tabella 4. Variabili di ambiente per il prodotto*

| **Piattaforma** | **Variabile di ambiente** |
|---|---|
| AIX | LD_LIBRARY_PATH=/usr/mqm/java/lib |
| HP_UX | SHLIB_PATH=/opt/mqm/java/lib |
| Sun Solaris | LD_LIBRARY_PATH=/opt/mqm/java/lib |
| Windows 95, 98, 2000 e NT | PATH=*install_dir*\lib |
| z/OS & OS/390 | LIBPATH=*install_dir*/mqm/java/lib |
| **Nota:** *install_dir* è la directory di installazione del prodotto | |

**Note:**

1. Per utilizzare MQSeries Bindings for Java su OS/400, assicurarsi che la libreria QMQMJAVA sia contenuta nell’elenco delle librerie.
2. Assicurarsi di aver aggiunto le variabili MQSeries e di non aver sovrascritto nessuna variabile di ambiente del sistema esistenti. Una sovrascrittura potrebbe causare infatti un errore dell’applicazione durante la compilazione o in fase di runtime.

## Configurazione del server Web

Se si installa MQSeries Java su un server Web, è possibile effettuare il download ed eseguire le applicazioni MQSeries Java su macchine in cui MQSeries Java non è stato installato localmente. Per rendere i file MQSeries Java accessibili nel server Web, è necessario impostare la configurazione del server Web in modo che punti alla directory in cui è installato il client. Consultare la documentazione del server Web per ulteriori dettagli su come eseguire questa configurazione.

**Nota:** Su z/OS & OS/390, le classi installate non supportano la connessione del client e non possono essere scaricate in modo utili sui client. Tuttavia, i file jar di un’altra piattaforma possono essere trasferiti su z/OS o OS/390 e serviti ai client.

# Capitolo 3. Utilizzo delle MQSeries classi per Java (MQ base Java)

In questo capitolo verrà illustrato:

- Come configurare il sistema per l'esecuzione dell'applet di esempio e dei programmi per la verifica dell'installazione di MQ base Java.
- Come modificare le procedure di esecuzione dei propri programmi.

Ricordarsi di verificare il file README installato con il codice MQ Java per ulteriori e specifiche informazioni per l'ambiente.

Le procedure dipendono dall'opzione di connessione che si desidera utilizzare. Seguire le istruzioni riportate nella sezione che riguarda i propri requisiti.

## Utilizzo dell'applet di esempio per la verifica del client TCP/IP

MQ base Java include un'applet di verifica dell'installazione, `mqjavac.html`. E' possibile utilizzare l'applet per verificare la modalità client connesso TCP/IP di MQ base Java ad eccezione della piattaforma z/OS & OS/390. Le applet non sono supportate sulla piattaforma z/OS & OS/390, cosicché è necessario utilizzare l'applicazione di esempio invece che la verifica. Per le relative istruzioni,consultare "Verifica con l'applicazione di esempio" a pagina 18.

In aggiunta, le impostazioni di sicurezza standard per le applet in Java 1.2 e superiore richiedono che tutte le classi a cui si fa riferimento siano caricate dalla stessa posizione come le applet che si desiderano eseguire. Per informazioni relative alle modalità per ottenere le applet che utilizzano MQ Java per lavorare, consultare "Appendice F. Utilizzo di MQ Java nelle applet con Java 1.2 o successiva" a pagina 407.

L'applet si connette a un determinato gestore code, esercita tutte le chiamate di MQSeries e produce messaggi diagnostici in presenza di eventuali errori.

E' possibile eseguire l'applet dall'appletviewer fornito con JDK. L'appletviewer è in grado di accedere ad un gestore code su qualsiasi host.

In tutti i casi, se l'esecuzione dell'applet non viene completata correttamente, seguire le indicazioni riportate nei messaggi diagnostici e ripetere l'esecuzione.

### Utilizzo dell'applet di esempio su iSeries o AS/400

Il sistema operativo OS/400 non dispone di una GUI (Graphical User Interface) nativa. Per eseguire l'applet di esempio, è necessario utilizzare il Remote Abstract Window Toolkit for Java (AWT) o il Class Broker for Java (CBJ), su hardware in grado di supportare la grafica. E' anche possibile verificare il client dalla riga comandi (vedere "Verifica con l'applicazione di esempio" a pagina 18).

### Configurazione del gestore code per l'accettazione di connessioni client

Utilizzare le procedure descritte di seguito per configurare il gestore code in modo che accetti le richieste di connessioni in arrivo dai client.

## Verifica della modalità client

### Client TCP/IP

1. Definire un canale di connessione server mediante le seguenti procedure:

   **Per la piattaforma iSeries o AS/400:**

   a. Avviare il gestore code utilizzando il comando STRMQM.
   b. Definire un canale di esempio denominato JAVA.CHANNEL digitando il seguente comando:

      ```
      CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
      MCAUSERID(SOMEUSERID) TEXT('Canale di esempio per MQSeries Client per Java')
      ```

      dove `QMGRNAME` è il nome del gestore code e `SOMEUSERID` è un ID utente iSeries o AS/400 con l'autorizzazione appropriata alle risorse MQSeries.

   **Per piattaforme z/OS o OS/390:**

   **Nota:** E' necessario avere la funzione allegato Client installato sul gestore code di destinazione per connettersi via TCP/IP.

   a. Avviare il gestore code utilizzando il comando AVVIA QMGR.
   b. Definire un canale di esempio denominato JAVA.CHANNEL digitando il seguente comando:

      ```
      DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP)
      DESCR('Canale di esempio per MQSeries Client per Java')
      ```

   **Per altre piattaforme:**

   a. Avviare il gestore code utilizzando il comando strmqm.
   b. Digitare il seguente comando per avviare il programma runmqsc:

      ```
      runmqsc
      ```

   c. Definire un canale di esempio denominato JAVA.CHANNEL digitando il seguente comando:

      ```
      DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Canale di esempio per MQSeries Client per Java')
      ```

2. Avviare un programma listener con i seguenti comandi:

   **Per i sistemi operativi OS/2 e NT:**

   Eseguire il comando:

   ```
   runmqlsr -t tcp [-m QMNAME] -p 1414
   ```

   **Nota:** Se si utilizza il gestore code predefinito, è possibile omettere l'opzione -m.

   **Utilizzo di VisiBroker per Java sul sistema operativo Windows NT:**

   Avviare il server IIOP (Internet Inter-ORB Protocol) con il seguente comando:

   ```
   java com.ibm.mq.iiop.Server
   ```

   **Nota:** Per arrestare il server IIOP, eseguire il seguente comando:

   ```
   java com.ibm.mq.iiop.samples.AdministrationApplet shutdown
   ```

   **Per i sistemi operativi UNIX:**

   Configurare il daemon inetd, in modo che inetd avvii i canali di MQSeries. Consultare il manuale *MQSeries Clients* per istruzioni specifiche sull'esecuzione di questa operazione.

   **Per il sistema operativo OS/400:**

   Eseguire il comando:

```
STRMQMLSR MQMNAME(QMGRNAME)
```

dove QMGRNAME è il nome del gestore code.

**Per il sistema operativo z/OS o OS/390:**

a. Assicurarsi che l'iniziatore del canale sia avviato. Se non lo è, avviarlo eseguendo il comando AVVIA CHINIT.
b. Avviare il listener eseguendo il comando AVVIA LISTENER TRPTYPE(TCP) PORT(1414)

## Esecuzione da appletviewer

Per utilizzare questo metodo, è necessario che nel proprio computer sia installato il JDK (Java Development Kit).

**Procedura di installazione locale**

1. Passare alla directory degli esempi relativa alla propria lingua.
2. Digitare:

```
appletviewer mqjavac.html
```

**Procedura di installazione nel server Web:**

Immettere il comando:

```
appletviewer http://Web.server.host/MQJavaclient/mqjavac.html
```

**Note:**

1. Su alcune piattaforme il comando è "applet" e non "appletviewer".
2. Su alcune piattaforme può essere necessario selezionare Proprietà dal menu Applet nella parte superiore sinistra dello schermo e quindi impostare Accesso alla rete su "Senza limitazioni".

Utilizzando questa tecnica dovrebbe essere possibile connettersi a qualsiasi gestore code in esecuzione su qualsiasi host di cui si dispone dell'accesso TCP/IP.

## Personalizzazione dell'applet di verifica

Il file mqjavac.html include alcuni parametri facoltativi che consentono di modificare l'applet per fare in modo che risponda nel migliore dei modi alle proprie esigenze specifiche. Ciascun parametro viene definito in una riga di HTML, simile a quanto segue:

```
<!PARAM name="xxx" value="yyy">
```

Per specificare il valore di un parametro, rimuovere il punto esclamativo iniziale e modificare nel modo desiderato il valore. E' possibile specificare i seguenti parametri:

**nomehost** Il valore da visualizzare inizialmente nella casella di modifica del nome host.

**porta** Il valore da visualizzare inizialmente nella casella di modifica della porta.

**canale** Il valore da visualizzare inizialmente nella casella di modifica del canale.

**queueManager** Il valore da visualizzare inizialmente nella casella di modifica del gestore code.

**userID** Utilizza l'ID utente specificato durante la connessione al gestore code.

**password** Utilizza la password specificata durante la connessione al gestore code.

**trace** Determina la scrittura da parte di MQ base Java di un file di registrazione dell'analisi. Utilizzare questa opzione solo in base alle indicazioni del servizio IBM.

## Verifica con l'applicazione di esempio

Un programma di verifica dell'installazione, MQIVP, viene fornito con MQ base Java. Tale applicazione può essere utilizzata per verificare tutte le modalità di connessione di MQ base Java. Il programma richiede una serie di opzioni e altri dati per determinare la modalità di connessione che si desidera verificare. Attenersi alla seguente procedura per verificare la propria installazione:

1. Per verificare una connessione di esempio:
   a. Configurare il proprio gestore code, in base alle istruzioni riportate nella sezione "Configurazione del gestore code per l'accettazione di connessioni client" a pagina 15.
   b. Completare la procedura sul computer client.

   Per verificare una connessione binding, completare la procedura sul computer server di MQSeries.
2. Passare alla directory degli esempi.
3. Digitare:
   ```
   java MQIVP
   ```

   Il programma cercherà di:
   a. Connettersi, e disconnettersi, dal gestore code in questione.
   b. Aprire, inserire, ottenere e chiudere la coda locale predefinita del sistema.
   c. Restituire un messaggio se le operazioni vengono eseguite correttamente.
4. Nella richiesta comandi (1), lasciare l'impostazione predefinita "MQSeries".
5. Nella richiesta comandi (2):
   - Per utilizzare una connessione TCP/IP, digitare un nome host del server di MQSeries.
   - Per utilizzare una connessione nativa (modalità binding), lasciare il campo vuoto. Non immettere un nome.

Ecco un esempio delle richieste e delle risposte che potrebbero apparire. Le richieste e le risposte effettive dipendono dalla rete di MQSeries.

```
Immettere il tipo di connessione (MQSeries)                    :(MQSeries)(1)
Immettere l'indirizzo IP del server MQSeries                        : myhost(2)
Immettere la porta a cui collegarsi                                 :(1414)(3)
Immettere il nome del canale di collegamento del server             : JAVA.CHANNEL(3)
Immettere il nome del programma di gestione code                    :
Esito positivo: Collegamento al gestore code.
Esito positivo: Aperto SYSTEM.DEFAULT.LOCAL.QUEUE
Esito positivo: Collocare un messaggio in SYSTEM.DEFAULT.LOCAL.QUEUE
Esito positivo: Ricevuto messaggio da SYSTEM.DEFAULT.LOCAL.QUEUE
Esito positivo: Chiuso SYSTEM.DEFAULT.LOCAL.QUEUE
Esito positivo: Scollegati dal programma di gestione code

Test completati -
ESITO POSITIVO: Il trasporto funziona correttamente.
Premere Invio per continuare ...
```

**Note:**
1. Se si sceglie una connessione server, non saranno visibili le richieste contrassegnate (3).
2. Su z/OS & OS/390, lasciare il campo vuoto nella richiesta comandi (2).
3. In OS/400 è possibile eseguire il comando `java MQIVP` solo dall'interfaccia interattiva Qshell (Qshell è l'opzione 30 di OS/400, 5769-SS1). In alternativa, è possibile eseguire l'applicazione utilizzando il comando CL `RUNJVA CLASS(MQIVP)`.
4. Per utilizzare i binding di MQSeries per Java su OS/400, è necessario assicurarsi che la libreria QMQMJAVA si trovi nell'elenco delle librerie.

### Utilizzo della connettività VisiBroker

Se si utilizza VisiBroker, le procedure descritte nella sezione "Configurazione del gestore code per l'accettazione di connessioni client" a pagina 15 non sono richieste.

Per verificare un'installazione che utilizza VisiBroker, attenersi alle procedure descritte nella sezione "Verifica con l'applicazione di esempio" a pagina 18, ma alla richiesta comandi (1), digitare `VisiBroker`, rispettando la combinazione di lettere maiuscole e minuscole riportata.

## Esecuzione dei propri programmi MQ base Java

Per eseguire le proprie applet o applicazioni Java, utilizzare le procedure descritte per i programmi di verifica, sostituendo il nome della propria applicazione al posto di "mqjavac.html" o "MQIVP".

Per informazioni sulla scrittura di applet e applicazioni MQ base Java, consultare "Parte 2. Programmazione con MQ base Java" a pagina 51.

## Risoluzione dei problemi di MQ base Java

Se l'esecuzione di un programma non viene completata correttamente, eseguire l'applet o il programma di verifica dell'installazione e seguire i suggerimenti riportati nei messaggi di diagnostica. Entrambi i programmi sono descritti nel "Capitolo 3. Utilizzo delle MQSeries classi per Java (MQ base Java)" a pagina 15.

Se i problemi persistono e diventa necessario rivolgersi all'assistenza IBM, potrebbe essere richiesto di attivare la funzione di analisi. Il metodo per l'esecuzione di questa operazione dipende dall'esecuzione in corso, a seconda che sia in modalità client o binding. Consultare le sezioni che seguono per informazioni sulle procedure più appropriate per il sistema utilizzato.

### Analisi dell'applet di esempio

Per eseguire l'analisi con l'applet di esempio, modificare il file `mqjavac.html`. Trovare la seguente riga:

```
<!PARAM name="trace" value="1">
```

Eliminare il punto esclamativo e passare il valore da 1 a un numero da 1 a 5, a seconda del livello di dettaglio richiesto. Più alto è il numero, maggiore è la quantità di informazioni raccolte. La riga dovrebbe quindi contenere quanto segue:

```
<PARAM name="trace" value="n">
```

dove "n" è un numero compreso tra 1 e 5.

### Esecuzione dell'analisi per MQ base Java

L'output dell'analisi appare nella console Java o nel file di registrazione Java del browser Web.

## Analisi dell'applicazione di esempio

Per analizzare il programma `MQIVP`, immettere quanto segue:

```
java MQIVP -trace n
```

dove "n" è un numero compreso tra 1 e 5, che dipende dal livello del dettaglio richiesto. Più alto è il numero, maggiore è la quantità di informazioni raccolte.

Per ulteriori informazioni su come utilizzare l'analisi, consultare la sezione "Analisi dei programmi MQ base Java" a pagina 78.

## Messaggi di errore

Di seguito sono riportati alcuni dei messaggi di errore più comuni:

**Impossibile identificare l'indirizzo IP dell'host locale**
Il server non è connesso alla rete.

*Azione consigliata*:: Connettere il server alla rete e riprovare.

**Impossibile caricare il file gatekeeper.ior**
Questo errore può verificarsi su un server Web che distribuisce le applet VisiBroker, quando il file `gatekeeper.ior` non si trova nella posizione corretta.

*Azione consigliata*: Riavviare il VisiBroker Gatekeeper dalla directory in cui è distribuita l'applet. Il file `gatekeeper` verrà scritto in questa directory.

**Errore: software mancante, potrebbe trattarsi di MQSeries o della variabile VBROKER_ADM**
Questo errore si verifica nel programma di esempio MQIVP se l'ambiente del software è incompleto.

*Azione consigliata*: Sul client, assicurarsi che la variabile di ambiente VBROKER_ADM sia impostata in modo che faccia riferimento alla directory di gestione di VisiBroker per Java (`adm`) e riprovare.

Sul server, assicurarsi che sia installata la versione più recente di MQ base Java e riprovare.

**NO_IMPLEMENT**
Esiste un problema di comunicazione che riguarda VisiBroker Smart Agents.

*Azione consigliata*: Consultare la documentazione relativa a VisiBroker.

**COMM_FAILURE**
Esiste un problema di comunicazione che riguarda VisiBroker Smart Agents.

*Azione consigliata*: Utilizzare lo stesso numero di porta per tutti i VisiBroker Smart Agents e riprovare. Consultare la documentazione relativa a VisiBroker.

**MQRC_ADAPTER_NOT_AVAILABLE**
Se viene visualizzato questo messaggio di errore quando si tenta di utilizzare VisiBroker, è probabile che sia impossibile trovare la classe JAVA org.omg.CORBA.ORB in CLASSPATH.

*Azione consigliata*: Assicurarsi che l'istruzione CLASSPATH includa il percorso dei file VisiBroker `vbjorb.jar` e `vbjapp.jar`.

**MQRC_ADAPTER_CONN_LOAD_ERROR**
Se viene visualizzato questo messaggio di errore durante l'esecuzione su OS/390, assicurarsi che le serie di dati MQSeries SCSQANLE e SCSQAUTH siano nell'istruzione STEPLIB.

## Messaggi di errore

# Capitolo 4. Utilizzo delle MQSeries classi per Java Message Service (MQ JMS)

In questo capitolo verranno descritte le seguenti attività:

- Come impostare il sistema per l'utilizzo dei programmi di verifica e di esempio
- Come eseguire il programma IVT (Installation Verification Test) point-to-point per verificare la propria installazione di MQSeries classi per Java Message Service
- Come eseguire il programma PSIVT (Publish/Subscribe Installation Verification Test) di esempio per verificare l'installazione Publish/Subscribe
- Come eseguire i propri programmi

## Configurazione post-installazione

**Nota:** Ricordarsi di consultare il file README, installato con i programmi MQ Java, per informazioni che possono sostituire questo manuale.

Per rendere disponibili tutte le risorse necessarie per i programmi MQ JMS, è necessario aggiornare le seguenti variabili di sistema:

**Percorso di ricerca delle classi**

Un funzionamento corretto dei programmi JMS presuppone che una serie di pacchetti Java sia disponibile per JVM. E' necessario specificarli nel percorso di ricerca delle classi dopo aver ottenuto e installato i pacchetti necessari.

Aggiungere i seguenti file .jar al percorso di ricerca delle classi:

- com.ibm.mq.jar
- com.ibm.mqjms.jar
- connector.jar
- jms.jar
- jndi.jar[1]
- jta.jar
- ldap.jar[1]
- providerutil.jar

**Variabili di ambiente**

Nella sottodirectory `bin` dell'installazione di MQ JMS sono presenti diversi script da utilizzare come vere e proprie scorciatoie per l'esecuzione di alcune tra le operazioni più comuni. Molti script presuppongono che la variabile di ambiente `MQ_JAVA_INSTALL_PATH` sia definita e che punti alla directory in cui MQ JMS è installato. Non è obbligatorio impostare questa variabile, tuttavia se non viene impostata è necessario modificare di conseguenza gli script nella directory `bin`.

In Windows NT, è possibile impostare il percorso di ricerca delle classi e la nuova variabile di ambiente utilizzando la scheda **Ambiente** delle **Proprietà di sistema**.

1. Per z/OS & OS/390, utilizzare `ibmjndi.jar` e `jndi.jar` da `/usr/lpp/ldap/lib` invece di `jndi.jar` e `ldap.jar`. Questi file vengono forniti con il sistema operativo.

In UNIX, verrebbero normalmente impostate dagli script di accesso di ciascun utente. In tutte le piattaforme, è possibile scegliere di utilizzare gli script per gestire percorsi di classe differenti e altre variabili di ambiente per diversi progetti.

## Configurazione ulteriore per la modalità Publish/Subscribe

Prima di poter utilizzare l'implementazione di MQ JMS di JMS Publish/Subscribe, sono richieste alcune operazioni di configurazione ulteriori:

**Assicurarsi che il Broker sia in esecuzione**

Per verificare che il broker MQSeries Publish/Subscribe sia installato ed in esecuzione, utilizzare il comando:

```
dspmqbrk -m MY.QUEUE.MANAGER
```

dove `MY.QUEUE.MANAGER` è il nome del gestore code su cui è in esecuzione il broker. Se il broker è in esecuzione, verrà visualizzato un messaggio simile a quello riportato di seguito:

```
Broker
del messaggio MQSeries per il gestore code MY.QUEUE.MANAGER in esecuzione.
```

Se il sistema operativo segnala che non è in grado di eseguire il comando `dspmqbrk`, assicurarsi che il broker MQSeries Publish/Subscribe sia stato installato correttamente.

Se invece indica che il broker non è attivo, attivarlo utilizzando il comando:

```
strmqbrk -m MY.QUEUE.MANAGER
```

**Creare le code di sistema MQ JMS**

Affinché l'implementazione di MQ JMS Publish/Subscribe funzioni correttamente, è necessario creare un certo numero di code di sistema. Per facilitare l'esecuzione di questa attività, viene fornito uno script nella sottodirectory `bin` dell'installazione di MQ JMS. Per utilizzare lo script, immettere i seguenti comandi:

**Per iSeries & AS/400:**

1. Copiare lo script dal file system integrato nella libreria del file system nativa utilizzando un comando analogo a

   ```
   CPYFRMSTMF FROMSTMF('/QIBM/ProdData/mqm/java/bin/MQJMS_PSQ.mqsc')
              TOMBR('/QSYS.LIB/QGPL.LIB/QCLSRC.FILE/MQJMS_PSQ.MBR')
   ```

2. Chiamare il file script utilizzando STRMQMMQSC:

   ```
   STRMQMMQSC  SRCMBR(MQJMS_PSQ) SRCFILE(QGPL/QCLSRC)
   ```

**Per z/OS & OS/390:**

1. Copiare lo script dall'HFS in un PDS utilizzando un comando TSO analogo a

   ```
   OGET '/usr/lpp/mqm/java/bin/MQJMS_PSQ.mqsc' 'USERID.MQSC(MQJMSPSQ)'
   ```

   Il PDS dovrebbe essere in formato blocco-fisso con una lunghezza record di 80.

2. Sia che si utilizzi l'applicazione CSQUTIL per eseguire questo script del comando o che si aggiunga lo script nella concatenazione CSQINP2 DD nella JCL della task avviata dal gestore code. In entrambi i casi, consultare il manuale *MQSeries for OS/390 System Management Guide* per maggiori dettagli.

**Per altre piattaforme:**

```
runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc
```

Se si verifica un errore, verificare che il nome del gestore code sia stato digitato correttamente e che il gestore code sia in esecuzione.

### Esecuzione di un broker su un gestore code remoto

Per un'operazione con un broker in esecuzione su un gestore code remoto, è richiesta un'ulteriore impostazione.

1. Definire una coda di trasmissione sul gestore code remoto con un nome coda che corrisponde al gestore code locale. Questi nomi devono corrispondere per una corretta distribuzione dei messaggi da MQSeries.
2. Definire un canale mittente sul gestore code remoto ed un canale destinatario sul gestore code locale. Il canale mittente dovrebbe utilizzare la coda di trasmissione definita nel passaggio1.
3. Impostare il gestore code locale per la comunicazione con il broker remoto:
   a. Definire una coda di trasmissione locale. E' necessario assegnarle lo stesso nome del gestore code in esecuzione sul broker remoto.
   b. Definire il mittente locale ed i canali destinatario remoti nel gestore code del broker remoto. Il canale mittente dovrebbe utilizzare la coda di trasmissione definita nel passaggio3a.
4. Per far funzionare il broker remoto, seguire i passaggi di seguito riportati:
   a. Avviare il gestore code del broker remoto.
   b. Avviare un listener per il gestore code del broker remoto (canali TCP/IP).
   c. Avviare i canali mittente e destinatario sul gestore code locale.
   d. Avviare il broker sul gestore code remoto.

      Un esempio del comando è

      ```
      strmqbrk -m MyBrokerMgr
      ```
5. Per far funzionare il gestore code locale per comunicare con il broker remoto, seguire i passaggi riportati di seguito:
   a. Avviare il gestore code locale.
   b. Avviare un listener per il gestore code locale.
   c. Avviare i canali mittente e destinatario sul gestore code del broker remoto.

## Code che richiedono l'autorizzazione per utenti che non dispongono di privilegi

Gli utenti senza privilegi necessitano di un'autorizzazione per accedere alle code utilizzate da JMS. Per informazioni sul controllo dell'accesso in MQSeries, consultare il capitolo relativo alla protezione degli oggeti MQSeries in *MQSeries System Administration*.

Per la modalità point-to-point JMS, le emissioni del controllo dell'accesso sono simili a quelle relative alle MQSeries classi per Java:

- Le code utilizzate da QueueSender richiedono l'autorizzazione put.
- Le code utilizzate da QueueReceiver e QueueBrowser richiedono le autorizzazioni get, inq e browse.
- Il metodo QueueSession.createTemporaryQueue richiede l'accesso alla coda del modello definita nel campo QueueConnectionFactory temporaryModel. In base all'impostazione predefinita, sarà SYSTEM.DEFAULT.MODEL.QUEUE.

Per la modalità JMS publish/subscribe, vengono utilizzate le seguenti code di sistema:

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

Inoltre tutte le applicazioni che pubblicano messaggi richiedono l'accesso alla coda STREAM specificata nella connessione utilizzata. Il valore predefinito è:

SYSTEM.BROKER.DEFAULT.STREAM

Se si utilizza la funzionalità ConnectionConsumer, potrebbero essere necessarie autorizzazioni aggiuntive. Le code da leggere attraverso ConnectionConsumer necessitano di autorizzazioni get, inq e browse. La coda delle lettere in attesa del sistema, qualsiasi coda accodata nei rifiuti o code di report utilizzati mediante ConnectionConsumer, necessitano di autorizzazioni put e passall.

# Esecuzione del programma IVT point-to-point

In questa sezione verrà illustrato il programma di verifica dell'installazione point-to-point IVT (installation verification test) fornito con MQ JMS.

Il programma IVT tenta di verificare l'installazione connettendosi al gestore code predefinito sulla macchina locale, utilizzando MQ JMS in modalità binding. Invia quindi un messaggio alla coda `SYSTEM.DEFAULT.LOCAL.QUEUE` e lo rilegge nuovamente.

E' possibile eseguire il programma in una delle due possibili modalità.

**Con la ricerca JNDI degli oggetti amministrati**
La modalità JNDI impone al programma di ottenere gli oggetti amministrati da uno spazio dei nomi JNDI, l'operazione prevista delle applicazioni client JMS. Fare riferimento al "Amministrazione degli oggetti JMS" a pagina 40 per una descrizione degli oggetti amministrati. Questo metodo di richiamo presenta gli stessi prerequisiti dello strumento di amministrazione (consultare il "Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS" a pagina 35).

**Senza la ricerca JNDI degli oggetti amministrati**
Se non si desidera utilizzare JNDI, gli oggetti amministrati possono essere creati in fase di runtime eseguendo il programma IVT in modalità non-JNDI. Dal momento che un archivio basato su JNDI è relativamente complesso da configurare, si consiglia di eseguire IVT per la prima volta senza JNDI.

## Verifica point-to-point senza JNDI

Uno script, denominato `IVTRun` su UNIX o `IVTRun.bat` su Windows NT, viene fornito per l'esecuzione di IVT. Il file viene installato nella sottodirectory `bin` dell'installazione.

Per eseguire la verifica senza JNDI, emettere il seguente comando:

```
IVTRun -nojndi [-m <qmgr>]
```

Per la modalità client, per eseguire la verifica senza JNDI, emettere il seguente comando:

```
IVTRun -nojndi -client -m <qmgr> -host <nomehost> [-port <porta>]
             [-channel <canale>]
```

dove:

**qmgr** è il nome del gestore code a cui si desidera effettuare la connessione

**nomehost** è l'host su cui è in esecuzione il gestore code

**porta** è la porta TCP/IP su cui il listener del gestore è in esecuzione (quella predefinita è 1414)

**canale** è il canale della connessione client (quello predefinito è SYSTEM.DEF.SVRCONN)

Se la verifica viene completata correttamente, dovrebbe essere visualizzato un risultato simile al seguente:

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - 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: Message Class:   jms_text          JMSType:          null
JMSDeliveryMode: 2                              JMSExpiration:    0
JMSPriority:     4                              JMSMessageID:     ID:414d5120716
d312020202020202020203000c43713400000
JMSTimestamp:    935592657000                   JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:      null
JMSRedelivered:  false
JMS_IBM_Format:MQSTR            JMS_IBM_PutApplType:11
JMSXGroupSeq:1                  JMSXDeliveryCount:0
JMS_IBM_MsgType:8               JMSXUserID:kingdon
JMSXApplID:D:\jdk1.1.8\bin\java.exe
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 point-to-point con JNDI

Per eseguire il programma IVT con JNDI, è necessario che il server LDAP sia in esecuzione e sia configurato per accettare gli oggetti Java. Se viene visualizzato il messaggio riportato di seguito, esso indica che è in corso una connessione al server LDAP, ma il server non è configurato correttamente:

```
Unable to bind to object
```

Questo messaggio indica che il server non sta memorizzando gli oggetti Java oppure che le autorizzazioni sugli oggetti o sul suffisso non sono corrette. Consultare la sezione “Verifica della configurazione del server LDAP” a pagina 393.

Inoltre, i seguenti oggetti amministrati devono essere richiamabili da uno spazio nomi JNDI:
- MQQueueConnectionFactory
- MQQueue

Uno script, denominato `IVTSetup` su UNIX o `IVTSetup.bat` su Windows NT, viene fornito per la creazione automatica di questi oggetti. Immettere il comando:

```
IVTSetup
```

Lo script richiama lo strumento di gestione di MQ JMS (vedere il “Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS” a pagina 35) e crea gli oggetti in uno spazio dei nomi JNDI.

L’oggetto MQQueueConnectionFactory è collegato all’interno del nome `ivtQCF` (per LDAP, `cn=ivtQCF`). Tutte le proprietà sono valori predefiniti:

```
TRANSPORT(BIND)
PORT(1414)
HOSTNAME(localhost)
CHANNEL(SYSTEM.DEF.SVRCONN)
VERSION(1)
CCSID(819)
TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
QMANAGER()
```

L’oggetto MQQueue è collegato all’interno del nome `ivtQ` (`cn=ivtQ`). Il valore della proprietà `QUEUE` diventa `QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)`. Tutte le altre proprietà presentano valori predefiniti:

```
PERSISTENCE(APP)
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
EXPIRY(APP)
TARGCLIENT(JMS)
ENCODING(NATIVE)
VERSION(1)
CCSID(1208)
PRIORITY(APP)
QMANAGER()
```

Una volta creati gli oggetti amministrati nello spazio dei nomi JNDI, eseguire lo script `IVTRun` (`IVTRun.bat` su Windows NT) utilizzando il seguente comando:

```
IVTRun [ -t ]  -url <"providerURL"> [ -icf <initCtxFact> ]
```

dove:

**-t** indica l’attivazione dell’analisi (in base all’impostazione predefinita l’analisi è disattivata)

**providerURL** è la posizione JNDI degli oggetti amministrati. Se il context factory iniziale predefinito è in uso, si tratta dell’URL LDAP con il seguente formato:

```
ldap://nomehost.societa.com/contextName
```

Se viene utilizzato un provider di servizi del file system, (vedere `initCtxFact` di seguito), l’URL presenta il seguente formato:

```
file://directorySpec
```

**Nota:** Racchiudere la stringa *providerURL* tra virgolette (″).

**initCtxFact** è il nome della classe del context factory iniziale. L'impostazione predefinita è per un fornitore di servizi LDAP e presenta il valore:

```
com.sun.jndi.ldap.LdapCtxFactory
```

Se viene utilizzato un provider di servizi del file system, impostare questo parametro su:

```
com.sun.jndi.fscontext.RefFSContextFactory
```

Se il test viene completato in modo corretto, l'output è simile a quello non-JNDI, ad eccezione del fatto che le righe 'create' `QueueConnectionFactory` e `Queue` indicano un richiamo dell'oggetto da JNDI. Il frammento di codice che segue mostra un esempio.

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) 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
 ...
 ...
```

Sebbene non sia strettamente necessario, è buona norma rimuovere gli oggetti creati dallo script IVTSetup da uno spazio dei nomi JNDI. A tale scopo viene fornito uno script chiamato `IVTTidy` (`IVTTidy.bat` su Windows NT).

## Ripristino degli errori IVT

Se il test non viene eseguito correttamente, potrebbero risultare utili le note che seguono:

- Per informazioni sui messaggi di errore che riguardano il percorso di ricerca delle classi, verificare che il percorso di ricerca delle classi sia impostato correttamente, come descritto nella sezione "Configurazione post-installazione" a pagina 23.
- L'esecuzione del programma IVT potrbbe non riuscire ed essere accompagnata da l messaggio "Impossibile creare MQQueueManager", con un ulteriore messaggio che include il numero 2059. Ciò indica che MQSeries non è riuscito a connettersi al gestore code locale predefinito sulla macchina in cui è stato eseguito il programma IVT. Verificare che il gestore code sia in esecuzione e che sia contrassegnato come gestore code predefinito.
- Il messaggio "Impossibile aprire la coda MQ " indica che MQSeries si è connesso al gestore code predefinito, ma non può aprire SYSTEM.DEFAULT.LOCAL.QUEUE. Ciò potrebbe indicare che la coda non esiste sul gestore code predefinito oppure che la coda non sia abilitata per PUT e GET. Aggiungere o abilitare la coda per la durata del test.

Nella Tabella 5 sono riportate le classi verificate da IVT e il pacchetto da cui provengono:

*Tabella 5. Classi verificate da IVT*

| Classe | File Jar |
|---|---|
| Classi MQSeries JMS | com.ibm.mqjms.jar |

*Tabella 5. Classi verificate da IVT (Continua)*

| Classe | File Jar |
|---|---|
| com.ibm.mq.MQMessage | com.ibm.mq.jar |
| javax.jms.Message | jms.jar |
| javax.naming.InitialContext | jndi.jar |
| javax.resource.cci.Connection | connector.jar |
| javax.transaction.xa.XAException | jta.jar |
| com/sun/jndi/toolkit/ComponentDirContext | providerutil.jar |
| com.sun.jndi.ldap.LdapCtxFactory | ldap.jar |

# Il programma ITV (Installation Verification Test) Publish/Subscribe

Il programma PSITV (Publish/Subscribe Installation Verification Test) viene fornito solo in formato compilato e si trova nel pacchetto `com.ibm.mq.jms`.

Il test richiede un broker come il broker publish/subscribe MQSeries (SupportPac MA0C) o MQSeries Integrator V2 da installare ed eseguire.

Il programma PSIVT tenta di:

1. Creare un publisher, `p`, che pubblica sull'argomento `MQJMS/PSIVT/Information`
2. Creare un subscriber, `s`, che sottoscrive sull'argomento `MQJMS/PSIVT/Information`
3. Utilizzare `p` per pubblicare un semplice messaggio di testo
4. Utilizzare `s` per ricevere un messaggio in attesa sulla propria coda di input

Quando si esegue il PSIVT, il publisher pubblica il messaggio e il subscriber lo riceve e lo visualizza. Il publisher pubblica sul flusso predefinito del broker. Il subscriber non è durevole, non esegue selezione di messaggi e accetta messaggi da connessioni locali. Esegue una ricezione sincrona, attendendo un massimo di 5 messaggi l'arrivo di un messaggio.

E' possibile eseguire il PSIVT, come l'IVT, in modalità JNDI o standalone. La modalità JNDI utilizza JNDI per richiamare un `TopicConnectionFactory` e un `Topic` da uno spazio dei nomi JNDI. Se non si utilizza JNDI, gli oggetti vengono creati in fase di runtime.

## Verifica Publish/Subscribe senza JNDI

Uno script chiamato `PSIVTRun` (`PSIVTRun.bat` su Windows NT) viene fornito per eseguire PSIVT. Il file si trova nella sottodirectory `bin` dell'installazione.

Per eseguire la verifica senza JNDI, emettere il seguente comando:

```
PSIVTRun -nojndi [-m <qmgr>] [-bqm <broker>] [-t]
```

Per la modalità client, per eseguire la verifica senza JNDI, emettere il seguente comando:

```
PSIVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
               [-channel <channel>] [-bqm <broker>] [-t]
```

dove:

**-nojndi** significa nessuna ricerca JNDI degli oggetti amministrati

**qmgr** è il nome del gestore code a cui si desidera effettuare la connessione

**nomehost** è l'host su cui è in esecuzione il gestore code

**porta** è la porta TCP/IP su cui il listener del gestore è in esecuzione (quella predefinita è 1414)

**canale** è il canale della connessione client (quello predefinito è SYSTEM.DEF.SVRCONN)

**broker** è il nome del gestore code remoto sul quale il broker è in esecuzione. Se tale nome non viene specificato, viene assunto il valore utilizzato per **qmgr**.

**-t** indica l'attivazione dell'analisi (in base all'impostazione predefinita l'analisi è disattivata)

Se la verifica viene completata correttamente, dovrebbe essere visualizzato un risultato simile al seguente:

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes per Java(tm) Message ServicePublish/Subscribe Installation Verification Test
Creating a TopicConnectionFactory
Creating a Topic
Creating a Connection
Creating a Session
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...

Got message:

JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d2e504f4c415249532e4254b7dc3753700000
  JMSTimestamp:    937232048000
  JMSCorrelationID:ID:414d515800000000000000000000000000000000000000000000
  JMSDestination:  topic
  ://MQJMS/PSIVT/Information
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_Format:MQSTR
  UNIQUE_CONNECTION_ID:937232047753
  JMS_IBM_PutApplType:26
  JMSXGroupSeq:1
  JMSXDeliveryCount:0
  JMS_IBM_MsgType:8
  JMSXUserID:hollingl
  JMSXApplID:QM.POLARIS.BROKER
A simple text message from the MQJMSPSIVT program

Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT completed OK
PSIVT completato
```

## Verifica Publish/Subscribe con JNDI

Per eseguire il programma PSIVT in modalità JNDI, è necessario richiamare due oggetti amministrati da uno spazio dei nomi JNDI:
- Un bound TopicConnectionFactory con il nome `ivtTCF`
- Un bound Topic con il nome `ivtT`

E' possibile definire questi oggetti utilizzando lo strumento di amministrazione MQ JMS (vedere il "Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS" a pagina 35) e utilizzando i seguenti comandi:

```
DEFINE TCF(ivtTCF)
```

Questo comando definisce il TopicConnectionFactory.

```
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)
```

Questo comando definisce il Topic.

Queste definizioni presuppongono che sia disponibile un gestore code predefinito su cui è in esecuzione il broker. Per informazioni dettagliate su come configurare questi oggetti per l'utilizzo di un gestore code non predefinito, consultare la sezione "Amministrazione degli oggetti JMS" a pagina 40. Questi oggetti dovrebbero risiedere in un contesto puntato dal parametro della riga comandi `-url` descritto di seguito.

Per eseguire il test in modalità JNDI, immettere il seguente comando:

```
PSIVTRun -url <purl> [-icf <initcf>] [-t]
```

dove:

**-t** indica l'attivazione dell'analisi (in base all'impostazione predefinita l'analisi è disattivata)

**-url <purl>** è URL del percorso JNDI in cui risiedono gli oggetti amministrati

**-icf <initcf>** è il initialContextFactory per JNDI [`com.sun.jndi.ldap.LdapCtxFactory` ]

Se il test viene completato in maniera corretta, l'output è simile a quello non JNDI, ad eccezione del fatto che le righe "create" `QueueConnectionFactory` e `Queue` indicano un richiamo dell'oggetto da JNDI.

## Ripristino degli errori PSIVT

Se il test non viene eseguito correttamente, potrebbero risultare utili le note che seguono:

- La visualizzazione del seguente messaggio:

  ```
  *** The broker is not running! Please start it using 'strmqbrk' ***
  ```

  indica che il broker è installato nel gestore code di destinazione ma la coda di controllo contiene alcuni messaggi in eccesso. Pertanto il broker non è in esecuzione. Per avviarlo, utilizzare il comando `strmqbrk`. (Vedere "Configurazione ulteriore per la modalità Publish/Subscribe" a pagina 24.)
- Se viene visualizzato il messaggio:

  ```
  Unable to connect to queue manager: <default>
  ```

  assicurarsi che nel sistema MQSeries sia configurato un gestore code predefinito.
- Se viene visualizzato il messaggio:

```
Unable to connect to queue manager: ...
```

assicurarsi che l'oggetto TopicConnectionFactory amministrato utilizzato dal programma PSIVT sia configurato con nome di gestore code valido. In alternativa, se è stata utilizzata l'opzione `-nojndi`, assicurarsi che sia stata fornito un gestore code valido (utilizzare l'opzione `-m`).

- Se viene visualizzato il messaggio:

```
Unable to access broker control queue on queue manager: ...
Please ensure the broker is installed on this queue manager
```

assicurarsi che l'oggetto TopicConnectionFactory amministrato utilizzato dal programma PSIVT sia configurato con il nome del gestore code in cui è installato il broker. Se è stata utilizzata l'opzione `-nojndi`, assicurarsi che sia stato fornito un nome di gestore code (utilizzare l'opzione `-m`).

## Esecuzione dei programmi MQ JMS

Per informazioni sulla scrittura dei propri programmi MQ JMS, consultare il "Capitolo 10. Scrittura di programmi MQ JMS" a pagina 189.

MQ JMS include un file del programma di utilità, `runjms` (`runjms.bat` su Windows NT), che consente di eseguire i programmi forniti e quelli scritti dall'utente.

Il programma di utilità fornisce i percorsi predefiniti dell'analisi e dei file di registrazione e consente di aggiungere tutti i parametri di runtime dell'applicazione necessari. Lo script fornito presuppone che la variabile di ambiente MQ_JAVA_INSTALL_PATH sia impostata sulla directory in cui è installato MQ JMS. Presuppone inoltre che le sottodirectory `trace` e `log` all'interno della directory siano utilizzate rispettivamente per l'analisi e l'output della registrazione. Si tratta solo di percorsi consigliati ed è possibile modificare lo script per utilizzare qualsiasi directory venga scelta.

Utilizzare il seguente comando per eseguire l'applicazione:

```
runjms <nomeclasse
applicazione> [application-specific arguments]
```

Per informazioni sulla scrittura di applet e applicazioni MQ JMS, consultare "Parte 3. Programmazione con MQ JMS" a pagina 187.

## Risoluzione dei problemi

Se l'esecuzione di un problema non viene completata correttamente, eseguire il programma di verifica dell'installazione, descritto nel "Capitolo 4. Utilizzo delle MQSeries classi per Java Message Service (MQ JMS)" a pagina 23 e seguire i suggerimenti forniti nei messaggi di diagnostica.

### Analisi dei programmi

La funzione di analisi di MQ JMS viene fornita per assistere il personale IBM nella diagnosi dei problemi dei consumatori.

La funzione di analisi è disablitata in base all'impostazione predefinita in quanto l'output diventa rapidamente molto grande ed è improbabile che sia utile in circostanze normali.

### esecuzione della funzione di analisi di MQ JMS

Se viene chiesto di fornire l'output dell'analisi, è possibile abilitarla impostando la proprietà Java MQJMS_TRACE_LEVEL su uno dei seguenti valori:

**on** analizza solo le chiamate MQ JMS

**base** analizza sia le chiamate MQ JMS che le chiamate MQ base Java sottostanti

Ad esempio:

```
java -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

Per disabilitare l'analisi, impostare MQJMS_TRACE_LEVEL su **off**.

In base all'impostazione predefinita, l'analisi è l'output in un file denominato `mqjms.trc` nella directory di lavoro corrente. E' possibile reindirizzarla in un'altra directory utilizzando la proprietà Java `MQJMS_TRACE_DIR`.

Ad esempio:

```
java -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```

Lo script del programma di utilità `runjms` imposta queste proprietà utilizzando le variabili di ambiente `MQJMS_TRACE_LEVEL` e `MQ_JAVA_INSTALL_PATH` nel modo seguente:

```
java -DMQJMS_LOG_DIR=%MQ_JAVA_INSTALL_PATH%\log
-DMQJMS_TRACE_DIR=%MQ_JAVA_INSTALL_PATH%\trace
-DMQJMS_TRACE_LEVEL=%MQJMS_TRACE_LEVEL% %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Si tratta solo di un suggerimento, che può essere modificato in base alle proprie esigenze.

## Registrazione

La funzione di registrazione di MQ JMS viene fornita per riportare i problemi gravi, in particolare quelli che indicano errori di configurazione piuttosto che errori di programmazione. In base all'impostazione predefinita, l'output della registrazione viene inviato al flusso `System.err`, che generalmente appare sul `stderr` della console in cui è in esecuzione il JVM.

E' possibile ridirigere l'output in un file utilizzando una proprietà Java che specifica il nuovo percorso, ad esempio:

```
java -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

Lo script del programma di utilità `runjms`, nella directory bin dell'installazione di MQ JMS imposta questa proprietà su:

```
<MQ_JAVA_INSTALL_PATH>/log
```

dove `MQ_JAVA_INSTALL_PATH` è il percorso dell'installazione di MQ JMS. Si tratta solo di un suggerimento, che è possibile modificare in base alle esigenze specifiche.

Quando la registrazione viene ridiretta verso un file, l'output è in formato binario. Per visualizzare il file di registrazione, viene fornito il programma di utilità `formatLog` (`formatLog.bat` in Windows NT), che converte il file in formato di solo testo. Il programma di utilità viene memorizzato nella directory `bin` dell'installazione di MQ JMS. Eseguire la conversione nel modo seguente:

```
formatLog <fileinput> <fileoutput>
```

# Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS

Lo strumento di amministrazione consente agli amministratori di definire le proprietà di otto tipi di oggetti MQ JMS e di memorizzarli all'interno di uno spazio dei nomi JNDI. I client JMS possono quindi recuperare questi oggetti amministrati dallo spazio dei nomi tramite JNDI e utilizzarli.

Gli oggetti JMS che è possibile amministrare utilizzando lo strumento sono:
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

Per informazioni dettagliate su questi oggetti, fare riferimento alla sezione "Amministrazione degli oggetti JMS" a pagina 40.

**Nota:** JMSWrapXAQueueConnectionFactory e JMSWrapXATopicConnectionFactory sonoclassi specifiche di WebSphere e sono contenute nel pacchetto **com.ibm.ejs.jms.mq**.

Lo strumento consente inoltre agli amministratori di manipolare i subcontesti dello spazio dei nomi delle directory all'interno di JNDI. Consultare la sezione "Manipolazione dei subcontesti" a pagina 40.

## Richiamo dello strumento di amministrazione

Lo strumento di amministrazione è dotato di un'interfaccia a riga comandi. E' possibile utilizzarlo in modo interattivo o per avviare un processo batch. La modalità interattiva fornisce un prompt dei comandi in cui è possibile immettere i comandi di amministrazione. Nella modalità batch il comando per avviare lo strumento include il nome di un file che contiene uno script dei comandi di amministrazione.

Per avviare lo strumento in modalità interattiva, immettere il comando:

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

dove:

| | |
|---|---|
| **-t** | attiva l'analisi (in base all'impostazione predefinita l'analisi è disattivata) |
| **-v** | Produce un output verbose (l'impostazione predefinita è l'output terse) |
| **-cfg config_filename** | Il nome del file di configurazione alternativo (vedere "Configurazione" a pagina 36) |

Viene visualizzato un prompt dei comandi che indica che lo strumento è pronto ad accettare i comandi di amministrazione. Questo prompt inizialmente appare come:

### Richiamo dello strumento di amministrazione

```
InitCtx>
```

e indica che il contesto corrente, vale a dire il contesto JNDI a cui al momento fanno riferimento tutte le operazioni di assegnazione nomi e della directory, è il contesto iniziale definito nel parametro di configurazione PROVIDER_URL (vedere "Configurazione").

Man mano che si entra nello spazio dei nomi della directory, il prompt cambia per riflettere questa operazione, in modo che possa sempre visualizzare il contesto corrente.

Per avviare lo strumento in modalità batch, immettere il comando:

```
JMSAdmin <test.scp
```

dove *test.scp* è un file script che contiene i comandi di amministrazione (vedere "Comandi di amministrazione" a pagina 39). L'ultimo comando nel file deve essere il comando `END`.

## Configurazione

E' necessario configurare lo strumento di amministrazione con dei valori per i tre parametri che seguono:

**INITIAL_CONTEXT_FACTORY**
Indica il provider di servizi utilizzato dallo strumento. Sono disponibili al momento tre valori supportati per questa proprietà:
- com.sun.jndi.ldap.LdapCtxFactory (per LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (per il contesto del file system)
- com.ibm.ejs.ns.jndi.CNInitialContextFactory (per un utilizzo con il repository CosNaming WebSphere')

Inoltre, su z/OS & OS/390, com.ibm.jndi.LDAPCtxFactory l'accesso è supportato e fornito ad un server LDAP. Tuttavia, bisogna notare che esso è incompatibile con com.sun.jndi.ldap.LdapCtxFactory, in quanto gli oggetti creati utilizzando Initial Context Factory non possono essere letti o modificati dagli altri.

**PROVIDER_URL**
Indica l'URL del contesto iniziale della sessione, la radice di tutte le operazioni JNDI eseguite dallo strumento. Tre forme di questa proprietà sono attualmente supportate:
- ldap://nomehost/nomecontesto (per LDAP)
- file:[unità]/nomepercorso (per il contesto del file system)
- iiop://hostname[:porta] /[?TargetContext=ctx] (per accedere alllo spazio dei nomi CoseNaming "base" WebSphere)

**SECURITY_AUTHENTICATION**
Indica se JNDI trasmette le credenziali di sicurezza al provider di servizi. Questo parametro viene utilizzato solo quando viene utilizzato un provider di servizi LDAP. Questa proprietà può prendere al momento uno dei tre valori indicati di seguito:
- none (autenticazione anonima)
- simple (autenticazione semplice)
- CRAM-MD5 (meccanismo di autenticazione CRAM-MD5)

Se non viene fornito un valore valido, la proprietà sarà impostata automaticamente su none. Consultare la sezione “Sicurezza” per ulteriori dettagli sulla sicurezza con lo strumento di amministrazione.

Questi parametri sono impostati in un file di configurazione. Quando si richiama lo strumento, è possibile specificare questa configurazione utilizzando il parametro della riga comandi `-cfg`, come viene illustrato nella sezione “Richiamo dello strumento di amministrazione” a pagina 35. Se non si specifica un nome del file di configurazione, lo strumento tenta di caricare il file di configurazione predefinito (`JMSAdmin.config`). Prima cerca il file nella directory corrente, quindi nella directory `<MQ_JAVA_INSTALL_PATH>/bin`, dove “`<MQ_JAVA_INSTALL_PATH>`” è il percorso dell’installazione MQ JMS.

Il file di configurazione è un file di solo testo composto da un insieme di coppie chiave-valore, separate da un “=”. come viene illustrato nell’esempio che segue:

```
#Impostare il provider del servizio
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Impostare il contesto iniziale
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Impostare il tipo di autenticazione
    SECURITY_AUTHENTICATION=none
```

(Un “#” nella prima colonna della riga indica un commento o una riga non utilizzata.)

L’installazione è corredata da un file di configurazione di esempio chiamato `JMSAdmin.config` che si trova nella directory `<MQ_JAVA_INSTALL_PATH>/bin`. Modificare questo file in base alla configurazione del proprio sistema.

## Configurazione per WebSphere

Perché lo strumento di amministrazione (o qualsiasi applicazione client che richieda l’esecuzione di ricerche successive) possa funzionare con il repository CosNaming WebSphere’, è richiesta la seguente configurazione:

- CLASSPATH deve includere i file jar relativi a JNDI WebSphere’
  - Per WebSphere V3.5:

    `<WSAppserver>\lib\ujc.jar`
- PATH per WebSphere V.3.5 deve includere:

  `<WSAppserver>\jdk\jre\bin`

  dove “`<WSAppserver>`” è il percorso di installazione per WebSphere.

## Sicurezza

E’ necessario che gli amministratori siano informati dell’effetto della proprietà `SECURITY_AUTHENTICATION` descritta nella sezione “Configurazione” a pagina 36.

- Se questo parametro viene impostato su `none`, JNDI non trasmette alcuna credenziale di sicurezza sul provider dei servizi e viene eseguita una “autenticazione anonima”.
- Se questo parametro è impostato su `simple` o `CRAM-MD5`, le credenziali di sicurezza sono trasmesse attraverso JNDI al provider dei servizi sottostante. Queste credenziali di sicurezza hanno il formato di un nome distinto dell’utente (User DN) e di una password.

Se sono necessarie le credenziali di sicurezza, verranno chieste al momento dell’inizializzazione dello strumento.

**Nota:** Il testo digitato viene visualizzato sullo schermo, con l'inclusione della password. Pertanto assicurarsi che le password non vengano rivelate ad utenti non autorizzati.

Lo strumento non esegue autenticazioni in quanto l'attività è delegata al server LDAP. E' responsabilità dell'amministratore del server LDAP l'impostazione e il mantenimento dei privilegi di accesso a diverse parti della directory. Se l'autenticazione non riesce, lo strumento visualizza un messaggio di errore appropriato e termina.

Informazioni più dettagliate sulla sicurezza e su JNDI sono contenute nella documentazione pubblicata nel sito Web della Sun Java (`http://java.sun.com`).

## Comandi di amministrazione

Quando il prompt dei comandi viene visualizzato, lo strumento è pronto ad accettare i comandi di amministrazione. I comandi di amministrazione presentano generalmente il seguente formato:

```
verb [param]*
```

dove *verb* è uno dei verb di amministrazione elencato nella Tabella 6. Tutti i comandi validi sono composti da almeno un verb (e solo uno), che appare all'inizio del comando nel formato standard o breve.

I parametri che un verb può assumere dipendono dal verb. Per esempio, il verb `END` non può assumere alcun parametro, ma il verb `DEFINE` può assumerne da 1 a 20. Dettagli dei verb che assumono almeno un parametro vengono descritti nelle sezioni successive di queso capitolo.

*Tabella 6. verb di amministrazione*

| Verb | Formato breve | Descrizione |
|---|---|---|
| ALTER | ALT | Modificare almeno una delle proprietà di un determinato oggetto amministrato |
| DEFINE | DEF | Creare e memorizzare un oggetto amministrato oppure creare un nuovo subcontesto |
| DISPLAY | DIS | Visualizzare le proprietà di uno o più oggetti amministrati oppure il contenuto del contesto corrente |
| DELETE | DEL | Rimuovere uno o più oggetti amministrati dallo spazio dei nomi oppure rimuovere un subcontesto vuoto |
| CHANGE | CHG | Modificare il contesto corrente, consentendo all'utente di esplorare lo spazio dei nomi della directory in qualsiasi punto al di sotto del contesto iniziale |
| COPY | CP | Effettuare una copia di un oggetto amministrato memorizzato, memorizzandolo con un nome alternativo |
| MOVE | MV | Modificare il nome con cui viene memorizzato un oggetto amministrato |
| END | | Chiudere lo strumento di amministrazione |

I nomi dei verb non distinguono tra lettere maiuscole e minuscole.

Generalmente, per terminare i comandi, si preme il tasto di ritorno a capo. Tuttavia, è possibile sostituire questa operazione digitando il simbolo "+" direttamente prima del ritorno a capo. In questo modo è possibile immettere comandi a più righe, come viene illustrato nell'esempio che segue:

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

Le righe che iniziano con uno dei caratteri *, # o / vengono trattate come commenti o righe ignorate.

## Manipolazione dei subcontesti

E' possibile utilizzare i verb `CHANGE`, `DEFINE`, `DISPLAY` e `DELETE` per manipolare i subcontesti degli spazi dei nomi delle directory. Il loro utilizzo viene descritto nella Tabella 7.

*Tabella 7. Sintassi e descrizioni dei comandi utilizzati per manipolare i subcontesti*

| Sintassi del comando | Descrizione |
|---|---|
| DEFINE CTX(ctxName) | Tenta di creare un nuovo subcontesto secondario del contesto corrente, che ha il nome ctxName. Non riesce se c'è una violazione della sicurezza, se il subcontesto esiste già o se il nome fornito non è valido. |
| DISPLAY CTX | Visualizza il contenuto del contesto corrente. Gli oggetti amministrati sono annotati con una `'a'`, i subcontesti con una `'[D]'`. Viene visualizzato anche il tipo Java di ciascun oggetto. |
| DELETE CTX(ctxName) | Tenta di eliminare il contesto secondario del contesto corrente che ha il nome ctxName. Non riesce se il contesto non viene trovato, non è vuoto o se c'è una violazione della sicurezza. |
| CHANGE CTX(ctxName) | Modifica il contesto corrente in modo che faccia riferimento al contesto secondario che ha il nome ctxName. Può essere fornito uno dei due valori speciali di ctxName:<br>**=UP** che sposta al livello principale del contesto corrente<br>**=INIT** che sposta direttamente al contesto iniziale<br><br>Non riesce se il contesto non esiste o se c'è una violazione della sicurezza. |

## Amministrazione degli oggetti JMS

In questa sezione verranno descritti gli otto tipi di oggetti che lo strumento di amministrazione può gestire. Sono inclusi dettagli su ciascuna delle proprietà configurabili e dei verb che le possono manipolare.

### Tipi di oggetto

Tabella 8 mostra gli otto tipi di oggetti amministrati. La colonna Parola chiave mostra la stringa che è possibile sostituire per *TYPE* nei comandi riportati nella Tabella 9 a pagina 42.

*Tabella 8. I tipi di oggetti JMS gestiti dallo strumento di amministrazione*

| Tipo di oggetto | Parola chiave | Descrizione |
|---|---|---|
| MQQueueConnectionFactory | QCF | L'implementazione MQSeries dell'interfaccia JMS QueueConnectionFactory. Si tratta di un oggetto predefinito per la creazione di connessioni nel dominio point-to-point di JMS. |

*Tabella 8. I tipi di oggetti JMS gestiti dallo strumento di amministrazione (Continua)*

| Tipo di oggetto | Parola chiave | Descrizione |
|---|---|---|
| MQTopicConnectionFactory | TCF | L'implementazione MQSeries dell'interfaccia JMS TopicConnectionFactory. Si tratta di un oggetto predefinito per la creazione di connessioni nel dominio publish/subscribe di JMS. |
| MQQueue | Q | L'implementazione MQSeries dell'interfaccia JMS Queue. Rappresenta una destinazione per i messaggi nel dominio point-to-point di JMS. |
| MQTopic | T | L'implementazione MQSeries dell'interfaccia JMS Topic. Rappresenta una destinazione per i messaggi nel dominio publish/subscribe di JMS. |
| MQXAQueueConnectionFactory[1] | XAQCF | L'implementazione MQSeries dell'interfaccia JMS XAQueueConnectionFactory. Si tratta di un oggetto predefinito per la creazione di connessioni nel dominio point-to-point di JMS che utilizza le versioni XA delle classi JMS. |
| MQXATopicConnectionFactory[1] | XATCF | L'implementazione MQSeries dell'interfaccia JMS XATopicConnectionFactory. Si tratta di un oggetto predefinito per la creazione di connessioni nel dominio publish/subscribe di JMS che utilizza le versioni XA delle classi JMS. |
| JMSWrapXAQueueConnectionFactory[2] | WSQCF | L'implementazione MQSeries dell'interfaccia JMS QueueConnectionFactory. Ciò rappresenta un oggetto predefinito per la creazione di connessioni nel dominio point-to-point di JMS che utilizza le versioni XA delle classi JMS con WebSphere. |
| JMSWrapXATopicConnectionFactory[2] | WSTCF | L'implementazione MQSeries dell'interfaccia JMS TopicConnectionFactory. Ciò rappresenta un oggetto predefinito per la creazione di connessioni nel dominio publish/subscribe di JMS che utilizza le versioni XA delle classi JMS con WebSphere. |

1. Queste classi vengono fornite per poter essere utilizzate dai fornitori di server applicazioni. E' improbabile che possano risultare direttamente utili ai programmatori delle applicazioni.
2. Utilizzare questo stile di ConnectionFactory se si desidera che le sessioni JMS partecipino alle transazioni globali coordinate da WebSphere.

## Verb utilizzati con gli oggetti JMS

E' possibile utilizzare i verb `ALTER`, `DEFINE`, `DISPLAY`, `DELETE`, `COPY` e `MOVE` per manipolare gli oggetti amministrati nello spazio dei nomi della directory. La Tabella 9 ne riepiloga l'utilizzo. Sostituire *`TYPE`* con la Parola chiave che rappresenta l'oggetto amministrato richiesto, come viene indicato nella Tabella 8 a pagina 40.

*Tabella 9. Sintassi e descrizioni dei comandi utilizzati per manipolare gli oggetti amministrati*

| Sintassi del comando | Descrizione |
|---|---|
| ALTER *TYPE*(name) [property]* | Tenta di aggiornare le proprietà del determinato oggetto amministrato con quelle fornite. Non riesce se c'è una violazione della sicurezza, se l'oggetto specificato non può essere trovato o se le nuove proprietà fornite non sono valide. |
| DEFINE *TYPE*(name) [property]* | Tenta di creare un oggetto amministrato del tipo *`TYPE`* con le proprietà fornite e cerca di memorizzarlo con il nome `name` nel contesto corrente. Non riesce se c'è una violazione della sicurezza, se il nome fornito non è valido o esiste già oppure se le proprietà fornite non sono valide. |
| DISPLAY *TYPE*(name) | Visualizza le proprietà dell'oggetto amministrato del tipo *`TYPE`*, collegate con il nome `name` nel contesto corrente. Non riesce se l'oggetto non esiste o se c'è una violazione della sicurezza. |
| DELETE *TYPE*(name) | Tenta di rimuovere l'oggetto amministrato del tipo *`TYPE`*, che ha il nome `name`, dal contesto corrente. Non riesce se l'oggetto non esiste o se c'è una violazione della sicurezza. |
| COPY *TYPE*(nameA) *TYPE*(nameB) | Effettua una copia dell'oggetto amministrato del tipo *`TYPE`*, che ha il nome `nameA`, assegnando alla copia il nome `nameB`. Il tutto si verifica nell'ambito del contesto corrente. Non riesce se l'oggetto da copiare non esiste, se un oggetto con il nome nameB esiste già oppure se c'è una violazione della sicurezza. |
| MOVE *TYPE*(nameA) *TYPE*(nameB) | Sposta (rinomina) l'oggetto del tipo *`TYPE`*, che ha il nome `nameA`, su `nameB`. Il tutto si verifica nell'ambito del contesto corrente. Non riesce se l'oggetto da spostare non esiste, se un oggetto con il nome nameB esiste già oppure se c'è una violazione della sicurezza. |

## Creazione di oggetti

Gli oggetti vengono creati e memorizzati in uno spazio dei nomi JNDI utilizzando la seguente sintassi dei comandi:

```
DEFINE TYPE(name) [property]*
```

In altre parole, il verb `DEFINE`, seguito da un riferimento dell'oggetto amministrato *`TYPE`*`(name)`, seguito da zero o più *proprietà* (vedere "Proprietà" a pagina 44).

### Considerazioni sull'assegnazione di nomi LDAP

Per memorizzare gli oggetti in un ambiente LDAP, i nomi devono essere conformi a determinate convenzioni. Una delle convenzioni è che i nomi dell'oggetto e del subcontesto devono includere un prefisso, ad esempio `cn=` (common name) o `ou=` (organizational unit).

Lo strumento di amministrazione semplifica l'utilizzo dei provider dei servizi LDAP consentendo all'utente di fare riferimento a nomi di oggetti e contesti senza un prefisso. Se non viene fornito un prefisso, lo strumento aggiunge automaticamente un prefisso predefinito, (al momento `cn=`) al nome fornito

La cosa viene illustrata nell'esempio che segue.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

Sebbene il nome di oggetto fornito (`testQueue`) non abbia un prefisso, lo strumento ne aggiunge automaticamente uno per garantire la conformità con la convenzione dei nomi LDAP. Analogamente, anche eseguendo il comando `DISPLAY Q(testQueue)`, il comando verrà aggiunto.

E' necessario configurare il server LDAP per memorizzare gli oggetti Java. Le informazioni che consentono di eseguire questa configurazione sono riportate nella "Appendice C. Definizione dello schema LDAP per la memorizzazione degli oggetti Java" a pagina 393.

## Proprietà

Una proprietà è composta da una coppia nome-valore nel formato:

```
PROPERTY_NAME(property_value)
```

I nomi delle proprietà non fanno distinzione tra lettere maiuscole e minuscole e sono limitati alla serie di nomi riconosciuti illustrati nella Tabella 10. In questa tabella sono anche riportati i valori delle proprietà validi per ciascuna proprietà.

*Tabella 10. Nomi di proprietà e valori validi*

| Proprietà | Formato breve | Valori validi (quelli predefiniti sono in grassetto) |
|---|---|---|
| DESCRIPTION | DESC | Qualsiasi stringa |
| TRANSPORT | TRAN | • **BIND** - Le connessioni utilizzano i binding MQSeries.<br>• CLIENT - Viene utilizzata la connessione client |
| CLIENTID | CID | Qualsiasi stringa |
| QMANAGER | QMGR | Qualsiasi stringa |
| HOSTNAME | HOST | Qualsiasi stringa |
| PORT | | Qualsiasi numero positivo intero |
| CHANNEL | CHAN | Qualsiasi stringa |
| CCSID | CCS | Qualsiasi numero positivo intero |
| RECEXIT | RCX | Qualsiasi stringa |
| RECEXITINIT | RCXI | Qualsiasi stringa |
| SECEXIT | SCX | Qualsiasi stringa |
| SECEXITINIT | SCXI | Qualsiasi stringa |
| SENDEXIT | SDX | Qualsiasi stringa |
| SENDXITINIT | SDXI | Qualsiasi stringa |
| TEMPMODEL | TM | Qualsiasi stringa |

*Tabella 10. Nomi di proprietà e valori validi (Continua)*

| Proprietà | Formato breve | Valori validi (quelli predefiniti sono in grassetto) |
|---|---|---|
| MSGRETENTION | MRET | • **Yes** - I messaggi non desiderati restano nella coda di input<br>• No - I messaggi non desiderati vengono gestiti in base alle relative opzioni di disposizione |
| BROKERVER | BVER | **V1** - L'unico valore attualmente consentito. |
| BROKERPUBQ | BPUB | Qualsiasi stringa (l'impostazione predefinita è **SYSTEM.BROKER.DEFAULT.STREAM**) |
| BROKERSUBQ | BSUB | Qualsiasi stringa (l'impostazione predefinita è **SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE**) |
| BROKERDURSUBQ | BDSUB | Qualsiasi stringa (l'impostazione predefinita è **SYSTEM.JMS.D.SUBSCRIPTION.QUEUE**) |
| BROKERCCSUBQ | CCSUB | Qualsiasi stringa (l'impostazione predefinita è **SYSTEM.JMS.ND.CC.SUBSCRIPTION.QUEUE**) |
| BROKERCCDSUBQ | CCDSUB | Qualsiasi stringa (l'impostazione predefinita è **SYSTEM.JMS.D.CC.SUBSCRIPTION.QUEUE**) |
| BROKERQMGR | BQM | Qualsiasi stringa |
| BROKERCONQ | BCON | Qualsiasi stringa |
| EXPIRY | EXP | • **APP** - La scadenza può essere definita dall'applicazione JMS.<br>• UNLIM - Nessuna scadenza.<br>• Qualsiasi numero intero positivo che rappresenti la scadenza in millisecondi. |
| PRIORITY | PRI | • **APP** - La priorità può essere definita dall'applicazione JMS.<br>• QDEF - La priorità prende il valore dell'impostazione predefinita della coda.<br>• Qualsiasi numero intero compreso nell'intervallo 0-9. |
| PERSISTENCE | PER | • **APP** - La persistenza può essere definita dall'applicazione JMS.<br>• QDEF - La persistenza prende il valore dell'impostazione predefinita della coda.<br>• PERS - I messaggi sono persistenti.<br>• NON - I messaggi non sono persistenti. |
| TARGCLIENT | TC | • **JMS** - La destinazione del messaggio è un'applicazione JMS.<br>• MQ - La destinazione del messaggio è un'applicazione MQSeries tradizionale non JMS. |
| ENCODING | ENC | Vedere "La proprietà ENCODING" a pagina 48 |
| QUEUE | QU | Qualsiasi stringa |
| TOPIC | TOP | Qualsiasi stringa |

Molte delle proprietà sono relative solo a uno specifico sottoinsieme dei tipi di oggetto. Tabella 11 a pagina 46mostra per ciascuna proprietà quali tipi di oggetto sono validi e fornisce una breve descrizione di ciascuna proprietà. I tipi di oggetto vengono identificati utilizzando la Parola chiave, per spiegazione dettagliata, consultare Tabella 8 a pagina 40.

*Tabella 11. Le combinazioni valide del tipo proprietà e oggetto*

| Proprietà | QCF | TCF | Q | T | WSQCF XAQCF | WSTCF XATCF | Descrizione |
|---|---|---|---|---|---|---|---|
| DESCRIPTION | Y | Y | Y | Y | Y | Y | Una descrizione dell'oggetto memorizzato |
| TRANSPORT | Y | Y | | | Y[1] | Y[1] | Se le connessioni utilizzeranno le MQ Bindings o una connessione client |
| CLIENTID | Y | Y | | | Y | Y | Un identificativo della stringa per il client |
| QMANAGER | Y | Y | Y | | Y | Y | Il nome del gestore code a cui connettersi |
| PORT | Y | Y | | | | | La porta sui cui il gestore code è in ascolto |
| HOSTNAME | Y | Y | | | | | Il nome dell'host su cui risiede il gestore code |
| CHANNEL | Y | Y | | | | | Il nome del canale della connessione client utilizzata |
| CCSID | Y | Y | Y | Y | | | L'ID coded-character-set da utilizzare sulle connessioni |
| RECEXIT | Y | Y | | | | | Il nome della classe completo del receive exit utilizzato |
| RECEXITINIT | Y | Y | | | | | Stringa di inizializzazione di receive exit |
| SECEXIT | Y | Y | | | | | Il nome della classe completo del security exit utilizzato |
| SECEXITINIT | Y | Y | | | | | Stringa di inizializzazione di security exit |
| SENDEXIT | Y | Y | | | | | Il nome della classe completo del send exit utilizzato |
| SENDEXITINIT | Y | Y | | | | | Stringa di inizializzazione di send exit |
| TEMPMODEL | Y | | | | Y | | Nome della coda modello da cui vengono create le code temporanee |
| MSGRETENTION | Y | | | | Y | | Se il consumer della connessione mantiene i messaggi non desiderati sulla coda di input |
| BROKERVER | | Y | | | | Y | La versione del broker utilizzata |
| BROKERPUBQ | | Y | | | | Y | Il nome della coda di input del broker (coda del flusso) |
| BROKERSUBQ | | Y | | | | Y | Il nome della coda da cui vengono recuperati i messaggi della sottoscrizione non durevole |
| BROKERDURSUBQ | | | | Y | | | Il nome della coda da cui vengono recuperati i messaggi della sottoscrizione durevole |
| BROKERCCSUBQ | | Y | | | | Y | Il nome della coda da cui vengono recuperati i messaggi della sottoscrizione non durevole per un ConnectionConsumer |

*Tabella 11. Le combinazioni valide del tipo proprietà e oggetto (Continua)*

| Proprietà | QCF | TCF | Q | T | WSQCF XAQCF | WSTCF XATCF | Descrizione |
|---|---|---|---|---|---|---|---|
| BROKERCCDSUBQ | | | | Y | | | Il nome della coda da cui vengono recuperati i messaggi della sottoscrizione durevole per un ConnectionConsumer |
| BROKERQMGR | | Y | | | | Y | Il gestore code su cui è in esecuzione il broker |
| BROKERCONQ | | Y | | | | Y | Nome della coda di controllo del broker |
| EXPIRY | | | Y | Y | | | Il periodo dopo il quale i messaggi in una destinazione scadono |
| PRIORITY | | | Y | Y | | | La priorità dei messaggi inviati a una destinazione |
| PERSISTENCE | | | Y | Y | | | La persistenza dei messaggi inviati a una destinazione |
| TARGCLIENT | | | Y | Y | | | Il campo indica se il formato RFH2 MQSeries è utilizzato per scambiare informazioni con le applicazioni di destinazione |
| ENCODING | | | Y | Y | | | Lo schema di codifica utilizzato per questa destinazione |
| QUEUE | | | Y | | | | Il nome sottostante della coda che rappresenta questa destinazione |
| TOPIC | | | | Y | | | Il nome sottostante dell'argomento che rappresenta questa destinazione |

**Note:**

1. Per gli oggetti WSTCF, WSQCF, XATCF e XAQCF, è consentito solo il tipo di trasporto BIND.
2. Nella "Appendice A. Associazione tra le proprietà dello strumento di amministrazione e le proprietà programmabili" a pagina 389 viene mostrata la relazione tra le proprietà impostate dallo strumento e le proprietà programmabili.
3. La proprietà TARGCLIENT indica se il formato RFH2 MQSeries viene utilizzato per scambiare informazioni con le applicazioni di destinazione.

   La costante MQJMS_CLIENT_JMS_COMPLIANT indica che il formato RFH2 viene utilizzato per inviare informazioni. Le applicazioni che utilizzano MQ JMS comprendono il formato RFH2. E' necessario impostare la costante MQJMS_CLIENT_JMS_COMPLIANT durante lo scambio di informazioni con un'applicazione MQ JMS di destinazione.

   La costante MQJMS_CLIENT_NONJMS_MQ indica che il formato RFH2 non viene utilizzato per l'invio di informazioni. Generalmente, questo valore viene utilizzato per un'applicazione MQSeries esistente (ossia, una che non tratta RFH2).

## Dipendenze delle proprietà

Alcune proprietà presentano delle dipendenze reciproche. Ciò significa che è inutile fornire una proprietà a meno che un'altra proprietà non venga impostata su un determinato valore. I due gruppi di proprietà specifici in cui ciò può accadere sono le proprietà Client e le stringhe di inizializzazione Exit.

**Proprietà Client**

Se la proprietà `TRANSPORT(CLIENT)` non è stata esplicitamente impostata su un'impostazione predefinita della connessione, il trasporto utilizzato sulle connessioni fornite è MQ Bindings. Di conseguenza, non è possibile configurare nessuna delle proprietà sull'impostazione predefinita di questa connessione. Esse sono:

- HOST
- PORT
- CHANNEL
- CCSID
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT

Se si tenta di impostare una qualsiasi di queste proprietà senza impostare la proprietà `TRANSPORT` su `CLIENT`, verrà visualizzato un errore.

**Stringhe di inizializzazione Exit**

Non è valido impostare nessuna stringa di inizializzazione dell'uscita, a meno che il nome dell'uscita corrispondente sia stato fornito. Le proprietà dell'inizializzazione dell'uscita sono:

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

Specificando, ad esempio, `RECEXITINIT(myString)` senza specificare `RECEXIT(some.exit.classname)`, verrà visualizzato un errore.

## La proprietà ENCODING

I valori validi che la proprietà `ENCODING` può prendere sono più complessi del resto delle proprietà. La proprietà di codifica viene creata da tre sub-proprietà:

| | |
|---|---|
| **codifica intera** | può essere normale o invertita |
| **codifica decimale** | può essere normale o invertita |
| **codifica a virgola mobile** | può essere IEEE normale, IEEE invertita o System/390 |

La proprietà `ENCODING` è espressa come stringa di tre caratteri con la seguente sintassi:

```
{N|R}{N|R}{N|R|3}
```

In questa stringa:

- `N` denota normale

- R denota invertita
- 3 denota System/390
- il primo carattere rappresenta la *codifica intera*
- il secondo carattere rappresenta la *codifica decimale*
- il terzo carattere rappresenta la *codifica a virgola mobile*

In questo modo viene quindi fornita una serie di dodici possibili valori della proprietà ENCODING.

Esiste un valore aggiuntivo, la stringa NATIVE, che imposta i valori di codifica appropriati per la piattaforma Java.

Gli esempi che seguono mostrano le combinazioni valide per ENCODING:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

## Condizioni di errore di esempio

In questa sezione vengono illustrati esempi delle condizioni di errore che potrebbero verificarsi durante la creazione di un oggetto.

**Proprietà sconosciuta**

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Impossibile creare un oggetto valido, verificare i parametri forniti
Proprietà sconosciuta: PIZZA
```

**Proprietà non valida per l'oggetto**

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Impossibile creare un oggetto valido, verificare i parametri forniti
Proprietà non valida per un QCF: PRI
```

**Tipo non valido per il valore della proprietà**

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Impossibile creare un oggetto valido, verificare i parametri forniti    Valore
non valido per la proprietà CCS: Inglese
```

**Valore della proprietà al di fuori dell'intervallo valido**

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Impossibile creare un oggetto valido, verificare i parametri forniti
Valore non valido per la proprietà PRI: 12
```

**Conflitto proprietà - client/binding**

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Impossibile creare un oggetto valido, verificare i parametri forniti
Proprietà non valida in questo contesto: Conflitto attributo client-binding
```

**Conflitto proprietà - inizializzazione Exit**

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Impossibile creare un oggetto valido, verificare i parametri forniti
Proprietà non valida in questo contesto: stringa ExitInit fornita senza la
stringa Exit
```

# Parte 2. Programmazione con MQ base Java

# Capitolo 6. Introduzione per i programmatori

In questo capitolo sono riportate informazioni destinate ai programmatori. Per istruzioni più dettagliate sulla scrittura dei programmi, fare riferimento al “Capitolo 7. Scrittura di programmi MQ base Java” a pagina 57.

## Vantaggi dell’utilizzo dell’interfaccia Java

L’interfaccia di programmazione delle MQSeries classi per Java rende disponibili i numerosi vantaggi di Java agli sviluppatori delle applicazioni di MQSeries:

- Il linguaggio di programmazione Java è **semplice da utilizzare**.

  Non sono necessari file di intestazione, puntatori, strutture, unioni e sovraccarico di operatori. Il debug e lo sviluppo dei programmi scritti in linguaggio Java è più semplice rispetto agli equivalenti in C e C++.
- Java è **orientato agli oggetti**.

  Le funzioni di orientamento agli oggetti di Java sono confrontabili con quelle del linguaggio C++, ma non c’è ereditarietà multipla. Al contrario, Java utilizza il concetto dell’interfaccia.
- Java è un linguaggio **distribuito**.

  Le librerie di classi Java contengono una libreria di routine per la gestione di protocolli TCP/IP, quali HTTP e FTP. I programmi Java possono accedere agli URL con la stessa facilità con cui accedono a un file system.
- Il linguaggio Java è **robusto**.

  Java pone molta enfasi sul controllo precoce dei possibili problemi, il controllo dinamico (runtime) e l’eliminazione di situazioni che possono potenzialmente causare degli errori. Java utilizza un concetto di riferimenti che eliminano la possibilità di sovrascrivere la memoria e danneggiare i dati.
- Il linguaggio Java è **protetto**.

  Java è destinato a un esecuzione in ambienti di rete o distribuiti, con una particolare attenzione sulla sicurezza. I programmi Java non possono sovraccaricare lo stack di runtime né danneggiare la memoria al di fuori dello spazio di elaborazione. Quando i programmi Java vengono scaricati da Internet, non possono nemmeno leggere o scrivere file locali.
- I programmi Java sono **portabili**.

  Non ci sono aspetti di ″dipendenza dall’implementazione″ della specifica Java. Il compilatore Java genera un formato di file a oggetti indipendente dall’architettura. Il codice compilato è eseguibile su molti processori, purché il sistema di runtime Java sia presente.

Se si scrive l’applicazione utilizzando MQSeries classi per Java, l’utente può eseguire il download dei codici di byte Java (chiamati*applet*) per il programma da Internet. Le applet scaricate possono quindi essere eseguite sui computer degli utenti che le hanno scaricate. Ciò significa che qualsiasi utente dotato di un accesso al server Web può caricare ed eseguire l’applicazione senza che sia necessario effettuare alcuna installazione sulla propria macchina.

Quando è richiesto un aggiornamento di un programma, viene aggiornata la copia sul server Web. Al successivo accesso all’applet, viene automaticamente ricevuta la versione più recente. In questo modo i costi relativi all’installazione e

all'aggiornamento delle applicazioni client tradizionali che coinvolgono un gran numero di PC possono essere ridotti in modo significativo.

Se si inserisce l'applet in un server Web accessibile al di fuori del firewall aziendale, qualsiasi utente su Internet può scaricare e utilizzare l'applicazione. Ciò significa che è possibile ottenere messaggi nel proprio sistema MQSeries da qualsiasi punto su Internet, aprendo così la strada alla creazione di un'intera nuova serie di servizi accessibili, supporto e applicazioni per il commercio elettronico su Internet.

## L'interfaccia delle MQSeries classi per Java

L'API MQSeries procedurale viene creata intorno ai seguenti verb:

```
MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX,
MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
```

Questi verb prendono tutti, in qualità di parametro, un handle dell'oggetto MQSeries su cui devono operare. Poiché Java è orientato agli oggetti, l'interfaccia di programmazione Java li riorganizza. Il programma è composto da un insieme di oggetti MQSeries, sui quali si agisce chiamando i metodi su questi oggetti, come nel seguente esempio.

Quando si utilizza l'interfaccia procedurale, ci si disconnette da un gestore code utilizzando la chiamata MQDISC(Hconn, CompCode, Reason), in cui *Hconn* è un handle del gestore code.

Nell'interfaccia Java, il gestore code è rappresentato da un oggetto della classe MQQueueManager. La disconnessione dal gestore code viene effettuata richiamando il metodo disconnect() su quella classe.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

## Java Development Kit

Prima di poter compilare qualsiasi applet o applicazione scritta, è necessario disporre dell'accesso JDK (Java Development Kit) relativo alla piattaforma di sviluppo. Il JDK contiene tutte le classi, variabili, costruttori e interfacce Java standard, da cui dipendono le classi MQSeries classi per Java. Contiene inoltre gli strumenti necessari per la compilazione e l'esecuzione deglle applet e dei programmi su ogni piattaforma supportata.

E' possibile eseguire il download IBM Developer Kits per Java dal IBM Software Download Catalog, disponibile sul World Wide Web all'indirizzo:

```
http://www.ibm.com/software/download
```

Inoltre, è possibile sviluppare le applicazioni utilizzando Developer Kit incluso con l'ambiente di sviluppo integrato IBM VisualAge perJava.

Per compilare le applicazioni Java sulle piattaforme iSeries & AS/400, è necessario installare prima:

- L' AS/400 Developer Kit per Java, 5769-JV1

- Il Qshell Interpreter, OS/400 (5769-SS1) Option 30

## Libreria delle classi delle MQSeries classi per Java

Le MQSeries classi per Java sono un insieme di classi Java che consentono alle applet ed alle applicazioni Java di interagire con MQSeries.

Sono fornite le seguenti classi:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

Le seguenti interfacce Java sono fornite:
- MQC
- MQPoolServicesEventListener
- MQReceiveExit
- MQSecurityExit
- MQSendExit

E' fornita anche l'implementazione delle interfacce Java indicate di seguito che non sono comunque destinate all'utilizzo da parte delle applicazioni:
- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

In Java, un *pacchetto* è un meccanismo per il raggruppamento i serie di classi correlate. Le classi e interfacce MQSeries sono fornite come un pacchetto Java denominato `com.ibm.mq`. Per includere il pacchetto delle MQSeries classi per Java nel proprio programma, aggiungere la seguente riga all'inizio del file di origine:

```
import com.ibm.mq.*;
```

## libreria delle classi delle MQ base Java

# Capitolo 7. Scrittura di programmi MQ base Java

Per utilizzare MQSeries classi per Java per accedere alle code MQSeries, è necessario scrivere dei programmi Java contenenti chiamate che consentono di eseguire operazioni di inserimento e di estrazione nelle code MQSeries. I programmi possono essere *applet* Java, *servlet* Java o *applicazioni* Java.

Questo capitolo fornisce informazioni utili per scrivere applet, servlet e applicazioni Java per interagire con i sistemi MQSeries. Per ulteriori informazioni sulle singole classi, consultare il "Capitolo 9. Le classi e le interfacce MQ base Java" a pagina 87.

## Scegliere tra applet e applicazioni

Scegliere se scrivere applet, servlet o applicazioni dipende dalla connessione che si desidera utilizzare e da dove si desidera eseguire i programmi.

Le differenze principali tra le applet e le applicazioni sono:

- Le applet vengono eseguite con un visualizzatore di applet oppure in un browser Web, le servlet vengono eseguite in un server delle applicazioni Web e le applicazioni vengono eseguite indipendentemente.
- Le applet possono essere scaricate da un server Web su una macchina browser Web, le applicazioni e le servlet no.
- Le applet vengono eseguite con le regole di protezione aggiuntive che limitano il loro funzionamento. Tali regole sono più restrittive di Java 1.2.

  Consultare "Appendice F. Utilizzo di MQ Java nelle applet con Java 1.2 o successiva" a pagina 407 per maggiori informazioni.

Sono valide le seguenti norme generali:

- Se si desidera eseguire i propri programmi da macchine in cui non è installato in locale MQSeries classi per Java, bisogna scrivere delle applet.
- La modalità di binding nativa di MQSeries classi per Java non supporta le applet. Pertanto, se si desidera utilizzare i propri programmi in tutte le modalità di connessione, inclusa la modalità di binding nativa, è necessario scrivere delle applet oppure delle applicazioni.

## Differenze tra le connessioni

La modalità di connessione che si desidera utilizzare influenza in una certa misura la programmazione per MQSeries classi per Java.

### Connessioni client

Quando MQSeries classi per Java viene utilizzato come un client, è simile al client C MQSeries ma presenta le seguenti differenze:

- Supporta solo TCP/IP.
- Non supporta le tabelle di connessione.
- Non legge le variabili di ambiente MQSeries in fase di avvio.

### Differenze tra le connessioni

- Le informazioni da memorizzare in una definizione di canale e nelle variabili di ambiente vengono memorizzate in una classe detta MQEnvironment. In alternativa, queste informazioni possono essere passate come parametri quando viene eseguita la connessione.
- Le condizioni di errore e di eccezione vengono scritte in un log specificato nella classe MQException. La destinazione dell'errore predefinito è la console Java.

I client MQSeries classi per Java non supportano l'istruzione MQBEGIN o i binding veloci.

Per informazioni generiche sui client MQSeries, consultare il manuale *MQSeries Clients*.

**Nota:** Quando si utilizza la connesssione VisiBroker, le impostazioni dell'ID utente e della password in MQEnvironment non vengono inoltrate nel server MQSeries. L'id utente effettivo è quello valido per il server IIOP.

## Modalità di binding

La modalità di binding di MQSeries classi per Java è diversa dalle modalità client in quanto:

- La maggior parte dei parametri forniti dalla classe MQEnvironment viene ignorata
- I binding supportano l'istruzione MQBEGIN ed i binding veloci nel gestore code di MQSeries

**Nota:** MQSeries per AS/400 non supporta l'utilizzo di MQBEGIN per avviare unità di lavoro globali coordinate dal gestore code.

## Definizione della connessione da utilizzare

La connessione è determinata dall'impostazione delle variabili nella classe MQEnvironment.

**MQEnvironment.properties**

Questa variabile può contenere le seguenti coppie chiave/valore:

- Per le connessioni client e binding:

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES
```

- Per le connessioni VisiBroker:

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_VISIBROKER
MQC.ORB_PROPERTY, orb
```

**MQEnvironment.hostname**

Impostare il valore di questa variabile nel seguente modo:

- Per le connessioni client, impostare questa variabile sul nome host del server MQSeriesal quale si desidera connettersi
- Per la modalità binding, impostare questa variabile su un valore nullo

---

# Frammenti di codice di esempio

Questa sezione contiene due frammenti di codice di esempio; Figura 1 a pagina 59 e Figura 2 a pagina 62. Ciascuno di questi frammenti utilizza una particolare connessione e contiene delle note per descrivere le modifiche necessarie per utilizzare delle connessioni alternative.

## Codice applet di esempio

Il seguente frammento di codice rappresenta una applet che utilizza una connessione TCP/IP per:

1. Stabilire una connessione con un Queue Manager
2. Inserire un messaggio in una SYSTEM.DEFAULT.LOCAL.QUEUE
3. Estrarre nuovamente il messaggio

```
// ============================================================================
//
// Materiale su licenza - Proprietà dell'IBM
//
// 5639-C34
//
// (c) Copyright IBM Corp. 1995,1999
//
// ============================================================================
// applet di esempio MQSeries Client per Java
//
// Questo esempio viene eseguito come una applet utilizzando il visualizzatore di applet ed il file HTML,
// utilizzando il comando :-
//            appletviewer MQSample.html
// L'output è diretto alla riga comandi, NON alla finestra del visualizzatore di applet.
//
// Nota. Se si riceve un errore MQSeries 2 con codice ragione 2059 e si è sicuri che
// le impostazioni di MQSeries e di TCP/IP sono corrette,
// fare clic sulla selezione "Applet" nella finestra dell'Applet Viewer,
// selezionare le proprietà e modificare l'impostazione relativa all'accesso alla rete per renderla non limitata.
import com.ibm.mq.*;            // Includere il pacchetto MQSeries classi per Java

public class MQSample extends java.applet.Applet
{

  private String hostname = "nome__host";      // definire il nome dell'host
                                                  // con cui stabilire la connessione
  private String channel  = "canale_server";     // definire il nome del canale
                                                  // che verrà utilizzato dal client
                                                  // Nota. si presuppone che il server MQSeries
                                                  // sia in ascolto sulla porta
                                                  // TCP/IP predefinita 1414
  private String qManager = "gestore_code";      // definire il nome dell'oggetto
                                                  // Queue Manager con cui stabilire
                                                  // la connessione.

  private MQQueueManager qMgr;                    // definire un oggetto Queue Manager

  // Quando viene eseguita la chiamata della classe, viene prima eseguita questa inizializzazione.

  public void init()
  {
     // Impostare l'ambiente MQSeries
     MQEnvironment.hostname = nomehost;           // E' possibile inserire la stringa
                                                  // nomehost & canale
     MQEnvironment.channel  = canale;             // direttamente qua.

     MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY,//Impostare la connessione TCP/IP
                                  MQC.TRANSPORT_MQSERIES);//o server

  } // fine unità
```

*Figura 1. Applet di esempio MQSeries classi per Java (Numero 1 di 3)*

## Codice di esempio

```
public void start()
{

  try {
    // Creare una connessione al gestore code
    qMgr = new MQQueueManager(qManager);

    // Impostare le opzioni sulla coda che si desidera aprire...
    // Nota. Tutte le opzioni MQSeries hanno come prefisso MQC in Java.
     int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                       MQC.MQOO_OUTPUT ;
    // Specificare adesso la coda che si desidera aprire e le opzioni di apertura...

    MQQueue system_default_local_queue =
            qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                             openOptions);

    // Definire un messaggio MQSeries semplice e scrivere del testo in formato UTF..

    MQMessage hello_world = new MQMessage();
    hello_world.writeUTF("Salve a tutti!");

    // specificare le opzioni di messaggio...

    MQPutMessageOptions pmo = new MQPutMessageOptions();  // accettare i valori predefiniti,
                                                          // uguale a
                                                          // MQPMO_DEFAULT
                                                          //
    // inserire il messaggio nella coda

    system_default_local_queue.put(hello_world,pmo);

    // estrarre nuovamente il messaggio...
    // Definire prima un buffer dei messaggi MQSeries in cui ricevere il messaggio..

    MQMessage retrievedMessage = new MQMessage();
    retrievedMessage.messageId = hello_world.messageId;

    // Impostare le opzioni di estrazione del messaggio..

    MQGetMessageOptions gmo = new MQGetMessageOptions();  //  accettare i valori predefiniti
                                                          // uguale a
                                                          // MQGMO_DEFAULT
    // estrarre il messaggio dalla coda..

    system_default_local_queue.get(retrievedMessage, gmo);

    // E provare di disporre del messaggio visualizzando il testo di messaggio UTF

    String msgText = retrievedMessage.readUTF();
    System.out.println("Il messaggio è: " + msgText);

    // Chiudere la coda

    system_default_local_queue.close();

    // Annullare la connessione al gestore code

    qMgr.disconnect();

  }

  // Se si è verificata una condizione di errore, tentare di identificarne la causa.
  // Cos'è un errore MQSeries?
```

*Figura 1. Applet di esempio MQSeries classi per Java (Numero 2 di 3)*

```
    catch (MQException ex)
    {
      System.out.println("Si è verificato un errore MQSeries : Codice di completamento " +
                         ex.completionCode +
                         " Codice motivo" + ex.reasonCode);
    }
    // Cos'è un errore di spazio buffer Java?
    catch (java.io.IOException ex)
    {
      System.out.println("Si è verificato un errore in fase di scrittura nel
      buffer dei messaggi: " + ex);
    }

  } // fine dell'inizio

} // fine dell'esempio
```

*Figura 1. Applet di esempio MQSeries classi per Java (Numero 3 di 3)*

## Modifica della connessione per utilizzare VisiBroker per Java

Modificare la riga:

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_MQSERIES);
```

in:

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_VISIBROKER);
```

ed aggiungere le seguenti righe per avviare l'ORB (object request broker):

```
ORB orb=ORB.init(this,null);
MQEnvironment.properties.put(MQC.ORB_PROPERTY,orb);
```

Occorre inoltre aggiungere la seguente istruzione di importazione all'inizio del file:

```
import org.omg.CORBA.ORB;
```

Non occorre specificare il numero di porta o il canale se si utilizza VisiBroker.

## Codice applicazione di esempio

Il seguente frammento di codice presenta una semplice applicazione che utilizza la modalità binding per:

1. Stabilire una connessione con un Queue Manager
2. Inserire un messaggio in una SYSTEM.DEFAULT.LOCAL.QUEUE
3. Estrarre nuovamente il messaggio

```
// ============================================================================
// Materiale su licenza - Proprietà dell'IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995,1999
// ============================================================================
// Applicazione di esempio delle classi MQSeries per Java
//
// Questo esempio viene eseguito come un'applicazione Java utilizzando il comando :- java MQSample

import com.ibm.mq.*;            // Includere il pacchetto delle classi MQSeries per Java


public class MQSample
{
  private String qManager = "gestore_code";      // definire il nome dell'oggetto
                                                 // Queue Manager con cui stabilire la connessione.
  private MQQueueManager qMgr;                   // definire un oggetto
                                                 // Queue Manager
  public static void main(String args[]) {
     new MQSample();
  }

  public MQSample() {
   try {

      // Creare una connessione al gestore code

      qMgr = new MQQueueManager(qManager);

      // Impostare le opzioni sulla coda che si desidera aprire...
      // Nota. Tutte le opzioni MQSeries hanno come prefisso MQC in Java.

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;

      // Specificare adesso la coda che si desidera aprire...
      // e le opzioni di apertura...

      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);

      // Definire un messaggio MQSeries semplice e scrivere del testo in formato UTF..

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Salve a tutti!");

      // specificare le opzioni di messaggio...

      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accettare i valori predefiniti,
                                                           // uguale a MQPMO_DEFAULT
```

*Figura 2. Applicazione di esempio MQSeries classi per Java (Numero 1 di 2)*

```
     // inserire il messaggio nella coda

      system_default_local_queue.put(hello_world,pmo);

      // estrarre nuovamente il messaggio...
      // Definire prima un buffer dei messaggi MQSeries in cui ricevere  il messaggio..

      MQMessage retrievedMessage = new MQMessage();
      retrievedMessage.messageId = hello_world.messageId;

      // Impostare le opzioni di estrazione del messaggio...

      MQGetMessageOptions gmo = new MQGetMessageOptions();  //  accettare i valori predefiniti
                                                            // uguale a MQGMO_DEFAULT
      // estrarre il messaggio dalla coda...

      system_default_local_queue.get(retrievedMessage, gmo);

      // E provare di disporre del messaggio visualizzando il testo di messaggio UTF

      String msgText = retrievedMessage.readUTF();
      System.out.println("Il messaggio è: " + msgText);
      // Chiudere la coda...
      system_default_local_queue.close();
      // Annullare la connessione al gestore code

      qMgr.disconnect();
    }
      // Se si è verificata una condizione di errore, tentare di identificarne la causa
      // Cos'è un errore MQSeries?
    catch (MQException ex)
    {
      System.out.println("Si è verificato un errore MQSeries : Codice completamento " +
                         ex.completionCode + " Codice motivo " + ex.reasonCode);
    }
      // Cos'è un errore di spazio buffer Java?
    catch (java.io.IOException ex)
    {
      System.out.println("Si è verificato un errore in fase di scrittura nel buffer dei messaggi: " + ex);
    }
  }
} // fine dell'esempio
```

*Figura 2. Applicazione di esempio MQSeries classi per Java (Numero 2 di 2)*

# Operazioni sui gestori code

Questa sezione descrive come stabilire connessioni con un gestore code e come annullarle, utilizzando MQSeries classi per Java.

## Impostazione dell'ambiente MQSeries

**Nota:** Eseguire questa procedura non è necessario se si utilizza MQSeries classi per Java in modalità binding. In questo caso, andare direttamente alla sezione "Connessione ad un Queue Manager". Prima di utilizzare la connessione client per stabilire una connessione a un gestore code, è necessario impostare la classe MQEnvironment.

I client MQSeries basati sul linguaggio ″C″ dipendono dalle variabili di ambiente per controllare il comportamento della chiamata MQCONN. Poiché le applet Java non dispongono dell'accesso alle variabili di ambiente, l'interfaccia di programmazione Java comprende una classe MQEnvironment. Questa classe consente di specificare i seguenti dettagli che devono essere utilizzati durante i tentativi di stabilimento di connessioni:

- Nome canale
- Nome host
- Numero porta
- ID utente
- Password

Per specificare il nome canale ed il nome host, utilizzare il seguente codice:

```
MQEnvironment.hostname = "dominio.host.com";
MQEnvironment.channel  = "canale.client.java";
```

Questo codice equivale ad un'impostazione della variabile di ambiente MQSERVER su:

```
"java.client.channel/TCP/dominio.host.com".
```

Per impostazione predefinita, i client Java tentano di stabilire connessioni con un listener MQSeries sulla porta 1414. Per specificare una porta differente, utilizzare il codice:

```
MQEnvironment.port = nnnn;
```

L'ID utente e la password assumono per impostazione predefinita dei valori nulli. Per specificare un ID utente oppure una password non nulli, utilizzare il codice:

```
MQEnvironment.userID  = "uid";  // equivalente alla variabile di ambiente MQ_USER_ID
MQEnvironment.password = "pwd";  // equivalente alla variabile di ambiente MQ_PASSWORD
```

**Nota:** Se si sta impostando una connessione utilizzando VisiBroker per Java, consultare la sezione "Modifica della connessione per utilizzare VisiBroker per Java" a pagina 61.

## Connessione ad un Queue Manager

Si è adesso pronti a stabilire una connessione a un gestore code creando una nuova istanza della classe MQQueueManager:

```
MQQueueManager
queueManager = new MQQueueManager("NomeqMgr");
```

Per annullare una connessione a un gestore code, richiamare il metodo () sul Queue Manager:

```
queueManager.disconnect();
```

Se si richiama il metodo di annullamento della connessione, tutte le code e tutti i processi aperti cui si è avuto accesso tramite il Queue Manager verranno chiusi. Tuttavia, è buona norma di programmazione chiudere esplicitamente queste risorse dopo aver finito di utilizzarle. Per eseguire quest'operazione, utilizzare il metodo close().

I metodi commit() e backout() su un gestore code sostituiscono le chiamate MQCMIT e MQBACK utilizzate con l'interfaccia procedurale.

## Accesso alle code e ai processi

Per accedere alle code e ai processi, utilizzare la classe MQQueueManager. La MQOD (struttura del descrittore degli oggetti) è compressa nei parametri di questi metodi. Ad esempio, per aprire una coda su un gestore code ″queueManager″, utilizzare il seguente codice:

```
MQQueue queue = queueManager.accessQueue("Nomecoda",
                                         MQC.MQOO_OUTPUT,
                                         "NomeqMgr",
                                         "Nomecodadinamica",
                                         "IdUtentealt");
```

Il parametro *options* è uguale al parametro Options nella chiamata MQOPEN.

Il metodo accessQueue restituisce un nuovo oggetto della classe MQQueue.

Dopo aver finito di utilizzare la coda, utilizzare il metodo close() per chiuderla, come nel seguente esempio:

```
queue.close();
```

Con MQSeries classi per Java, è anche possibile creare una coda utilizzando il costruttore MQQueue. I parametri sono esattamente identici per il metodo accessQueue; è presente solo un ulteriore parametro, Queue Manager. Ad esempio:

```
MQQueue queue = new MQQueue(queueManager,
                            "Nomecoda",
                            MQC.MQOO_OUTPUT,
                            "NomeqMgr",
                            "Nomecodadinamica",
                            "IdUtentealt");
```

Creare un oggetto coda in questo modo consente di scrivere le proprie classi di MQQueue.

Per accedere ad un processo, utilizzare il metodo accessProcess invece di accessQueue. Questo metodo non dispone di un parametro *dynamic queue name*, poiché esso non è valido per i processi.

Il metodo accessProcess restituisce un nuovo oggetto della classe MQProcess.

Dopo aver finito di utilizzare l'oggetto processo, utilizzare il metodo close() per chiuderlo, come nel seguente esempio:

```
process.close();
```

Con MQSeries classi per Java, è anche possibile creare un processo utilizzando il costruttore MQProcess. I parametri sono esattamente identici per il metodo

accessProcess; è presente solo un ulteriore parametro, Queue Manager. Creare un oggetto processo in questo modo consente di scrivere le proprie classi di MQProcess.

## Gestione dei messaggi

Inserire i messaggi nelle code utilizzando il metodo () della classe MQQueue. Estrarre i messaggi dalle code utilizzando il metodo get() della classe MQQueue. Diversamente dall'interfaccia procedurale, dove MQPUT e MQGET eseguono inserimenti ed estrazioni di matrici di byte, il linguaggio di programmazione Java inserisce ed estrarre istanze della classe MQMessage. La classe MQMessage incapsula il buffer di dati che contiene gli effettivi dati del messaggio è tutti i parametri MQMD (message descriptor) che lo descrivono.

Per creare un nuovo messaggio, creare una nuova istanza della classe MQMessage ed utilizzare i metodi writeXXX per inserire i dati nel buffer dei messaggi.

Quando viene creata la nuova istanza di messaggio, tutti i parametri MQMD vengono automaticamente impostati sui loro valori predefiniti, come definito nel manuale *MQSeries Application Programming Reference*. Il metodo put() di MQQueue prende inoltre un'istanza della classe MQPutMessageOptions come un parametro. Questa classe rappresenta la struttura MQPMO. Il seguente esempio crea un messaggio e lo inserisce in una coda:

```
// Creare un nuovo messaggio che contiene la mia età seguita dal mio nome
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Wendy Ling";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Utilizzare le opzioni di inserimento messaggio predefinite...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// inserire il messaggio!
queue.put(myMessage,pmo);
```

Il metodo get() di MQQueue restituisce una nuova istanza di MQMessage, che rappresenta il messaggio appena preso dalla coda. Prende anche un'istanza della classe MQGetMessageOptions come parametro. Questa classe rappresenta la struttura MQGMO.

Non è necessario specificare una dimensione massima del messaggio in quanto il metodo get() regola automaticamente le dimensioni del buffer interno per adattarsi al messaggio in arrivo. Utilizzare i metodi readXXX della classe MQMessage per accedere ai dati nel messaggio restituito.

Nell'esempio che segue viene illustrato come richiamare un messaggio da una coda:

```
// Richiamare un messaggio dalla coda
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // ha valori predefiniti

// Estrarre i dati del messaggio
int age = theMessage.readInt();
```

```
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

E' possibile modificare il formato numerico che i metodi di lettura e scrittura utilizzano impostando la variabile del membro *encoding*.

E' possibile modificare il set di caratteri da utilizzare per la lettura e scrittura di stringhe impostando la variabile del metodo *characterSet*.

Consultare la sezione "MQMessage" a pagina 113 per ulteriori dettagli.

**Nota:** Il metodo writeUTF() di MQMessage codifica automaticamente la lunghezza della stringa oltre che i byte Unicode che contiene. Quando il messaggio verrà letto da un altro programma Java utilizzando readUTF(), questo sarà il modo più semplice per l'invio di informazioni sulla stringa.

## Gestione degli errori

I metodi nell'interfaccia Java non restituiscono un codice di completamento e un codice motivo. Al contrario, producono un errore ogni volta che il codice di completamento e il codice motivo ottenuti da una chiamata MQSeries non sono entrambi pari a zero. Ciò semplifica la logica del programma e non è quindi necessario verificare i codici restituiti dopo ogni chiamata a MQSeries. E' possibile decidere in quali punti del programma affrontare la possibilità di un errore. In questi punti è possibile circondare il codice con blocchi 'try' e 'catch', come viene illustrato nell'esempio che segue:

```
try {
myQueue.put(messageA,putMessageOptionsA);
myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
// Questo blocco di codice viene eseguito solo se
// uno dei due metodi put determina un codice di completamento
// o un codice motivo pari a un valore diverso da zero.
System.out.println("Si è verificato un errore durante l'operazione put:" +
                           "CC = " + ex.completionCode +
                           "RC = " + ex.reasonCode);
}
```

I codici di motivo della chiamata MQSeries riportati nelle eccezioni Java sono documentati nel capitolo chiamato "Codici di ritorno" nel manuale *MQSeries Application Programming Reference*.

## Richiamo e impostazione dei valori degli attributi

Per molti degli attributi più comuni, le classi MQManagedObject, MQQueue, MQProcess e MQQueueManager contengono i metodi getXXX() e setXXX() che consentono di richiamare e impostare i valori dei relativi attributi. Per MQQueue, i metodi funzionano solo se si specificano i flag "inquire" e "set" appropriati quando si apre la coda.

Per gli attributi meno comuni, le classi MQQueueManager, MQQueue e MQProcess ereditano tutte da una classe denominata MQManagedObject. Tale classe definisce le interfacce inquire() e set().

### Utilizzo dei valori degli attributi

Quando si crea un nuovo oggetto del gestore code utilizzando l'operatore *new*, viene automaticamente aperto per "inquiry". Quando si utilizza il metodo accessProcess() per accedere all'oggetto di un processo, l'oggetto in questione viene aperto automaticamente per "inquiry". Quando si utilizza il metodo accessQueue() per accedere ad un oggetto della coda, l'oggetto in questione *non* viene aperto automaticamente per entrambe le operazioni " inquire" o "set". Ciò accade in quanto l'aggiunta di queste opzioni automaticamente può causare problemi con alcuni tipi di code remote. Per utilizzare i metodi inquire, set, getXXX e setXXX su una coda, è necessario specificare i flag "inquire" e "set" appropriati nel parametro openOptions del metodo del accessQueue().

I metodi inquire e set prendono tre parametri:
- selectors array
- intAttrs array
- charAttrs array

Non sono necessari i parametri SelectorCount, IntAttrCount e CharAttrLength trovati in MQINQ, in quanto la lunghezza di una matrice in Java è sempre conosciuta. Nell'esempio che segue viene illustrato come effettuare una richiesta su una coda:

```
// richiesta su una coda
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("Priorità predefinita = " + intAttrs[0]);
System.out.println("Descrizione : " + new String(charAttrs,0));
```

## Programmi a più thread

I programmi a più thread sono difficili da evitare in Java. Si consideri un programma semplice che si connette a un gestore code e apre all'avvio una coda. Il programma visualizza un solo pulsante sullo schermo. Quando un utente preme quel pulsante, il programma prende un messaggio dalla coda.

L'ambiente runtime Javaè interamente multithreaded. Pertanto l'inizializzazione dell'applicazione si verifica su un thread e il codice da eseguire in risposta alla pressione del pulsante viene eseguito in un thread separato, vale a dire il thread dell'interfaccia utente.

Con il client "C" basato su MQSeries, ciò potrebbe causare un problema in quanto gli handle non possono essere condivisi su thread multipli. MQSeries classi per Java risolvono questa limitazione consentendo a un oggetto del gestore code (e agli oggetti della coda e del processo associato) di essere condiviso su più thread.

L'implementazione MQSeries classi per Java assicura che, per una determinata connessione (istanza dell'oggetto MQQueueManager), tutto l'accesso al gestore code MQSeries di destinazione sia sincronizzato. Pertanto, un thread che desidera emettere una chiamata a un gestore code è bloccato fino al completamento delle altre chiamate in corso relative a quella connessione. Se si richiede l'accesso

simultaneo allo stesso gestore code da parte di thread multipli all'interno del programma, creare un nuovo oggetto MQQueueManager per ciascun thread che richiede un accesso simultaneo. Questa operazione equivale all'esecuzione di una chiamata MQCONN separata per ciascun thread.

# Scrittura delle uscite utente (user exit)

Le MQSeries classi per Java consentono di fornire uscite di invio, ricezione e sicurezza personalizzate.

Per implementare un'uscita, viene definita una nuova classe Java che implementa l'interfaccia appropriata. Tre interfacce di uscita sono definite nel pacchetto MQSeries:
- MQSendExit
- MQReceiveExit
- MQSecurityExit

Nell'esempio che segue viene definita una classe che le implementa tutte e tre:

```
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {

  // Questo metodo proviene dall'uscita invio
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // riempire il corpo dell'uscita invio in questo punto
  }


  // Questo metodo proviene dall'uscita ricezione
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
     // riempire il corpo dell'uscita ricezione in questo punto
  }


  // Questo metodo proviene dall'uscita sicurezza
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
     // riempire il corpo dell'uscita sicurezza in questo punto
  }

}
```

A ogni uscita viene trasmessa un'istanza dell'oggetto MQChannelExit e MQChannelDefinition. Tali oggetti rappresentano le strutture MQCXP e MQCD definite nell'interfaccia procedurale.

Per un'uscita di invio, il parametro *agentBuffer* contiene i dati che ci si accinge a inviare. Per un'uscita di ricezione o di sicurezza, il parametro *agentBuffer* contiene i dati appena ricevuti. Non è necessario un parametro della lunghezza in quanto l'espressione agentBuffer.length indica la lunghezza della matrice.

Per le uscite di invio e di sicurezza il codice di uscita dovrebbe restituire la matrice di byte che si desidera inviare al server. Per un'uscita di ricezione il codice di uscita dovrebbe restituire i dati modificati che si desidera che le MQSeries classi per Java interpretino.

Il corpo dell'uscita più semplice possibile è:

```
{
  return agentBuffer;
}
```

Se il programma deve essere eseguito come applet Java scaricata, le limitazioni di sicurezza applicate indicano che non è possibile leggere o scrivere nessun file locale. Se l'uscita necessita di un file di configurazione, è possibile inserire il file sul Web e utilizzare la classe java.net.URL per scaricarlo ed esaminarne il contenuto.

## Pool di connessioni

MQSeries classi per Java Versione 5.2 fornisce un supporto aggiuntivo per le applicazioni che gestiscono connessioni multiple sui gestori code MQSeries. Quando una connessione non è più necessaria, anziché distruggerla, può essere inserita in un pool e riutilizzata in un secondo momento. In questo modo si ottiene un sostanziale miglioramento delle prestazioni di applicazioni e middleware che si connette in modo seriale a gestori code arbitrari.

MQSeries 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 pool è attivo, quando MQ base Java costruisce un oggetto MQQueueManager, effettua una ricerca in questo pool predefinito e riutilizza tutte le connessioni adatte. Quando si verifica una chiamata MQQueueManager.disconnect(), la connessione sottostante viene restituita al pool.

In alternativa le applicazioni possono costruire un pool di connessioni MQSimpleConnectionManager per un particolare uso. L'applicazione può quindi specificare il pool durante la costruzione di un oggetto MQQueueManager oppure passare quel pool MQEnvironment perché possa essere utilizzato come pool di connessione predefinito.

Inoltre, MQ base Java fornisce un'implementazione parziale di Java 2 Platform Enterprise Edition (J2EE) Connector Architecture. Le applicazioni in esecuzione su una JVM Java 2 v1.3 con JAAS 1.0 (Java Authentication and Authorization Service) possono fornire il proprio pool di connessioni implementando l'interfaccia **javax.resource.spi.ConnectionManager**. Questa interfaccia può essere specificata sul costruttore MQQueueManager o specificata come pool di connessioni predefinito.

### Controllo del pool di connessioni predefinito

Si consideri 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]);
            :
            : (eseguire qualche operazione con qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 prende una lista dei gestori code locali dalla riga comandi, si connette a ciascuno di essi ed esegue alcune operazioni. Tuttavia, quando la riga comandi

elenca molte volte lo stesso gestore code, è più efficiente connettersi solo una volta e riutilizzare più volte quella connessione.

MQ base Java fornisce un pool di connessioni predefinito che può essere utilizzato per questo scopo. Per abilitare il pool, utilizzare uno dei metodi MQEnvironment.addConnectionPoolToken(). Per disabilitare il pool, utilizzare MQEnvironment.removeConnectionPoolToken().

L'applicazione di esempio che segue, 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]);
            :
            : (eseguire qualche operazione con qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

La prima riga in grassetto attiva il pool di connessioni predefinito, registrando un oggetto MQPoolToken con MQEnvironment.

Il costruttore MQQueueManager ricerca a questo punto nel pool una connessione appropriata e crea solo una connessione al gestore code se non è in grado di trovarne una esistente. La chiamata qmgr.disconnect() restituisce la connessione al pool per un riutilizzo successivo. Queste chiamate dell'API sono le stesse dell'applicazione di esempio MQApp1.

La seconda riga evidenziata disattiva il pool di connessioni predefinito, che distrugge qualsiasi connessione del gestore code memorizzata nel pool. Questo è importante in quanto se si facesse altrimenti l'applicazione terminerebbe con una serie di connessioni del gestore code attive nel pool. Tale situazione causerebbe degli errori che apparirebbero nel file di registrazione del gestore code.

Il pool di connessioni predefinito memorizza un massimo di dieci connessioni inutilizzate e le mantiene attive per un massimo di cinque minuti. L'applicazione può comunque modificare questa impostazione. Per informazioni più dettagliate a riguardo, consultare la sezione "Fornitura di un pool di connessioni differente" a pagina 75).

Anziché utilizzare MQEnvironment per fornire un MQPoolToken, l'applicazione ne può creare uno personalizzato:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Alcuni fornitori di applicazioni e middleware possono fornire sottoclassi di MQPoolToken per trasmettere informazioni a un pool di connessioni personalizzato. Esse possono essere costruite e trasmesse a

addConnectionPoolToken() in questo modo per far sì che al pool di connessioni possano essere trasmesse informazioni supplementari.

## Il pool di connessioni predefinito e componenti multipli

MQEnvironment conserva una serie statica di oggetti MQPoolToken registrati. Per aggiungere o rimuovere oggetti MQPoolTokens da questa serie, utilizzare i seguenti metodi:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Un'applicazione può essere composta da molti componenti che esistono indipendentemente ed eseguire il lavoro utilizzando un gestore code. In un'applicazione del genere, ogni componente dovrebbe aggiungere un MQPoolToken alla serie di MQEnvironment per tutta la sua durata.

Ad esempio, l'applicazione di esempio MQApp3 crea dieci thread ed avvia ciascuno di essi. Ogni thread registra il proprio MQPoolToken, attende un determinato lasso di tempo, quindi si connette al gestore code. Dopo la disconnessione del thread, rimuove il proprio MQPoolToken.

Il pool di connessioni predefinito resta attivo mentre c'è almeno un token nella serie di MQPoolToken, per cui resta attivo per la durata dell'applicazione. L'applicazione non deve mantenere un oggetto principale nel controllo complessivo 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");
            :
            : (eseguire qualche operazione con qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Si è verificato un errore");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

## Fornitura di un pool di connessioni differente

In questa sezione viene descritto come utilizzare la classe **com.ibm.mq.MQSimpleConnectionManager** per fornire un diverso pool di connessioni. Questa classe fornisce funzioni di base per il pool di connessioni e le applicazioni possono utilizzarla per personalizzare il comportamento del pool.

Una volta avviato, un oggetto MQSimpleConnectionManager può essere specificato sul costruttore MQQueueManager. MQSimpleConnectionManager quindi gestisce la connessione alla base del MQQueueManager costruito. Se MQSimpleConnectionManager contiene una connessione del pool adatta, la connessione verrà utilizzata e restituita a MQSimpleConnectionManager dopo una chiamata MQQueueManager.disconnect().

Il frammento di codice che segue dimostra questo comportamento:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (eseguire qualche operazione con qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (eseguire qualche operazione con qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

La connessione simulata durante il primo costruttore MQQueueManager è memorizzata in myConnMan dopo la chiamata qmgr.disconnect(). La connessione viene quindi riutilizzata durante la seconda chiamata al costruttore MQQueueManager.

La seconda riga abilita MQSimpleConnectionManager. L'ultima riga disabilita MQSimpleConnectionManager, distruggendo tutte le connessioni conservate nel pool. Un oggetto MQSimpleConnectionManager è, in base all'impostazione predefinita, in MODE_AUTO, di cui viene riportata una descrizione più avanti in questo capitolo.

Un oggetto MQSimpleConnectionManager assegna le connessioni in base al concetto dell'utilizzo più recente e le distrugge sulla base dell'utilizzo meno recente. In base all'impostazione predefinita, una connessione viene distrutta se non è stata utilizzata per cinque minuti, o se ci sono più di dieci connessioni inutilizzate nel pool. E' possibile modificare questi valori utilizzando:

- MQSimpleConnectionManager.setTimeout()
- MQSimpleConnectionManager.setHighThreshold()

E' anche possibile impostare un MQSimpleConnectionManager per un utilizzo come pool di connessioni predefinito, da usare quando nessun Connection Manager viene fornito sul costruttore MQQueueManager.

L'applicazione che segue 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.setHighThreshold(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

Le righe in grassetto impostano un MQSimpleConnectionManager. Viene impostato quanto segue:

- distruzione delle connessioni che non sono state utilizzate per un'ora
- limitazione del numero di connessioni inutilizzate conservate nel pool fino a 50
- MODE_AUTO (in realtà l'impostazione predefinita). Ciò significa che il pool è attivo solo se è il gestore delle connessioni predefinito e che c'è almeno un token nella serie di MQPoolTokens conservate da MQEnvironment.

Il nuovo MQSimpleConnectionManager viene quindi impostato come gestore connessioni predefinito.

Nell'ultima riga l'applicazione richiama MQApp3.main(), Tale applicazione esegue un certo numero di thread, dove ciascun thread utilizza MQSeries indipendentemente. Questi thread utilizzeranno myConnMan quando simulano le connessioni.

## Fornitura del proprio ConnectionManager

Su Java 2 v1.3, con JAAS 1.0 installato, i fornitori di applicazioni e middleware possono fornire implementazioni alternative di pool di connessioni. MQ base Java fornisce un'implementazione parziale di J2EE Connector Architecture. Le implementazioni di **javax.resource.spi.ConnectionManager** possono essere utilizzate come gestore connessioni predefinito o essere specificate sul costruttore MQQueueManager.

MQ base Java è conforme al contratto del gestore connessioni di J2EE Connector Architecture. Si consiglia di leggere questa sezione unitamente al contratto di Connection Management di J2EE Connector Architecture nel sito Web della Sun all'indirizzo `http://java.sun.com`).

L'interfaccia di ConnectionManager definisce un solo metodo:

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

Il costruttore MQQueueManager chiama allocateConnection sul ConnectionManager appropriato. Trasmette le implementazioni appropriate di ManagedConnectionFactory e ConnectionRequestInfo come parametri per descrivere la connessione richiesta.

Il ConnectionManager ricerca nel pool un oggetto javax.resource.spi.ManagedConnection creato con oggetti identici

ManagedConnectionFactory e ConnectionRequestInfo. Se ConnectionManager trova oggetti ManagedConnection adatti, crea un java.util.Set contenente il ManagedConnections candidato. Quindi il ConnectionManager effettua la seguente chiamata:

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

L'implementazione MQSeries di ManagedConnectionFactory ignora il parametro dell'argomento. Questo metodo seleziona e restituisce un ManagedConnection adatto dalla serie o restituisce null se non trova un ManagedConnection adatto. Se nel pool non è disponibile un ManagedConnection adatto, il ConnectionManager ne può creare uno utilizzando:

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

Il parametro viene nuovamente ignorato. Questo metodo connette un gestore code MQSeries e restituisce un'implementazione di javax.resource.spi.ManagedConnection che rappresenta la connessione appena simulata. Dopo che ConnectionManager ha ottenuto un ManagedConnection (dal pool o appena creato), crea un nuovo handle della connessione utilizzando:

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

Questo handle della connessione può essere restituito da allocateConnection().

Un ConnectionManager dovrebbe registrare un interesse nel ManagedConnection attraverso:

```
mc.addConnectionEventListener()
```

Al ConnectionEventListener viene segnalato se si verifica un errore grave sulla connessione oppure quando viene chiamato il metodo MQQueueManager.disconnect(). Quando MQQueueManager.disconnect() viene chiamato, il ConnectionEventListener può fare una delle seguenti cose:

- reimpostare il ManagedConnection mediante la chiamata mc.cleanup(), quindi restituire il ManagedConnection al pool
- distruggere il ManagedConnection mediante la chiamata mc.destroy()

Se il ConnectionManager è da considerarsi il ConnectionManager predefinito, può registrare anche un interesse nello stato della serie of MQPoolTokens gestita da MQEnvironment. A tale scopo, è necessario prima costruire un oggetto MQPoolServices, quindi registrare un oggetto MQPoolServicesEventListener con l'oggetto MQPoolServices:

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

Il listener viene notificato quando un MQPoolToken viene aggiunto o rimosso dalla serie o quando il ConnectionManager predefinito cambia. L'oggetto MQPoolServices fornisce anche un modo per richiedere la dimensione corrente della serie di MQPoolTokens.

## Compilazione e verifica dei programmi MQ base Java

Prima di compilare i programmi MQ base Java, è necessario assicurare che la directory di installazione di MQSeries classi per Java sia nella variabile di ambiente CLASSPATH, come viene descritto nel "Capitolo 2. Procedure di installazione" a pagina 9.

### Compilazione e verifica dei programmi MQ base Java

Per compilare una classe “MyClass.java”, utilizzare il comando:

```
javac MyClass.java
```

## Esecuzione di applet MQ base Java

Se si scrive un’applet (sottoclasse di java.applet.Applet), è necessario creare un file HTML che fa riferimento alla propria classe prima di poterla eseguire. Un file HTML di esempio potrebbe avere il seguente aspetto:

```
<html>
<body>
<applet code="MyClass.class" width=200 height=400>
</applet>
</body>
</html>
```

Eseguire l’applet caricando questo file HTML in un browser Web Java oppure utilizzando il visualizzatore delle applet fornito con il Java Development Kit (JDK).

Per eseguire il visualizzatore, immettere il comando:

```
appletviewer myclass.html
```

## Esecuzione delle applicazioni di MQ base Java

Se si scrive un’applicazione (una class contenente un metodo main()), utilizzando il client o la modalità binding, eseguire il programma utilizzando l’interprete Java. Utilizzare il comando:

```
java MyClass
```

**Nota:** L’estensione “.class” viene omessa dal nome della classe.

## Analisi dei programmi MQ base Java

MQ base Java include una funzione di analisi da utilizzare per produrre messaggi diagnostici se si sospetta che il codice presenti un problema. Generalmente questa funzione viene utilizzata solo dietro richiesta del servizio IBM.

L’analisi è controllata dai metodi enableTracing e disableTracing della classe MQEnvironment. Ad esempio:

```
MQEnvironment.enableTracing(2);   // analisi a livello 2
 ...                              // questi comandi verranno utilizzati
MQEnvironment.disableTracing();   // disattivare l'analisi
```

L’analisi viene scritta nella console di Java (`System.err`).

Se il programma è un’applicazione, oppure viene eseguita dal disco locale utilizzando il comando appletviewer, è possibile ridirigere l’output dell’analisi in un file a scelta. Il frammento di codice che segue mostra un esempio di come ridirigere l’output dell’analisi in un file chiamato `myapp.trc`:

```
import java.io.*;

try {
  FileOutputStream
  traceFile = new FileOutputStream("myapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
catch (IOException ex) {
```

```
  // impossibile aprire il file,
  // analizzare su System.err invece
  MQEnvironment.enableTracing(2);
}
```

Sono disponibili cinque diversi livelli di analisi:

1. Fornisce analisi di entrata, uscita ed eccezioni
2. Fornisce informazioni sul parametro in aggiunta a 1
3. Fornisce intestazioni e blocchi di dati di MQSeries trasmessi e ricevuti in aggiunta a 2
4. Fornisce dati dei messaggi utente trasmessi e ricevuti in aggiunta a 3
5. Fornisce l'analisi dei metodi nella Virtual Machine Java in aggiunta a 4

Per analizzare dei metodi nella Virtual Machine Java con il livello di analisi 5:

- Per un'applicazione, eseguirla emettendo il comando `java_g` (anziché `java`)
- Per un'applet, eseguirla emettendo il comando `appletviewer_g` (anziché `appletviewer`)

**Nota:** `java_g` non è supportato su OS/400, ma una funzione simile viene fornita utilizzando `OPTION(*VERBOSE)` sul comando RUNJVA.

## Analisi dei programmi MQ base Java

# Capitolo 8. Comportamento dipendente dall'ambiente

In questo capitolo verrà descritto il comportamento delle classi Java nei vari ambienti in cui è possibile utilizzarle. Le classi MQSeries classi per Java consentono di creare applicazioni da utilizzare nei seguenti ambienti:

1. MQSeries Client per Java connesso ad un server MQSeries V2.x su piattaforme UNIX o Windows
2. MQSeries Client per Java connesso ad un server MQSeries V5 su piattaforme UNIX o Windows
3. MQSeries Client per Java connesso ad un server MQSeries per OS/390
4. MQSeries Bindings for Java in esecuzione su un server MQSeries V5 su piattaforme UNIX o Windows
5. MQSeries Bindings for Java in esecuzione su un server MQSeries per OS/390

In tutti i casi il codice MQSeries classi per Java utilizza i servizi forniti dal server MQSeries sottostante. Esistono delle differenze nel livello di funzione (ad esempio MQSeries V5 fornisce un superset della funzione di V2). Ci sono anche delle differenze nel comportamento di alcune chiamate e opzioni dell'API. Molte differenze di comportamento sono quasi irrilevanti e molte sono tra i server OS/390 ed i server su altre piattaforme.

Nel precedente elenco degli ambienti, un server MQSeries per OS/390 può essere in esecuzione con uno qualsiasi dei seguenti gestori code supportati:

- MQSeries per MVS/ESA V1R2
- MQSeries per OS/390 V2R1
- MQSeries per OS/390 V5R2

MQSeries classi per Java fornisce un "nucleo" di classi, che offrono una funzione ed un comportamento costante in tutti gli ambienti. Fornisce "estensioni V5", progettati per un utilizzo esclusivo in ambienti 2 e 4. Nelle sezioni che seguono viene descritto il nucleo e le estensioni.

## Dettagli sul nucleo

Le MQSeries classi per Java contengono il seguente insieme delle classi del nucleo, che può essere utilizzato in tutti gli ambienti soltanto con le variazioni minori elencate nella sezione "Restrizioni e variazioni per le classi centrali" a pagina 82.

- MQEnvironment
- MQException
- MQGetMessageOptions

  Tranne:
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  Tranne:
  - inquire()
  - set()
- MQMessage

  Tranne:

  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  Tranne:
  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  Tranne:
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQC

**Note:**

1. Alcune costanti non sono incluse nel nucleo (vedere "Restrizioni e variazioni per le classi centrali" per informazioni dettagliate) e non devono essere utilizzate in programmi completamente portabili.
2. Alcune piattaforme non supportano tutte le modalità di connessione. Su queste piattaforme è possibile utilizzare solo le classi e le opzioni centrali che fanno riferimento alle modalità supportate. (Vedere Tabella 1 a pagina 6.)

## Restrizioni e variazioni per le classi centrali

Sebbene le classi centrali generalmente si comportino in modo coerente in tutti gli ambienti, esistono alcune restrizioni e variazioni minori, documentate nella Tabella 12.

Al di là delle variazioni documentate, le classi centrali forniscono un comportamento coerente in tutti gli ambienti, anche se le classi MQSeries equivalenti normalmente presentano differenze di ambiente. In generale, il comportamento sarà lo stesso negli ambienti 2 e 4.

*Tabella 12. Restrizioni e variazioni delle classi centrali*

| Classe o elemento | Restrizioni e variazioni |
|---|---|
| MQGMO_LOCK<br>MQGMO_UNLOCK<br>MQGMO_BROWSE_MSG_UNDER_CURSOR | Causare MQRC_OPTIONS_ERROR se utilizzata negli ambienti 3 o 5. |
| MQPMO_NEW_MSG_ID<br>MQPMO_NEW_CORREL_ID<br>MQPMO_LOGICAL_ORDER | Produrre errori tranne che negli ambienti 2 e 4. (Vedere le estensioni V5.) |
| MQGMO_LOGICAL_ORDER<br>MQGMO_COMPLETE_MESSAGE<br>MQGMO_ALL_MSGS_AVAILABLE<br>MQGMO_ALL_SEGMENTS_AVAILABLE | Produrre errori tranne che negli ambienti 2 e 4. (Vedere le estensioni V5.) |

*Tabella 12. Restrizioni e variazioni delle classi centrali (Continua)*

| Classe o elemento | Restrizioni e variazioni |
|---|---|
| MQGMO_SYNCPOINT_IF_PERSISTENT | Produce errori nell'ambiente 1. (Vedere le estensioni V5.) |
| MQGMO_MARK_SKIP_BACKOUT | Causare MQRC_OPTIONS_ERROR tranne che negli ambienti 3 e 5. |
| MQCNO_FASTPATH_BINDING | Supportata solo nell'ambiente 4. (Vedere le estensioni V5.) |
| Campi MQPMRF_* | Supportata solo negli ambienti 2 e 4. |
| Inserimento di un messaggio con MQQueue.priority > MaxPriority | Respinta con MQCC_FAILED e MQRC_PRIORITY_ERROR negli ambienti 3 e 5.<br><br>Altri ambienti lo accettano con le avvertenze MQCC_WARNING e MQRC_PRIORITY_EXCEEDS_MAXIMUM e trattano il messaggio come se fosse inserito con MaxPriority. |
| BackoutCount | Ambienti 3 e 5 restituiscono un numero massimo di rifiuti pari a 255, anche se il messaggio è stato respinto più di 255 volte. |
| Nome della coda dinamica predefinita | CSQ.* per ambienti 3 e 5. AMQ.* per gli altri sistemi. |
| Opzioni di MQMessage.report:<br>MQRO_EXCEPTION_WITH_FULL_DATA<br>MQRO_EXPIRATION_WITH_FULL_DATA<br>MQRO_COA_WITH_FULL_DATA<br>MQRO_COD_WITH_FULL_DATA<br>MQRO_DISCARD_MSG | Non supportata se un messaggio di prospetto viene generato da un gestore code OS/390, sebbene possano essere impostate in tutti gli ambienti. La questione riguarda tutti gli ambienti Java in quanto il gestore code OS/390 potrebbe essere distante dall'applicazione Java. Evitare di basarsi su una qualsiasi di queste opzioni se esiste la possibilità che un gestore code OS/390 possa essere coinvolto. |
| Costruttori di MQQueueManager | Nell'ambiente 5, se le opzioni presenti in MQEnvironment (e l'argomento delle proprietà facoltativo) presuppongono una connessione client, il costruttore non riesce con MQRC_ENVIRONMENT_ERROR.<br><br>Nell'ambiente 5, il costruttore potrebbe anche restituire MQRC_CHAR_CONVERSION_ERROR. Assicurarsi che il componente National Language Resources di OS/390 Language Environment sia installato.In particolare, assicurarsi che siano disponibili le conversioni tra le code page IBM-1047 e ISO8859-1.<br><br>Nell'ambiente 5, il costruttore potrebbe anche restituire MQRC_UCS2_CONVERSION_ERROR. Le classi MQSeries per Java tentano di effettuare la conversione da Unicode dalla code page del gestore code ed utilizzano quella predefinita IBM-500 se non è disponibile una code page specifica. Assicurarsi di disporre delle tabelle di conversione appropriate per Unicode, che dovrebbero essere installate come parte della funzione opzionale di OS/390 C/C++ e assicurarsi che il Language Environment sia in grado di individuarle.Consultare il manuale *OS/390 C/C++ Programming Guide*, SC09-2362, per ulteriori informazioni sull'abilitazione delle conversioni UCS-2. |

## Estensioni Versione 5 in funzione in altri ambienti

Le MQSeries classi per Java contiene le seguenti funzioni specificamente destinate a un utilizzo con le estensioni API introdotte in MQSeries V5. Tali funzioni operano nel modo previsto solo negli ambienti 2 e 4. In questa sezione verrà descritto come funzionerebbero in altri ambienti.

**Opzione del costruttore MQQueueManager**

Il costruttore MQQueueManager include un argomento intero facoltativo che si associa nel campo MQCNO.options di MQI e viene utilizzato per passare da una connessione normale a una connessione veloce e viceversa.

Questa forma estesa del costruttore è accettata in tutti gli ambienti, purché le uniche opzioni utilizzate siano MQCNO_STANDARD_BINDING o MQCNO_FASTPATH_BINDING. Qualsiasi altra opzione determina la mancata riuscita del costruttore con MQRC_OPTIONS_ERROR. L'opzione veloce MQC.MQCNO_FASTPATH_BINDING è valida solo quando viene utilizzata nella modalità binding V5 MQSeries (ambiente 4). Se viene utilizzata in qualsiasi altro ambiente, l'opzione viene ignorata.

**Metodo MQQueueManager.begin()**

Questo metodo può essere utilizzato solo nell'ambiente 4. in qualsiasi altro ambiente non riesce con MQRC_ENVIRONMENT_ERROR. MQSeries per AS/400non supporta l'utilizzo del metodo `begin()` per avviare unità globali di lavoro coordinate dal gestore code.

**Opzioni MQPutMessageOptions**

I flag che seguono possono essere impostati nei campi delle opzioni MQPutMessageOptions in qualsiasi ambiente. Se tuttavia vengono utilizzati con un metodo MQQueue.put() successivo in ambienti diversi da 2 o 4, il metodo put() non riesce con MQRC_OPTIONS_ERROR.

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

**Opzioni MQGetMessageOptions**

I flag che seguono possono essere impostati nei campi delle opzioni MQGetMessageOptions in qualsiasi ambiente. Se tuttavia vengono utilizzati con un metodo MQQueue.get() successivo in ambienti diversi da 2 o 4, il metodo get() non riesce con MQRC_OPTIONS_ERROR.

- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

Il flag che segue può essere impostato nei campi delle opzioni MQGetMessageOptions in qualsiasi ambiente. Se tuttavia viene utilizzato con un metodo MQQueue.get() successivo nell'ambiente 1, il metodo get() non riesce con MQRC_OPTIONS_ERROR.

- MQGMO_SYNCPOINT_IF_PERSISTENT

**Campi MQGetMessageOptions**

I valori possono essere impostati nei seguenti campi, indipendentemente dall'ambiente. Se tuttavia il MQGetMessageOptions utilizzato su un metodo MQQueue.get() successivo contiene valori non predefiniti nell'esecuzione in un qualsiasi ambiente che non sia 2 o 4, il metodo get() non riesce con MQRC_GMO_ERROR. In ambienti diversi da 2 o 4, questi campi sono sempre impostati sui loro valori iniziali dopo ogni metodo get()eseguito correttamente.

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

**Nota:** Con MQSeries per OS/390 V2R1 o MQSeries per OS/390 V5R2 in esecuzione sul server, il campo MatchOptions non supporta i flag MQMO_MATCH_MSG_ID e MQMO_MATCH_CORREL_ID. Altri flag fanno in modo che il metodo get() non riesca con MQRC_GMO_ERROR.

**Elenchi di distribuzione**

Le classi indicate di seguito vengono utilizzate per la creazione di elenchi di distribuzione:

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

E' possibile creare e riempire MQDistributionList e MQDistributionListItems in qualsiasi ambiente, ma la creazione e l'apertura di MQDistributionList può avvenire correttamente solo negli ambienti 2 e 4. Un tentativo di crearne e aprirne una in qualsiasi altro ambiente viene respinto con MQRC_OD_ERROR.

**Campi MQPutMessageOptions**

Quattro campi in MQPMO vengono resi come le seguenti variabili nella classe MQPutMessageOptions:

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

Sebbene sia stato principalmente concepito per un utilizzo con gli elenchi di distribuzione, il server MQSeries V5 riempie anche i campi DestCount dopo un MQPUT a una singola coda. Se ad esempio, la coda risolve a una coda locale, knownDestCount è impostato su 1 e gli altri due campi sono impostati su 0. Negli ambienti 2 e 4, i valori impostati dal server V5 sono restituiti nella classe MQPutMessageOptions. Negli altri ambienti, i valori restituiti sono simulati nel modo seguente:

- Se il metodo put() riesce, unknownDestCount è impostato su 1 e gli altri sono impostati su 0.
- Se il metodo put() non riesce, invalidDestCount è impostato su 1 e gli altri sono impostati su 0.

recordFields viene utilizzato con gli elenchi di distribuzione. Un valore può essere scritto all'interno di recordFields in qualsiasi momento, indipendentemente dall'ambiente. Tuttavia viene ignorato se le opzioni MQPutMessage vengono utilizzate su un MQQueue.put() successivo anziché su MQDistributionList.put().

**Campi MQMD**

I seguenti campi MQMD riguardano principalmente la segmentazione dei messaggi:

- GroupId
- MsgSeqNumber
- Offset MsgFlags
- OriginalLength

Se un'applicazione imposta uno qualsiasi di questi campi MQMD su valori non predefiniti, e quindi utilizza un metodo put() o get() in un ambiente diverso da 2 o 4, put() o get() producono un errore (MQRC_MD_ERROR). Una corretta applicazione di put() o get() in un ambiente diverso da 2 o 4 lascia sempre i nuovi campi MQMD impostati sui loro valori predefiniti. Un messaggio raggruppato o segmentato non dovrebbe normalmente essere inviato ad un'applicazione Java eseguita su un gestore code che non sia MQSeries Versione 5 o successiva. Se un'applicazione del genere non emette un comando get, e il messaggio fisico da recuperare fa parte di un gruppo o di un messaggio segmentato (presenta valori non predefiniti per i

campi MQMD), viene recuperato senza errori. Tuttavia, i campi MQMD in MQMessage non vengono aggiornati. La proprietà del formato MQMessage viene impostata su MQFMT_MD_EXTENSION e i dati del messaggio vengono fatti precedere da una struttura MQMDE contenente i valori relativi ai nuovi campi.

# Capitolo 9. Le classi e le interfacce MQ base Java

Questo capitolo descrive tutte le classi e le interfacce MQSeries classi per Java. Contiene informazioni relative alle variabili, ai costruttori ed ai metodi delle singole classi e interfacce.

Sono descritte le seguenti classi:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

Sono descrite le seguenti interfacce:
- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData

# MQChannelDefinition

```
java.lang.Object
  |
  +-- com.ibm.mq.MQChannelDefinition
```

classe pubblica **MQChannelDefinition**
estende **Object**

La classe MQChannelDefinition viene utilizzata per trasmettere informazioni relativa alla connessione al Gestore code alle uscite di invio, ricezione e sicurezza.

**Nota:** Questa classe non è valida quando si stabilisce una connessione diretta a MQSeries in modalità di binding.

## Variabili

**channelName**
```
public String channelName
```
Il nome del canale tramite cui viene stabilita la connessione.

**queueManagerName**
```
public String queueManagerName
```
Il nome del Gestore code con cui viene stabilita la connessione.

**maxMessageLength**
```
public int maxMessageLength
```
La lunghezza massima di messaggio che è possibile inviare al Gestore code.

**securityUserData**
```
public String securityUserData
```
Un'area di memorizzazione che può essere utilizzata dall'uscita di sicurezza. Le informazioni memorizzate in quest'area vengono conservate tra le chiamate all'uscita di sicurezza e sono anche disponibili per le uscite di invio e di ricezione.

**sendUserData**
```
public String sendUserData
```
Un'area di memorizzazione che può essere utilizzata dall'uscita di invio. Le informazioni memorizzate in quest'area vengono conservate tra le chiamate all'uscita di invio e sono anche disponibili per le uscite di sicurezza e di ricezione.

**receiveUserData**
```
public String receiveUserData
```
Un'area di memorizzazione che può essere utilizzata dall'uscita di ricezione. Le informazioni memorizzate in quest'area vengono conservate tra le chiamate all'uscita di ricezione e sono anche disponibili per le uscite di invio e di sicurezza.

**connectionName**
```
public String connectionName
```
Il nome host TCP/IP della macchina su cui risiede il Gestore code.

**remoteUserId**

```
public String remoteUserId
```

L'id utente utilizzato per stabilire la connessione.

**remotePassword**

```
public String remotePassword
```

La password utilizzata per stabilire la connessione.

## Costruttori

**MQChannelDefinition**

```
public MQChannelDefinition()
```

# MQChannelExit

```
java.lang.Object
   |
   └ com.ibm.mq.MQChannelExit
```

classe pubblica **MQChannelExit**
estende **Object**

Questa classe definisce le informazioni di contesto passate alle uscite di invio, ricezione e sicurezza quando vengono chiamate. La variabile exitResponse deve essere impostata dall'uscita per indicare la successiva operazione che deve essere eseguita da MQSeries Client per Java.

**Nota:** Questa classe non è valida quando si stabilisce una connessione diretta a MQSeries in modalità di binding.

## Variabili

**MQXT_CHANNEL_SEC_EXIT**
```
public final static int MQXT_CHANNEL_SEC_EXIT
```

**MQXT_CHANNEL_SEND_EXIT**
```
public final static int MQXT_CHANNEL_SEND_EXIT
```

**MQXT_CHANNEL_RCV_EXIT**
```
public final static int MQXT_CHANNEL_RCV_EXIT
```

**MQXR_INIT**
```
public final static int MQXR_INIT
```

**MQXR_TERM**
```
public final static int MQXR_TERM
```

**MQXR_XMIT**
```
public final static int MQXR_XMIT
```

**MQXR_SEC_MSG**
```
public final static int MQXR_SEC_MSG
```

**MQXR_INIT_SEC**
```
public final static int MQXR_INIT_SEC
```

**MQXCC_OK**
```
public final static int MQXCC_OK
```

**MQXCC_SUPPRESS_FUNCTION**
```
public final static int MQXCC_SUPPRESS_FUNCTION
```

**MQXCC_SEND_AND_REQUEST_SEC_MSG**
```
public final static int MQXCC_SEND_AND_REQUEST_SEC_MSG
```

**MQXCC_SEND_SEC_MSG**
```
public final static int MQXCC_SEND_SEC_MSG
```

**MQXCC_SUPPRESS_EXIT**
```
public final static int MQXCC_SUPPRESS_EXIT
```

**MQXCC_CLOSE_CHANNEL**
```
public final static int MQXCC_CLOSE_CHANNEL
```

**exitID** `public int exitID`

Il tipo di uscita richiamato. Per una MQSecurityExit è sempre MQXT_CHANNEL_SEC_EXIT. Per una MQSendExit è sempre MQXT_CHANNEL_SEND_EXIT e per una MQReceiveExit è sempre MQXT_CHANNEL_RCV_EXIT.

**exitReason**
`public int exitReason`

La condizione per cui richiamare l'uscita. I valori possibili sono:

**MQXR_INIT**
Avvio dell'uscita; viene richiamata dopo che è stata eseguita la negoziazione delle condizioni di connessione del canale ma prima che venga eseguita la trasmissione di flussi di sicurezza.

**MQXR_TERM**
Chiusura dell'uscita; viene richiamata dopo che è stata eseguita la trasmissione dei flussi di annullamento della connessione ma prima che venga eseguita la cancellazione della connessione socket.

**MQXR_XMIT**
Per un'uscita di invio, indica che i dati devono essere trasmessi al Queue Manager.

Per un'uscita di ricezione, indica che i dati sono stati ricevuti dal Queue Manager.

**MQXR_SEC_MSG**
Indica all'uscita di sicurezza che il Queue Manager ha ricevuto un messaggio di sicurezza.

**MQXR_INIT_SEC**
Indica che l'uscita deve avviare il dialogo di sicurezza con il Queue Manager.

**exitResponse**
`public int exitResponse`

Viene impostata dall'uscita per indicare la successiva operazione che deve essere eseguita da MQSeries classi per Java. I valori validi sono:

**MQXCC_OK**
Impostata dall'uscita di sicurezza per indicare che gli scambi di sicurezza sono stati completati.

Impostata dall'uscita di invio per indicare che i dati restituiti devono essere trasmessi al Queue Manager.

Impostata dall'uscita di ricezione per indicare che i dati restituiti sono disponibili per essere elaborati da MQSeries Client per Java.

**MQXCC_SUPPRESS_FUNCTION**
Impostata dall'uscita di sicurezza per indicare che bisogna interrompere le comunicazioni con il Queue Manager.

**MQXCC_SEND_AND_REQUEST_SEC_MSG**
Impostata dall'uscita di sicurezza per indicare che i dati restituiti devono essere trasmessi al Queue Manager e che è prevista la ricezione di una risposta inviata dal Queue Manager.

**MQXCC_SEND_SEC_MSG**
Impostata dall'uscita di sicurezza per indicare che i dati restituiti devono essere trasmessi al Queue Manager e che non è prevista la ricezione di una risposta.

**MQXCC_SUPPRESS_EXIT**
Impostata da un'uscita per indicare che non deve più essere richiamata.

**MQXCC_CLOSE_CHANNEL**
Impostata da un'uscita per indicare che deve essere chiusa la connessione con il Queue Manager.

**maxSegmentLength**

```
public int maxSegmentLength
```

La lunghezza massima di una trasmissione al Queue Manager.

Se l'uscita restituisce dei dati che devono essere inviati al Queue Manager, la lunghezza di questi dati non deve essere superiore a questo valore.

**exitUserArea**

```
public byte exitUserArea[]
```

Un'area di memorizzazione disponibile per l'uscita.

I dati inseriti nella exitUserArea sono conservati da MQSeries Client per Java tra le chiamate alle uscite con lo stesso exitID. Questo significa che le uscite di invio, ricezione e sicurezza dispongono di aree utente indipendenti e dedicate.

**capabilityFlags**

```
public static final int capabilityFlags
```

Indica la capacità del Queue Manager.

E' supportato solo il flag MQC.MQCF_DIST_LISTS.

**fapLevel**

```
public static final int fapLevel
```

Il livello FAP (Format and Protocol) negoziato.

## Costruttori

**MQChannelExit**

```
public MQChannelExit()
```

# MQDistributionList

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQDistributionList
```

classe pubblica **MQDistributionList**
estende **MQManagedObject** (Vedere pagina 110.)

**Nota:** E' possibile utilizzare questa classe solo quando si è connessi a un gestore code MQSeries Versione 5 (o successiva).

Una MQDistributionList viene creata utilizzando il costruttore MQDistributionList oppure utilizzando il metodo accessDistributionList per MQQueueManager.

Un elenco di distribuzione rappresenta una serie di code aperte a cui è possibile inviare messaggi utilizzando una singola chiamata al metodo put(). (Vedere ″Elenchi di distribuzione″ in *MQSeries Application Programming Guide*.)

## Constructors

**MQDistributionList**

```
public MQDistributionList(MQQueueManager qMgr,
               MQDistributionListItem[] litems,
               int openOptions,
               String alternateUserId)
                          throws MQException
```

`qMgr` è il gestore code in cui l'elenco deve essere aperto.

`litems` sono le voci da includere nell'elenco di distribuzione.

Vedere "accessDistributionList" a pagina 166 per informazioni dettagliate sui metodi restanti.

## Metodi

**put**

```
public synchronized void put(MQMessage message,
              MQPutMessageOptions putMessageOptions )
                            throws MQException
```

Inserisce un messaggio destinato alle code negli elenchi di distribuzione.

**Parametri**

*message*
Un parametro di input/output contenente informazioni sul descriptor del messaggio e i dati sul messaggio restituiti.

*putMessageOptions*
Opzioni che controllano l'azione di MQPUT. (Vedere "MQPutMessageOptions" a pagina 144 per informazioni dettagliate.)

Produce MQException se l'inserimento non riesce.

## MQDistributionList

**getFirstDistributionListItem**

```
public MQDistributionListItem getFirstDistributionListItem()
```

Restituisce la prima voce nell'elenco di distribuzione o *null* se l'elenco è vuoto.

**getValidDestinationCount**

```
public int getValidDestinationCount()
```

Restituisce il numero di voci nell'elenco di distribuzione aperte correttamente.

**getInvalidDestinationCount**

```
public int getInvalidDestinationCount()
```

Restituisce il numero di voci nell'elenco di distribuzione la cui apertura non è riuscita.

# MQDistributionListItem

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
          |
          +-- com.ibm.mq.MQDistributionListItem
```

classe pubblica **MQDistributionListItem**
estende **MQMessageTracker** (Vedere pagina 136.)

**Nota:** E' possibile utilizzare questa classe solo quando si è connessi a un gestore code MQSeries Versione 5 (o successiva).

Una MQDistributionListItem rappresenta un solo elemento (coda) all'interno di un elenco di distribuzione.

## Variabili

**completionCode**
```
public int completionCode
```

Il codice di completamento ottenuto dall'ultima operazione su questo elemento. Se si è trattato della creazione di una MQDistributionList, il codice di completamento è relativo all'apertura della coda. Se si è trattato di una operazione di inserimento, il codice di completamento è relativo al tentativo di inserire un messaggio in questa coda.

Il valore iniziale è ″0″.

**queueName**
```
public String queueName
```

Il nome di una coda che si desidera utilizzare con un elenco di distribuzione. Non può essere il nome di una coda modello.

Il valore iniziale è ″″.

**queueManagerName**
```
public String queueManagerName
```

Il nome del gestore code su cui è definita la coda.

Il valore iniziale è ″″.

**reasonCode**
```
public int reasonCode
```

Il codice motivo ottenuto dall'ultima operazione su questo elemento. Se si è trattato della creazione di una MQDistributionList, il codice motivo è relativo all'apertura della coda. Se si è trattato di una operazione di inserimento, il codice motivo è relativo al tentativo di inserire un messaggio in questa coda.

Il valore iniziale è ″0″.

## Costruttori

**MQDistributionListItem**
```
public MQDistributionListItem()
```

### MQDistributionListItem

Costruire un nuovo oggetto MQDistributionListItem.

# MQEnvironment

```
java.lang.Object
    |
    +-- com.ibm.mq.MQEnvironment
```

classe pubblica **MQEnvironment**
estende **Object**

**Nota:** Tutti i metodi e gli attributi di questa classe si applicano alle connessioni client di MQSeries classi per Java, ma solo enableTracing, disableTracing, properties e version_notice si applicano alle connessioni binding.

MQEnvironment contiene variabili dei membri statici che controllano l'ambiente in cui viene creato un oggetto MQQueueManager (e la corrispondente connessione a MQSeries).

I valori impostati nella classe MQEnvironment hanno effetto quando viene richiamato il costruttore MQQueueManager, pertanto è necessario impostare i valori nella classe MQEnvironment prima di creare un'istanza MQQueueManager.

## Variabili

**Nota:** Le variabili contrassegnate da * non vengono applicate quando si effettua una connessione diretta a MQSeries in modalità binding.

**version_notice**

```
public final static String version_notice
```

La versione corrente di MQSeries classi per Java.

**securityExit***

```
public static MQSecurityExit securityExit
```

Un'uscita di sicurezza consente di personalizzare i flussi di sicurezza che hanno luogo quando viene effettuato un tentativo di connettersi a un gestore code.

Per fornire la propria uscita di sicurezza, definire una classe che implementa l'interfaccia MQSecurityExit e assegnare securityExit su un'istanza di quella classe. In alternativa è possibile lasciare securityExit impostato su null. In tal caso non verrà richiamata alcuna uscita di sicurezza.

Vedere anche la sezione "MQSecurityExit" a pagina 176.

**sendExit***

```
public static MQSendExit sendExit
```

Un'uscita di sicurezza consente di esaminare, e se possibile modificare, i dati inviati a un gestore code. Viene generalmente utilizzata insieme a un'uscita di ricezione corrispondente nel gestore code.

Per fornire la propria uscita di invio, definire una classe che implementa l'interfaccia MQSendExit e assegnare sendExit su un'istanza di quella classe. In alternativa è possibile lasciare sendExit impostato su null. In tal caso non verrà richiamata alcuna uscita di invio.

Vedere anche la sezione “MQSendExit” a pagina 178.

**receiveExit***

```
public static MQReceiveExit receiveExit
```

Un’uscita di ricezione consente di esaminare, e se possibile modificare, i dati ricevuti da un gestore code. Viene generalmente utilizzata insieme a un’uscita di invio corrispondente nel gestore code.

Per fornire la propria uscita di ricezione, definire una classe che implementa l’interfaccia MQReceiveExit e assegnare receiveExit su un’istanza di quella classe. In alternativa è possibile lasciare receiveExit impostato su null. In tal caso non verrà richiamata alcuna uscita di ricezione.

Vedere anche la sezione “MQReceiveExit” a pagina 174.

**hostname***

```
public static String hostname
```

Il nome host TCP/IP della macchina su cui risiede il MQSeries. Se il nome host non è impostato, e non è stata impostata alcuna proprietà di sostituzione, la modalità binding verrà utilizzata per connettersi al gestore code locale.

**port*** `public static int port`

La porta a cui connettersi. Questa è la porta su cui il server MQSeries è in attesa di richieste di connessione in arrivo. Il valore predefinito è 1414.

**channel***

```
public static String channel
```

Il nome del canale a cui connettersi sul gestore code di destinazione. *E’ necessario* impostare questa variabile dei membri, o la proprietà corrispondente, prima di creare un’istanza MQQueueManager per un utilizzo in modalità client.

**userID***

```
public static String userID
```

Equivalente alla variabile di ambiente MQ_USER_ID di MQSeries.

Se un’uscita di sicurezza non è definita per questo client, il valore di userID viene trasmesso al server ed è da quel momento in poi disponibile all’uscita di sicurezza del server quando viene richiamato. Il valore può essere utilizzato per verificare l’identità del client MQSeries.

Il valore predefinito è `""`.

**password***

```
public static String password
```

Equivalente alla variabile di ambiente MQ_PASSWORD di MQSeries.

Se un’uscita di sicurezza non è definita per questo client, il valore di password viene trasmesso al server ed è disponibile all’uscita di sicurezza del server quando viene richiamato. Il valore può essere utilizzato per verificare l’identità del client MQSeries.

Il valore predefinito è `""`.

**properties**

```
public static java.util.Hashtable properties
```

Una serie di coppie chiave/valore che definisce l'ambiente MQSeries.

Questa tabella hash consente di impostare le proprietà dell'ambiente come coppie chiave/valore anziché come singole variabili.

Le proprietà possono anche essere passate come una tabella hash in un parametro sul costruttore MQQueueManager. Le proprietà passate sul costruttore hanno la precedenza sui valori impostati con questa variabile delle proprietà, ma sono altrimenti interscambiabili. L'ordine di precedenza della ricerca delle proprietà è il seguente:

1. Parametro `properties` sul costruttore MQQueueManager
2. MQEnvironment.properties
3. Altre variabili MQEnvironment
4. Valori predefiniti costanti

Le possibili coppie chiave/valore sono illustrate nella tabella che segue:

| Chiave | Valore |
|---|---|
| MQC.CCSID_PROPERTY | Integer (sostituisce MQEnvironment.CCSID) |
| MQC.CHANNEL_PROPERTY | String (sostituisce MQEnvironment.channel) |
| MQC.CONNECT_OPTIONS_PROPERTY | Integer, che assume automaticamente MQC.MQCNO_NONE |
| MQC.HOST_NAME_PROPERTY | String (sostituisce MQEnvironment.hostname) |
| MQC.ORB_PROPERTY | org.omg.CORBA.ORB (facoltativo) |
| MQC.PASSWORD_PROPERTY | String (sostituisce MQEnvironment.password) |
| MQC.PORT_PROPERTY | Integer (sostituisce MQEnvironment.port) |
| MQC.RECEIVE_EXIT_PROPERTY | MQReceiveExit (sostituisce MQEnvironment.receiveExit) |
| MQC.SECURITY_EXIT_PROPERTY | MQSecurityExit (sostituisce MQEnvironment.securityExit) |
| MQC.SEND_EXIT_PROPERTY | MQSendExit (sostituisce MQEnvironment.sendExit.) |
| MQC.TRANSPORT_PROPERTY | MQC.TRANSPORT_MQSERIES_BINDINGS o MQC.TRANSPORT_MQSERIES_CLIENT o MQC.TRANSPORT_VISIBROKER o MQC.TRANSPORT_MQSERIES (l'impostazione predefinita, che seleziona binding o client, in base al valore di ″nomehost″.) |
| MQC.USER_ID_PROPERTY | String (sostituisce MQEnvironment.userID.) |

**CCSID***

```
public static int CCSID
```

Il CCSID utilizzato dal client.

La modifica di questo valore influisce sul modo in cui il gestore code con il quale si stabilisce la connessione traduce le informazioni nelle intestazioni MQSeries. Tutti i dati nelle intestazioni MQSeries vengono tratti dalla parte fissa del codeset ASCII, ad eccezione dei dati nei campi applicationIdData e putApplicationName della classe MQMessage. (Vedere “MQMessage” a pagina 113.)

Se si evita di utilizzare dei caratteri della parte variabile del codeset ASCII per questi due campi, è possibile passare il CCSID senza provocare danni da 819 a qualsiasi altro codeset ASCII.

Se il CCSID diventa identico a quello del gestore code a cui si sta effettuando la connessione, si ottengono dei considerevoli vantaggi in termini di prestazioni in quanto non vengono effettuati tentativi di tradurre le intestazioni dei messaggi.

Il valore predefinito è 819.

## Costruttori

**MQEnvironment**

```
public MQEnvironment()
```

## Metodi

**disableTracing**

```
public static void disableTracing()
```

Disattiva la funzione di analisi di MQSeries Client per Java.

**enableTracing**

```
public static void enableTracing(int level)
```

Attiva la funzione di analisi di MQSeries Client per Java.

**Parametri**

*level* Il livello di analisi richiesto, da 1 a 5 (5 corrisponde al livello più dettagliato).

**enableTracing**

```
public static void enableTracing(int level,
                                 OutputStream stream)
```

Attiva la funzione di analisi di MQSeries Client per Java.

**Parametri:**

*level* Il livello di analisi richiesto, da 1 a 5 (5 corrisponde al livello più dettagliato).

*stream* Il flusso su cui viene scritta l’analisi.

**setDefaultConnectionManager**

```
public static void setDefaultConnectionManager(MQConnectionManager cxManager)
```

Imposta il MQConnectionManager fornito in modo che sia il ConnectionManager predefinito. Il ConnectionManager predefinito viene utilizzato quando non viene specificato alcun ConnectionManager sul costruttore MQQueueManager. Questo metodo inoltre svuota la serie di MQPoolTokens.

**Parametri:**

*cxManager*
Il MQConnectionManager destinato ad essere il ConnectionManager predefinito.

**setDefaultConnectionManager**

```
public static void setDefaultConnectionManager
                    (javax.resource.spi.ConnectionManager cxManager)
```

Imposta il ConnectionManager predefinito e svuota la serie di MQPoolToken. Il ConnectionManager predefinito viene utilizzato quando non viene specificato alcun ConnectionManager sul costruttore MQQueueManager.

Questo metodo richiede una JVM su Java 2 v1.3 o successiva, con JAAS 1.0 o versione successiva installato.

**Parametri:**

*cxManager*
Il ConnectionManager predefinito (che implementa l'interfaccia javax.resource.spi.ConnectionManager).

**getDefaultConnectionManager**

```
public static javax.resource.spi.ConnectionManager
                                getDefaultConnectionManager()
```

Restituisce il ConnectionManager predefinito. Se il ConnectionManager predefinito è in realtà un MQConnectionManager, restituisce null.

**addConnectionPoolToken**

```
public static void addConnectionPoolToken(MQPoolToken token)
```

Aggiunge il MQPoolToken fornito alla serie di token. Un ConnectionManager predefinito può utilizzarlo come suggerimento. Generalmente è abilitato quando nella serie è presente almeno un token.

**Parametri:**

*token* Il MQPoolToken da aggiungere alla serie di token.

**addConnectionPoolToken**

```
public static MQPoolToken addConnectionPoolToken()
```

Costruisce un MQPoolToken e lo aggiunge alla serie di token. Il MQPoolToken viene restituito all'applicazione da trasmettere successivamente in removeConnectionPoolToken().

**removeConnectionPoolToken**

```
public static void removeConnectionPoolToken(MQPoolToken token)
```

Rimuove il MQPoolToken specificato dalla serie di token. Se quel MQPoolToken non è compreso nella serie, non viene eseguita alcuna operazione.

**Parametri:**

*token* Il MQPoolToken da rimuovere dalla serie di token.

# MQException

```
java.lang.Object
   |
   +-- java.lang.Throwable
          |
          +-- java.lang.Exception
                 |
                 +-- com.ibm.mq.MQException
```

classe pubblica **MQException**
estende **Exception**

Una classe MQException viene prodotta ogni volta che si verifica un errore MQSeries. E' possibile modificare il flusso di output per le eccezioni registrate impostando il valore di MQException.log. Il valore predefinito è System.err. Questa classe contiene le definizioni delle costanti del codice di completamento e del codice di errore. Le costanti che iniziano con MQCC_ sono codici di completamento di MQSeries e le costanti che iniziano con MQRC_ sono i codici motivo MQSeries. La *MQSeries Application Programming Reference* contiene una descrizione completa di questi errori e delle loro probabili cause.

## Variabili

**log** `public static java.io.outputStreamWriter log`

Flusso in cui sono registrate le eccezioni. (L'impostazione predefinita è System.err.) Con un'impostazione su null, non si verifica alcuna registrazione.

**completionCode**
`public int completionCode`

Il codice di completamento di MQSeries che ha dato origine all'errore. I valori possibili sono:
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

**reasonCode**
`public int reasonCode`

Il codice motivo di MQSeries che descrive l'errore. Per una spiegazione completa dei codici motivo, fare riferimento a *MQSeries Application Programming Reference*.

**exceptionSource**
`public Object exceptionSource`

L'istanza dell'oggetto che ha prodotto l'eccezione. E' possibile utilizzarla come parte della diagnostica nel determinare la causa di un errore.

## Costruttori

**MQException**

```
public MQException(int completionCode,
                   int reasonCode,
                   Object source)
```

Costruire un nuovo oggetto MQException.

**Parametri**

*completionCode*
Il codice di completamento di MQSeries.

*reasonCode*
Il codice motivo di MQSeries.

*source* L'oggetto in cui si è verificato l'errore.

# MQGetMessageOptions

```
java.lang.Object
   |
   +-- com.ibm.mq.MQGetMessageOptions
```

classe pubblica **MQGetMessageOptions**
estende **Object**

Questa classe contiene le opzioni che controllano il comportamento di MQQueue.get().

**Nota:** Il comportamento di alcune delle opzioni disponibili in questa classe dipende dall'ambiente in cui vengono utilizzate. Questi elementi sono contrassegnati con un *. Consultare la sezione "Capitolo 8. Comportamento dipendente dall'ambiente" a pagina 81 per informazioni più dettagliate.

## Variabili

**options**

```
public int options
```

Opzioni che controllano l'azione di MQQueue.get. E' possibile specificare tutti o nessuno dei valori che seguono. Se sono richieste più opzioni, i valori possono essere aggiunti insieme o combinati utilizzando l'operatore OR.

**MQC.MQGMO_NONE**

**MQC.MQGMO_WAIT**
Attendere l'arrivo di un messaggio.

**MQC.MQGMO_NO_WAIT**
Restituire immediatamente non è disponibile alcun messaggio adatto.

**MQC.MQGMO_SYNCPOINT**
Richiamare il messaggio sotto il controllo del syncpoint. Il messaggio verrà contrassegnato come non disponibile per le altre applicazioni, ma verrà eliminato dalla coda solo quando si eseguirà il commit dell'unità di lavoro. Il messaggio verrà reso disponibile di nuovo se si esegue il backout dell'unità di lavoro.

**MQC.MQGMO_NO_SYNCPOINT**
Richiamare il messaggio senza controllo syncpoint.

**MQC.MQGMO_BROWSE_FIRST**
Esplorare dall'inizio della coda.

**MQC.MQGMO_BROWSE_NEXT**
Esplorare dalla posizione corrente della coda.

**MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR***
Esplorare il messaggio sotto al cursore di esplorazione.

**MQC.MQGMO_MSG_UNDER_CURSOR**
Richiamare il messaggio sotto al cursore di esplorazione.

**MQC.MQGMO_LOCK***
Bloccare il messaggio esplorato.

**MQC.MQGMO_UNLOCK***
Sbloccare un messaggio precedentemente bloccato.

**MQC.MQGMO_ACCEPT_TRUNCATED_MSG**
Consentire il troncamento dei dati del messaggio.

**MQC.MQGMO_FAIL_IF_QUIESCING**
L'operazione non riesce se il gestore code è in fase di chiusura.

**MQC.MQGMO_CONVERT**
Richiedere che i dati dell'applicazione da convertire siano conformi agli attributi characterSet e encoding di MQMessage, prima che dati vengano copiati nel buffer dei messaggi. Dal momento che la conversione viene applicata anche man mano che i dati vengono recuperati dal buffer, le applicazioni generalmente non impostano questa opzione.

**MQC.MQGMO_SYNCPOINT_IF_PERSISTENT***
Richiamare il messaggio con il controllo syncpoint se il messaggio è persistente.

**MQC.MQGMO_MARK_SKIP_BACKOUT***
Consentire il backout di un'unità di lavoro senza riavviare il messaggio sulla coda.

**Segmentare e raggruppare** I messaggi MQSeries possono essere inviati o ricevuti come una singola entità, possono essere suddivisi in diversi frammenti per l'invio e la ricezione e possono anche essere collegati ad altri messaggi in un gruppo.

Ogni porzione di dati inviata viene definita messaggio *fisico* e può essere un messaggio *logico* completo o un segmento di un messaggio logico più lungo.

Ogni messaggio fisico generalmente ha un diverso `MsgId`. Tutti i segmenti di un singolo messaggio logico hanno lo stesso valore `groupId` e `MsgSeqNumber`, ma il valore `Offset` è diverso per ciascun segmento. Il campo `Offset` fornisce l'offset dei dati nel messaggio fisico dall'inizio del messaggio logico. I segmenti generalmente hanno diversi valori `MsgId`, in quanto sono singoli messaggi fisici.

I messaggi logici che formano parte di un gruppo hanno lo stesso valore `groupId`, ma ciascun messaggio nel gruppo ha un valore `MsgSeqNumber` differente. I messaggi che fanno parte di un gruppo possono anche essere segmentati.

Le opzioni che seguono possono essere utilizzate per la gestione di messaggi segmentati o raggruppati:

**MQC.MQGMO_LOGICAL_ORDER***
Restituire messaggi in gruppi e segmenti di messaggi logici, in ordine logico.

**MQC.MQGMO_COMPLETE_MSG***
Richiamare solo i messaggi logici completi.

**MQC.MQGMO_ALL_MSGS_AVAILABLE***
Richiamare messaggi da un gruppo solo quando tutti gli altri messaggi nel gruppo sono disponibili.

**MQC.MQGMO_ALL_SEGMENTS_AVAILABLE***
Richiamare i segmenti di un messaggio logico solo quando tutti i segmenti nel gruppo sono disponibili.

**waitInterval**

```
public int waitInterval
```

Il tempo massimo (in millisecondi) che una chiamata MQQueue.get attende per l'arrivo di un messaggio appropriato (utilizzata insieme a MQC.MQGMO_WAIT). Un valore di MQC.MQWI_UNLIMITED indica che è richiesta un'attesa illimitata.

**resolvedQueueName**

```
public String resolvedQueueName
```

Questo è un campo di output che il gestore code imposta sul nome locale della coda da cui è stato richiamato il messaggio. Sarà diverso dal nome utilizzato per aprire la coda se una coda alias o modello è stata aperta.

**matchOptions***

```
public int matchOptions
```

Criteri di selezione che determinano il messaggio da recuperare. E' possibile impostare le seguenti opzioni di corrispondenza:

**MQC.MQMO_MATCH_MSG_ID**
ID messaggio da mettere in corrispondenza.

**MQC.MQMO_MATCH_CORREL_ID**
ID correlazione da mettere in corrispondenza.

**MQC.MQMO_MATCH_GROUP_ID**
ID gruppo da mettere in corrispondenza.

**MQC.MQMO_MATCH_MSG_SEQ_NUMBER**
Mettere in corrispondenza il numero di sequenza del messaggio.

**MQC.MQMO_NONE**
Nessuna corrispondenza richiesta.

**groupStatus***

```
public char groupStatus
```

Questo è un campo di output che indica se il messaggio richiamato si trova in un gruppo e, in tal caso, se è l'ultimo nel gruppo. I valori possibili sono:

**MQC.MQGS_NOT_IN_GROUP**
Il messaggio non è in un gruppo.

**MQC.MQGS_MSG_IN_GROUP**
Il messaggio è in un gruppo, ma non è l'ultimo.

**MQC.MQGS_LAST_MSG_IN_GROUP**
Il messaggio è l'ultimo nel gruppo. Questo è anche il valore restituito se il gruppo è composto da un solo messaggio.

**segmentStatus***

```
public char segmentStatus
```

Questo è un campo di output che indica se il messaggio recuperato è un segmento di un messaggio logico. Se il messaggio è un segmento, il flag indica se si tratta o meno dell'ultimo segmento. I valori possibili sono:

**MQC.MQSS_NOT_A_SEGMENT**
Il messaggio non è un segmento.

**MQC.MQSS_SEGMENT**
Il messaggio è un segmento, ma non è l'ultimo segmento del messaggio logico.

**MQC.MQSS_LAST_SEGMENT**
Il messaggio è l'ultimo segmento del messaggio logico. Questo è anche il valore restituito se il messaggio logico è composto da un solo segmento.

**segmentation***

```
public char segmentation
```

Questo è un campo di output che indica se la segmentazione è consentita dal momento che il messaggio recuperato è un segmento di un messaggio logico. I valori possibili sono:

**MQC.MQSEG_INHIBITED**
Segmentazione non consentita.

**MQC.MQSEG_ALLOWED**
Segmentazione consentita.

## Costruttori

**MQGetMessageOptions**

```
public MQGetMessageOptions()
```

Costruire un nuovo oggetto MQGetMessageOptions con opzioni impostate su MQC.MQGMO_NO_WAIT, un intervallo di attesa pari a zero e un nome di coda risolto vuoto.

# MQManagedObject

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
```

classe pubblica **MQManagedObject**
estende **Object**

MQManagedObject è una superclasse per MQQueueManager, MQQueue e MQProcess. Offre la possibilità di ricercare e impostare gli attributi di queste risorse.

## Variabili

**alternateUserId**
`public String alternateUserId`

L'eventuale id alternativo specificato all'apertura di questa risorsa. L'impostazione di questo attributo non ha effetto.

**name** `public String name`

Il nome di questa risorsa (il nome fornito sul metodo di accesso o il nome assegnato dal gestore code per una coda dinamica). L'impostazione di questo attributo non ha effetto.

**openOptions**
`public int openOptions`

Le opzioni specificate all'apertura di questa risorsa. L'impostazione di questo attributo non ha effetto.

**isOpen**
`public boolean isOpen`

Indica se questa risorsa è al momento aperta. Questo attributo è *deprecato* e non ha effetto.

**connectionReference**
`public  MQQueueManager connectionReference`

Il gestore code a cui appartiene la risorsa. L'impostazione di questo attributo non ha effetto.

**closeOptions**
`public int closeOptions`

Impostare questo attributo per controllare in che modo la risorsa viene chiusa. Il valore predefinito è MQC.MQCO_NONE e rappresenta l'unico valore permesso per tutte le risorse diverse dalle code dinamiche permanenti e le code dinamiche temporanee a cui accedono gli oggetti che le hanno create. Per queste code sono consentiti i seguenti valori aggiuntivi:

**MQC.MQCO_DELETE**
Eliminare la coda se non ci sono messaggi.

**MQC.MQCO_DELETE_PURGE**
Eliminare la coda, cancellando gli eventuali messaggi in essa contenuti.

## Costruttori

**MQManagedObject**

```
protected MQManagedObject()
```

Metodo del costruttore.

## Metodi

**getDescription**

```
public String getDescription()
```

Produce MQException.

Restituire la descrizione di questa risorsa così come viene conservata nel gestore code.

Se questo metodo viene richiamato dopo la chiusura della risorsa, viene prodotta una MQException.

**inquire**

```
public void inquire(int selectors[],
                    int intAttrs[],
                    byte charAttrs[])
```

throws MQException.

Restituisce una matrice di valori interi e una serie di stringhe di caratteri contenenti gli attributi di un oggetto (coda, processo gestore code).

Gli attributi da interrogare sono specificati nella matrice dei selettori. Fare riferimento a *MQSeries Application Programming Reference* per informazioni dettagliate sui selettori consentiti e sui corrispondenti valori interi.

Molti degli attributi più comuni possono essere interrogati utilizzando i metodi getXXX() definiti in MQManagedObject, MQQueue, MQQueueManager e MQProcess.

**Parametri**

*selectors*
Matrice di interi che identifica gi attributi con i valori su cui eseguire ricerche.

*intAttrs*
La matrice in cui i valori dell'attributo integer vengono restituiti. I valori dell'attributo integer vengono restituiti nello stesso ordine idei selettori dell'attributo integer nella matrice di selettori.

*charAttrs*
Il buffer in cui vengono restituiti gli attributi dei caratteri, concatenati. Gli attributi dei caratteri vengono restituiti nello stesso ordine dei selettori dell'attributo dei caratteri nella matrice di selettori. La lunghezza di ciascuna stringa dell'attributo viene fissata per ciascun attributo.

Produce MQException se l'operazione di ricerca non riesce.

## MQManagedObject

### isOpen

```
public boolean isOpen()
```

Restituisce il valore della variabile `isOpen`.

### set

```
public synchronized void set(int selectors[],
                             int intAttrs[],
                             byte charAttrs[])
```

throws MQException.

Impostare gli attributi definiti nel vettore del selettore.

Gli attributi da specificare sono specificati nella matrice dei selettori. Fare riferimento a *MQSeries Application Programming Reference* per informazioni dettagliate sui selettori consentiti e sui corrispondenti valori interi.

Alcuni degli attributi della coda possono essere impostati utilizzando i metodi setXXX() definiti in MQQueue.

**Parametri**

*selectors*
: Matrice di interi che identifica gi attributi con i valori da impostare.

*intAttrs*
: La matrice dei valori dell'attributo integer da impostare. Questi valori devono essere nello stesso ordine dei selettori dell'attributo integer nella matrice di selettori.

*charAttrs*
: Il buffer in cui sono concatenati gli attributi dei caratteri da impostare. Questi valori devono essere nello stesso ordine dei selettori dell'attributo dei caratteri nella matrice di selettori. La lunghezza di ogni attributo del carattere è fissa.

Produce MQException se l'impostazione non riesce.

### close

```
public synchronized void close()
```

throws MQException.

Chiudere l'oggetto. Dopo aver richiamato il metodo, non è consentita alcuna operazione ulteriore su questa risorsa. Il comportamento del metodo close può essere modificato impostando l'attributo closeOptions.

Produce MQException se la chiamata di MQSeries non riesce.

# MQMessage

```
java.lang.Object
    |
    +-- com.ibm.mq.MQMessage
```

classe pubblica **MQMessage**
implementa **DataInput**, **DataOutput**

MQMessage rappresenta sia il descrittore dei messaggi e i dati relativi a un messaggi di MQSeries. Esiste un gruppo di metodi readXXX per la lettura dei dati da un messaggio e un gruppo di metodi writeXXX per la scrittura di dati in un messaggio. Il formato di numeri e stringhe utilizzato da questi metodi di lettura e scrittura può essere controllato dalle variabili dei membri encoding e characterSet. Le variabili restanti contengono informazioni di controllo che accompagnano i dati dei messaggi dell'applicazione quando un messaggio viaggia tra le applicazioni che inviano e quelle che ricevono. L'applicazione può impostare i valori nella variabile dei membri prima di inserire un messaggio in una coda e può leggere i valori dopo aver richiamato un messaggio da una coda.

## Variabili

**report** `public int report`

Un prospetto è un messaggio su un altro messaggio. Questa variabile del membro consente all'applicazione di inviare il messaggio originale per specificare quali messaggi d prospetto sono richiesti, se i dati dell'applicazione devono essere inclusi in essi e in che modo devono essere impostati il messaggio e gli identificativi di correlazione nel prospetto. Possono essere richiesti tutti o nessuno dei tipi di prospetto che seguono:

- Errore
- Scadenza
- Conferma all'arrivo
- Conferma alla consegna

Per ciascun tipo solo uno dei tre valori specificati di seguito deve essere specificato, a seconda che i dati dei messaggi dell'applicazione debbano essere inclusi nel messaggio di prospetto.

**Nota:** I valori contrassegnati con ** nell'elenco che segue non sono supportati dai gestori code MVS e non dovrebbero essere utilizzati se è probabile che l'applicazione acceda a un gestore code MVS, indipendentemente dalla piattaforma su cui l'applicazione è in esecuzione.

I valori validi sono:

- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**
- MQC.MQRO_COA
- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD

- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

E' possibile specificare una delle opzioni che segue per controllare il modo in cui l'ID del messaggio viene generato per il messaggio di prospetto o di risposta:
- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

E' possibile specificare una delle opzioni che segue per controllare il modo in cui l'ID di correlazione del messaggio di prospetto o di risposta deve essere impostato:
- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

E' possibile specificare una delle seguenti opzioni per controllare la disposizione del messaggio originale quando non può essere consegnato alla coda di destinazione:
- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **

Se non è specificata alcuna opzione di prospetto, l'impostazione predefinita è:

```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

E' possibile specificare una o entrambe delle opzioni che seguono per richiedere che l'applicazione ricevente invii un messaggio di prospetto dell'azione positiva o negativa.
- MQRO_PAN
- MQRO_NAN

**messageType**

```
public int messageType
```

Indica il tipo di messaggio. I valori che seguono sono al momento definiti dal sistema:
- MQC.MQMT_DATAGRAM
- MQC.MQMT_REQUEST
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT

E' possibile anche utilizzare i valori definiti dall'applicazione. Tali valori dovrebbero essere compresi nell'intervallo che va da MQC.MQMT_APPL_FIRST a MQC.MQMT_APPL_LAST.

Il valore predefinito di questo campo è MQC.MQMT_DATAGRAM.

**expiry** `public int expiry`

Un'ora di scadenza espressa in decimi di secondo impostata dall'applicazione che inserisce il messaggio. Allo scadere dell'ora di scadenza del messaggio, il messaggio può essere cancellato dal gestore code. Se il messaggio ha specificato uno dei flag MQC.MQRO_EXPIRATION, verrà generato un prospetto quando il messaggio viene cancellato.

Il valore predefinito è MQC.MQEI_UNLIMITED e indica che il messaggio non scade mai.

**feedback**

```
public int feedback
```

Viene utilizzata con un messaggio di tipo MQC.MQMT_REPORT per indicare la natura del prospetto. I codici di feedback che seguono sono definiti dal sistema:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

Possono essere utilizzati anche i valori di feedback definiti dall'applicazione compresi nell'intervallo che va da MQC.MQFB_APPL_FIRST a MQC.MQFB_APPL_LAST.

Il valore predefinito di questo campo è MQC.MQFB_NONE, che indica che non viene fornito alcun feedback.

**encoding**

```
public int encoding
```

Questa variabile dei membri specifica la rappresentazione utilizzata per i valori numerici nei dati dei messaggi dell'applicazione.Valida per i dati binari, decimali compressi e a virgola mobile. Il comportamento dei metodi di lettura e scrittura per questi formati numerici viene modificata di conseguenza.

Le codifiche riportate di seguito sono definite per i valori interi binari.

**MQC.MQENC_INTEGER_NORMAL**
Valori interi in formato big-endian, come in Java

**MQC.MQENC_INTEGER_REVERSED**
Valori interi in formato little-endian, come quelli utilizzati dai PC.

Le codifiche riportate di seguito sono definite per i valori interi decimali compressi.

**MQC.MQENC_DECIMAL_NORMAL**
Valori decimali compressi in formato big-endian, come vengono utilizzati da System/390.

**MQC.MQENC_DECIMAL_REVERSED**
Decimale compresso in formato little-endian.

Le codifiche che seguono sono definite per i numeri a virgola mobile:

**MQC.MQENC_FLOAT_IEEE_NORMAL**
Virgole mobili IEEE big-endian, come in Java.

**MQC.MQENC_FLOAT_IEEE_REVERSED**
Virgole mobili IEEE little-endian, come utilizzati dai PC.

**MQC.MQENC_FLOAT_S390**
Virgola mobile del formato System/390.

Un valore per il campo della codifica dovrebbe essere creato aggiungendo un valore da ciascuna di queste tre sezioni oppure utilizzando l'operatore bitwise OR. Il valore predefinito è:

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

Per comodità, questo valore è rappresentato anche da MQC.MQENC_NATIVE. Questa impostazione determina la scrittura da parte di writeInt() di un valore intero in formato ig-endian e la lettura da parte di readnt() di un valore intero in formato big-endian. Se invece è stato impostato il flag MQC.MQENC_INTEGER_REVERSED, writeInt() dovrebbe scrivere un intero in formato little-endian mentre readInt() dovrebbe leggere un intero in formato little-endian.

Durante la conversione da virgole mobili in formato IEEE a System/390 può verificarsi una perdita in precisione.

**characterSet**

```
public int characterSet
```

Specifica l'identificativo del set di caratteri codificato dei dati dei caratteri nei dati dei messaggi dell'applicazione. Il comportamento dei metodi readString, readLine e writeString viene modificato di conseguenza.

Il valore predefinito per questo campo è MQC.MQCCSI_Q_MGR. Se tale valore viene utilizzato, si presuppone CharacterSet 819 (iso-8859-1/latin/ibm819). Sono supportati i valori dell'insieme dei caratteri illustrati in Tabella 13.

*Tabella 13. Identificati del set di caratteri*

| characterSet | Descrizione |
|---|---|
| 819 | iso-8859-1 / latin1 / ibm819 |
| 912 | iso-8859-2 / latin2 / ibm912 |
| 913 | iso-8859-3 / latin3 / ibm913 |
| 914 | iso-8859-4 / latin4 / ibm914 |
| 915 | iso-8859-5 / cirillico / ibm915 |
| 1089 | iso-8859-6 / arabo / ibm1089 |
| 813 | iso-8859-7 / greco / ibm813 |
| 916 | iso-8859-8 / ebraico / ibm916 |
| 920 | iso-8859-9 / latin5 / ibm920 |
| 37 | ibm037 |
| 273 | ibm273 |
| 277 | ibm277 |
| 278 | ibm278 |
| 280 | ibm280 |
| 284 | ibm284 |
| 285 | ibm285 |
| 297 | ibm297 |
| 420 | ibm420 |
| 424 | ibm424 |
| 437 | ibm437 / PC Original |

*Tabella 13. Identificati del set di caratteri (Continua)*

| characterSet | Descrizione |
|---|---|
| 500 | ibm500 |
| 737 | ibm737 / PC Greco |
| 775 | ibm775 / PC Baltico |
| 838 | ibm838 |
| 850 | ibm850 / PC Latin 1 |
| 852 | ibm852 / PC Latin 2 |
| 855 | ibm855 / PC Cirillico |
| 856 | ibm856 |
| 857 | ibm857 / PC Turco |
| 860 | ibm860 / PC Portoghese |
| 861 | ibm861 / PC Islandese |
| 862 | ibm862 / PC Ebraico |
| 863 | ibm863 / PC Francese canadese |
| 864 | ibm864 / PC Arabo |
| 865 | ibm865 / PC Nordico |
| 866 | ibm866 / PC Russo |
| 868 | ibm868 |
| 869 | ibm869 / PC Greco moderno |
| 870 | ibm870 |
| 871 | ibm871 |
| 874 | ibm874 |
| 875 | ibm875 |
| 918 | ibm918 |
| 921 | ibm921 |
| 922 | ibm922 |
| 930 | ibm930 |
| 933 | ibm933 |
| 935 | ibm935 |
| 937 | ibm937 |
| 939 | ibm939 |
| 942 | ibm942 |
| 948 | ibm948 |
| 949 | ibm949 |
| 950 | ibm950 / Big 5 Cinese tradizionale |
| 964 | ibm964 / CNS 11643 Cinese tradizionale |
| 970 | ibm970 |
| 1006 | ibm1006 |
| 1025 | ibm1025 |
| 1026 | ibm1026 |
| 1097 | ibm1097 |
| 1098 | ibm1098 |
| 1112 | ibm1112 |
| 1122 | ibm1122 |
| 1123 | ibm1123 |
| 1124 | ibm1124 |
| 1381 | ibm1381 |
| 1383 | ibm1383 |
| 2022 | JIS |
| 932 | PC Giapponese |
| 954 | EUCJIS |
| 1250 | Windows Latin 2 |
| 1251 | Windows Cirillico |
| 1252 | Windows Latino 1 |

*Tabella 13. Identificati del set di caratteri (Continua)*

| characterSet | Descrizione |
|---|---|
| 1253 | Windows Greco |
| 1254 | Windows Turco |
| 1255 | Windows Ebraico |
| 1256 | Windows Arabo |
| 1257 | Windows Baltico |
| 1258 | Windows Vietnamita |
| 33722 | ibm33722 |
| 5601 | ksc-5601 Korean |
| 1200 | Unicode |
| 1208 | UTF-8 |

**format**

```
public String format
```

Un nome di formato utilizzato dal mittente del messaggio per indicare la natura dei dati nel messaggio al destinatario. E' possibile utilizzare nomi di formato personalizzati, ma i nomi che iniziano con le lettere ″MQ″ hanno dei significati definiti dal gestore code. I formati incorporati del gestore code sono:

**MQC.MQFMT_NONE**
Nessun nome di formato.

**MQC.MQFMT_ADMIN**
Messaggio di richiesta/risposta del server comandi.

**MQC.MQFMT_COMMAND_1**
Messaggio di risposta del comando di Tipo 1.

**MQC.MQFMT_COMMAND_2**
Messaggio di risposta del comando di Tipo 2.

**MQC.MQFMT_DEAD_LETTER_HEADER**
Intestazione della lettera non consegnata.

**MQC.MQFMT_EVENT**
Messaggio dell'evento.

**MQC.MQFMT_PCF**
Messaggio definito dall'utente in formato di comando programmabile.

**MQC.MQFMT_STRING**
Messaggio composto interamente da caratteri.

**MQC.MQFMT_TRIGGER**
Messaggio trigger

**MQC.MQFMT_XMIT_Q_HEADER**
Intestazione della coda di trasmissione.

Il valore predefinito è MQC.MQFMT_NONE.

**priority**

```
public int priority
```

La priorità del messaggio. Il valore speciale MQC.MQPRI_PRIORITY_AS_Q_DEF può essere impostato anche nei messaggi in uscita. In tal caso la proprietà per il messaggio viene presa dall'attributo della proprietà predefinita della coda di destinazione.

Il valore predefinito è MQC.MQPRI_PRIORITY_AS_Q_DEF.

**persistence**

```
public int persistence
```

Persistenza del messaggio. Sono definiti i seguenti valori:
- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

Il valore predefinito è MQC.MQPER_PERSISTENCE_AS_Q_DEF, che indica che la persistenza relativa al messaggio dovrebbe essere presa dall'attributo della persistenza predefinito della coda di destinazione.

**messageId**
```
public byte messageId[]
```

Per una chiamata MQQueue.get() questo campo specifica l'identificativo del messaggio da richiamare. Generalmente il gestore code restituisce il primo messaggio con un identificativo di messaggio e di correlazione che corrisponde a quelli specificati. Il valore speciale MQC.MQMI_NONE consente a *tutti* gli identificativi di messaggio di corrispondere.

Per una chiamata MQQueue.put(), specifica l'identificativo di messaggio da utilizzare. Se MQC.MQMI_NONE non è specificata, il gestore code genera un identificativo del messaggio univoco quando il messaggio viene inserito. Il valore della variabile di questo membro viene aggiornato dopo l'operazione di inserimento per indicare che è stato utilizzato l'identificativo del messaggio.

Il valore predefinito è MQC.MQMI_NONE.

**correlationId**
```
public byte correlationId[]
```

Per una chiamata MQQueue.get(), questo campo specifica l'identificativo di correlazione del messaggio da richiamare. Generalmente il gestore code restituisce il primo messaggio con un identificativo di messaggio e di correlazione che corrisponde a quelli specificati. Il valore speciale MQC.MQCI_NONE consente la corrispondenza di *qualsiasi* identificativo di correlazione.

Per una chiamata MQQueue.put(), specifica l'identificativo di correlazione da utilizzare.

Il valore predefinito è MQC.MQCI_NONE.

**backoutCount**
```
public int backoutCount
```

Un conteggio del numero di volte in cui il messaggio è stato precedentemente restituito da una chiamata MQQueue.get() come parte di un'unità di lavoro e successivamente respinto.

Il valore predefinito è zero.

**replyToQueueName**
```
public String replyToQueueName
```

Il nome della coda messaggi a cui l'applicazione che ha emesso la richiesta di richiamo relativa al messaggio dovrebbe inviare i messaggi MQC.MQMT_REPLY e MQC.MQMT_REPORT.

Il valore predefinito è "".

**replyToQueueManagerName**
```
public String replyToQueueManagerName
```

Il nome del gestore code a cui dovrebbero essere inviati i messaggi di risposta o di prospetto.

Il valore predefinito è "".

Se il valore è "" su una chiamata MQQueue.put(), QueueManager è adatto al valore.

**userId** `public String userId`

Parte del contesto di identità del messaggio. identifica l'utente che ha originato questo messaggio.

Il valore predefinito è "".

**accountingToken**

```
public byte accountingToken[]
```

Parte del contesto di identità del messaggio. Consente a un'applicazione di determinare l'esecuzione del lavoro come conseguenza di un corretto caricamento del messaggio.

Il valore predefinito è "MQC.MQACT_NONE".

**applicationIdData**

```
public String applicationIdData
```

Parte del contesto di identità del messaggio. Si tratta di informazioni definite dalla suite di applicazioni e può essere utilizzato per fornire ulteriori informazioni sul messaggio o sull'elemento che lo ha originato.

Il valore predefinito è "".

**putApplicationType**

```
public int putApplicationType
```

Il tipo di applicazione che ha inserito il messaggio. Può trattarsi di un valore definito dal sistema o dall'utente. I valori che seguono sono definiti dal sistema:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

Il valore predefinito è il valore speciale MQC.MQAT_NO_CONTEXT, che indica che nel messaggio non è presente alcuna informazione sul contesto.

**putApplicationName**

```
public String putApplicationName
```

Il nome dell'applicazione che ha inserito il messaggio. Il valore predefinito è "".

**putDateTime**

```
public GregorianCalendar putDateTime
```

L'ora e la data di inserimento del messaggio.

**applicationOriginData**

```
public String applicationOriginData
```

Informazioni definite dall'applicazione che può essere utilizzata per fornire informazioni aggiuntive sull'origine del messaggio.

Il valore predefinito è "".

**groupId**
```
public byte[] groupId
```
Una stringa di byte che identifica il gruppo di messaggi a cui appartiene il messaggio fisico.

Il valore predefinito è ″MQC.MQGI_NONE″.

**messageSequenceNumber**
```
public int messageSequenceNumber
```
Il numero di sequenza di un messaggio logico all'interno di un gruppo.

**offset** `public int offset`

In un messaggio segmentato, l'offset dei dati in un messaggio fisico dall'inizio di un messaggio logico.

**messageFlags**
```
public int messageFlags
```
Flag che controllano la segmentazione e lo stato di un messaggio.

**originalLength**
```
public int originalLength
```
La lunghezza originale di un messaggio segmentato.

## Costruttori

**MQMessage**
```
public MQMessage()
```
Creare un nuovo messaggio con le informazioni sul descrittore del messaggio predefinito e un buffer di messaggi vuoto.

## Metodi

**getTotalMessageLength**
```
public int getTotalMessageLength()
```
Il numero totale di byte nel messaggio così come viene memorizzato nella coda dei messaggi da cui il messaggio è stato richiamato (o da cui si è tentato di richiamarlo). Quando un metodo MQQueue.get() non riesce con un codice di errore di messaggio troncato, questo metodo indica la dimensione totale del messaggio sulla coda.

Vedere anche la sezione "MQQueue.get" a pagina 149.

**getMessageLength**
```
public int getMessageLength
```
Produce IOException.

Il numero dei byte dei dati del messaggio in questo oggetto MQMessage.

**getDataLength**
```
public int getDataLength()
```
Produce MQException.

Il numero dei byte dei dati del messaggio da leggere ancora.

**seek**

```
public void seek(int pos)
```

Produce IOException.

Spostare il cursore sulla posizione assoluta nel buffer del messaggio fornita da *pos*. Le letture e le scritture successive verranno eseguite in questa posizione nel buffer.

Produce EOFException se pos è al di fuori della lunghezza dei dati del messaggio.

**setDataOffset**

```
public void setDataOffset(int offset)
```

Produce IOException.

Spostare il cursore sulla posizione assoluta nel buffer del messaggio. Questo metodo è un sinonimo di seek() e viene fornito per una compatibilità per tutte le lingue con le altre API di MQSeries.

**getDataOffset**

```
public int getDataOffset()
```

Produce IOException.

Restituire la posizione corrente del cursore all'interno dei dati del messaggio (il punto in cui le operazioni di lettura e scrittura diventano valide).

**clearMessage**

```
public void clearMessage()
```

Produce IOException.

Cancellare tutti i dati contenuti nel buffer del messaggio e impostare l'offset dei dati nuovamente su zero.

**getVersion**

```
public int getVersion()
```

Restituisce la versione della struttura in uso.

**resizeBuffer**

```
public void resizeBuffer(int size)
```

Produce IOException.

Un'indicazione all'oggetto MQMessage sulle dimensioni del buffer che potrebbe essere richiesta per le successive operazioni di tipo get. Se il messaggio contiene al momento dati del messaggio, e le nuove dimensioni sono inferiori rispetto a quelle correnti, i dati del messaggio verranno troncati.

**readBoolean**

```
public boolean readBoolean()
```

Produce IOException.

Leggere un byte dalla posizione corrente nel buffer dei messaggi.

**readChar**

```
public char readChar()
```

Produce IOException, EOFException.

Leggere un carattere Unicode dalla posizione corrente nel buffer dei messaggi.

**readDouble**

```
public double readDouble()
```

Produce IOException, EOFException.

Leggere un carattere double dalla posizione corrente nel buffer dei messaggi. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

I valori di MQC.MQENC_FLOAT_IEEE_NORMAL e MQC.MQENC_FLOAT_IEEE_REVERSED leggono valori doppi standard IEEE rispettivamente nei formati big-endian e little-endian.

Un valore di MQC.MQENC_FLOAT_S390 leggere un numero a virgola mobile del formato System/390.

**readFloat**

```
public float readFloat()
```

Produce IOException, EOFException.

Leggere un carattere a virgola mobile dalla posizione corrente nel buffer dei messaggi. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

I valori di MQC.MQENC_FLOAT_IEEE_NORMAL e MQC.MQENC_FLOAT_IEEE_REVERSED leggono valori a virgola mobile standard IEEE rispettivamente nei formati big-endian e little-endian.

Un valore di MQC.MQENC_FLOAT_S390 leggere un numero a virgola mobile del formato System/390.

**readFully**

```
public void readFully(byte b[])
```

Produce Exception, EOFException.

Riempire la matrice di byte `b` con i dati tratti dal buffer dei messaggi.

**readFully**

```
public void readFully(byte b[],
                      int off,
                      int len)
```

Produce IOException, EOFException.

Riempire gli elementi *len* della matrice di byte b con i dati dal buffer dei messaggi, a partire dall'offset *off*.

**readInt**

```
public int readInt()
```

Produce IOException, EOFException.

Leggere un valore intero dalla posizione corrente nel buffer dei messaggi. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_INTEGER_NORMAL legge un valore intero in formato big-endian, un valore di MQC.MQENC_INTEGER_REVERSED legge un valore intero in formato little-endian.

**readInt4**

```
public int readInt4()
```

Produce IOException, EOFException.

Synonym di readInt(), fornito per la compatibilità API MQSeries per tutte le lingue.

**readLine**

```
public String readLine()
```

Produce IOException.

Converte dal codeset identificato nella variabile del membro characterSet a Unicode, quindi legge in una riga che è stata terminata da \n, \r, \r\n o EOF.

**readLong**

```
public long readLong()
```

Produce IOException, EOFException.

Leggere un carattere long dalla posizione corrente nel buffer dei messaggi. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_INTEGER_NORMAL legge un valore long in formato big-endian, un valore di MQC.MQENC_INTEGER_REVERSED legge un valore long in formato little-endian.

**readInt8**

```
public long readInt8()
```

Produce IOException, EOFException.

Sinonimo di readLong(), fornito per la compatibilità API MQSeries per tutte le lingue.

**readObject**

```
public Object readObject()
```

Produce OptionalDataException, ClassNotFoundException, IOException.

Leggere un oggetto dal buffer dei messaggi. La classe dell'oggetto, la firma della classe e il valore dei campi non transienti e e non statici della classe sono tutti letti.

**readShort**

```
public short readShort()
```

Produce IOException, EOFException.

**readInt2**

```
public short readInt2()
```

Produce IOException, EOFException.

Sinonimo di readShort(), fornito per la compatibilità API MQSeries per tutte le lingue.

**readUTF**

```
public String readUTF()
```

Produce IOException.

Leggere una stringa UTF, preceduta da un campo dalla lunghezza di 2 byte dalla posizione corrente nel buffer dei messaggi.

**readUnsignedByte**

```
public int readUnsignedByte()
```

Produce IOException, EOFException.

Leggere un byte non firmato dalla posizione corrente nel buffer dei messaggi.

**readUnsignedShort**

```
public int readUnsignedShort()
```

Produce IOException, EOFException.

Leggere un carattere short non firmato dalla posizione corrente nel buffer dei messaggi. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_INTEGER_NORMAL legge un valore short non firmato big-endian, un valore di MQC.MQENC_INTEGER_REVERSED legge un valore short non firmato little-endian.

**readUInt2**

```
public int readUInt2()
```

Produce IOException, EOFException.

Sinonimo di readUnsignedShort(), fornito per la compatibilità API MQSeries per tutte le lingue.

**readString**

```
public String readString(int length)
```

Produce IOException, EOFException.

Leggere una stringa nel codeset identificato dalla variabile del membro characterSet e convertirla in Unicode.

**Parametri:**

*length* Il numero di caratteri da leggere che può variare dal numero di byte secondo il codeset, in quanto alcuni codeset utilizzano più di un byte per carattere.

**readDecimal2**

```
public short readDecimal2()
```

Produce IOException, EOFException.

Leggere un numero decimale compresso a 2 byte (-999..999). Il comportamento di questo metodo è controllato dal valore della variabile del membro di codifica. Un valore di MQC.MQENC_DECIMAL_NORMAL legge un numero decimale compresso in formato big-endian e un valore di MQC.MQENC_DECIMAL_REVERSED legge un numero decimale compresso in formato little-endian.

**readDecimal4**

```
public int readDecimal4()
```

Produce IOException, EOFException.

Leggere un numero decimale compresso a 4 byte (-9999999..9999999). Il comportamento di questo metodo è controllato dal valore della variabile del membro di codifica. Un valore di MQC.MQENC_DECIMAL_NORMAL legge un numero decimale compresso in formato big-endian e un valore di MQC.MQENC_DECIMAL_REVERSED legge un numero decimale compresso in formato little-endian.

**readDecimal8**

```
public long readDecimal8()
```

Produce IOException, EOFException.

Leggere un numero decimale compresso a 8 byte (da -999999999999999 a 999999999999999). Il comportamento di questo metodo è controllato dalla variabile del membro di codifica. Un valore di MQC.MQENC_DECIMAL_NORMAL legge un numero decimale compresso in formato big-endian e MQC.MQENC_DECIMAL_REVERSED legge un numero decimale compresso in formato little-endian.

**setVersion**

```
public void setVersion(int version)
```

Specifica la versione della struttura da utilizzare. I valori possibili sono:
- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

Generalmente non è necessario richiamare questo metodo, a meno che non si desideri fare in modo che il client utilizzi una struttura della versione 1 quando è connesso a un gestore code in grado di gestire strutture della versione 2. In tutte le altre situazioni il client determina la versione corretta della struttura da utilizzare interrogando le capacità del gestore code.

**skipBytes**

```
public int skipBytes(int n)
```

Produce IOException, EOFException.

Spostare in avanti n byte nel buffer dei messaggi.

Questo metodo blocca fino a quando non si verifica una delle seguenti situazioni:
- Tutti i byte sono saltati
- Viene rilevata la fine del buffer dei messaggi
- Viene prodotta un'eccezione

Restituisce il numero di byte saltati, che è sempre pari a n.

**write**

```
public void write(int b)
```

Produce IOException.

Scrivere un byte nel buffer dei messaggi nella posizione corrente.

**write**

```
public void write(byte b[])
```

Produce IOException.

Scrivere una matrice di byte nel buffer dei messaggi nella posizione corrente.

**write**

```
public void write(byte b[],
                  int off,
                  int len)
```

Produce IOException.

Scrivere una serie di byte nel buffer dei messaggi nella posizione corrente. *len* byte verranno scritti, presi dall'offset *off* nella matrice b.

**writeBoolean**

```
public void writeBoolean(boolean v)
```

Produce IOException.

Scrivere un valore booleano nel buffer dei messaggi nella posizione corrente.

**writeByte**

```
public void writeByte(int v)
```

Produce IOException.

Scrivere un byte nel buffer dei messaggi nella posizione corrente.

**writeBytes**

```
public void writeBytes(String s)
```

Produce IOException.

Scrive la stringa nel buffer dei messaggi come sequenza di byte. Ogni carattere nella stringa viene scritto in sequenza cancellando gli otto bit alti.

**writeChar**

```
public void writeChar(int v)
```

Produce IOException.

Scrivere un carattere Unicode nel buffer dei messaggi nella posizione corrente.

**writeChars**

```
public void writeChars(String s)
```

Produce IOException.

Scrivere una stringa come sequenza di caratteri Unicode nel buffer dei messaggi nella posizione corrente.

**writeDouble**

```
public void writeDouble(double v)
```

Produce IOException

Scrivere un carattere double nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

I valori di MQC.MQENC_FLOAT_IEEE_NORMAL e MQC.MQENC_FLOAT_IEEE_REVERSED scrivono valori a virgola mobile standard IEEE rispettivamente nei formati Big-endian e Little-endian.

Un valore di MQC.MQENC_FLOAT_S390 scrive un numero di formato a virgola mobile System/390. L'intervallo di valori double IEEE è maggiore dell'intervallo di numeri a virgola mobile di precisione double S/390, pertanto non è possibile convertire numeri molto grandi.

**writeFloat**

```
public void writeFloat(float v)
```

Produce IOException.

Scrivere un carattere a virgola mobile nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

I valori di MQC.MQENC_FLOAT_IEEE_NORMAL e MQC.MQENC_FLOAT_IEEE_REVERSED scrivono valori a virgola mobile standard IEEE rispettivamente nei formati big-endian e little-endian.

Un valore di MQC.MQENC_FLOAT_S390 scriverà un numero di formato a virgola mobile System/390.

**writeInt**

```
public void writeInt(int v)
```

Produce IOException.

Scrivere un carattere intero nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_INTEGER_NORMAL scrive un valore intero in formato big-endian, un valore di MQC.MQENC_INTEGER_REVERSED legge un valore intero in formato little-endian.

**writeInt4**

```
public void writeInt4(int v)
```

Produce IOException.

Sinonimo di writeInt(), fornito per la compatibilità API MQSeries per tutte le lingue.

**writeLong**

```
public void writeLong(long v)
```

Produce IOException.

Scrivere un carattere long nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_INTEGER_NORMAL scrive un valore long in formato big-endian, un valore di MQC.MQENC_INTEGER_REVERSED legge un valore long in formato little-endian.

**writeInt8**

```
public void writeInt8(long v)
```

Produce IOException.

Sinonimo di writeLong(), fornito per la compatibilità API MQSeries per tutte le lingue.

**writeObject**

```
public void writeObject(Object obj)
```

Produce IOException.

Scrivere l'oggetto specificato nel buffer dei messaggi. Vengono scritti: la classe dell'oggetto, la firma della classe, i valori dei campi non transienti e non statici della classe e i relativi supertipi.

**writeShort**

```
public void writeShort(int v)
```

Produce IOException.

Scrivere un carattere short nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_INTEGER_NORMAL scrive un valore short in formato big-endian, un valore di MQC.MQENC_INTEGER_REVERSED legge un valore short in formato little-endian.

**writeInt2**

```
public void writeInt2(int v)
```

Produce IOException.

Sinonimo di writeShort(), fornito per la compatibilità API MQSeries per tutte le lingue.

**writeDecimal2**

```
public void writeDecimal2(short v)
```

Produce IOException.

Scrivere un numero dal formato decimale compresso a 2 byte nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_DECIMAL_NORMAL scrive un valore decimale compresso di tipo big-endian, un valore di MQC.MQENC_DECIMAL_REVERSED scrive un valore decimale compresso little-endian.

**Parametri**

*v* può essere compreso nell'intervallo che va da -999 a 999.

**writeDecimal4**

```
public void writeDecimal4(int v)
```

Produce IOException.

Scrivere un numero dal formato decimale compresso a 4 byte nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_DECIMAL_NORMAL scrive un valore decimale compresso di tipo big-endian, un valore di MQC.MQENC_DECIMAL_REVERSED scrive un valore decimale compresso little-endian.

**Parametri**

*v* può essere nell'intervallo che va da -9999999 a 9999999.

**writeDecimal8**

```
public void writeDecimal8(long v)
```

Produce IOException.

Scrivere un numero dal formato decimale compresso a 8 byte nel buffer dei messaggi nella posizione corrente. Il valore della variabile del membro di codifica determina il comportamento di questo metodo.

Un valore di MQC.MQENC_DECIMAL_NORMAL scrive un valore decimale compresso di tipo big-endian, un valore di MQC.MQENC_DECIMAL_REVERSED scrive un valore decimale compresso little-endian.

**Parametri:**

*v* può essere compreso nell'intervallo che va da -999999999999999 a 999999999999999.

**writeUTF**

```
public void writeUTF(String str)
```

Produce IOException.

Scrivere una stringa UTF, preceduta da un campo dalla lunghezza di 2 byte nel buffer dei messaggi nella posizione corrente .

**writeString**

```
public void writeString(String str)
```

Produce IOException.

Scrivere una stringa nel buffer dei messaggi nella posizione corrente, convertendola nel codeset identificato dalla variabile del membro characterSet.

# MQMessageTracker

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
```

classe astratta pubblica **MQMessageTracker**
estende **Object**

**Nota:** E' possibile utilizzare questa classe solo quando si è connessi a un gestore code MQSeries Versione 5 (o successiva).

Questa classe viene ereditata da MQDistributionListItem (a pagina 95) dove viene utilizzata per personalizzare i parametri dei messaggi per una determinata destinazione in un elenco di distribuzione.

## Variabili

**feedback**

```
public int feedback
```

Viene utilizzata con un messaggio di tipo MQC.MQMT_REPORT per indicare la natura del prospetto. I codici di feedback che seguono sono definiti dal sistema:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

E' possibile utilizzare anche i valori di feedback definiti dall'applicazione nell'intervallo che va da MQC.MQFB_APPL_FIRST a MQC.MQFB_APPL_LAST.

Il valore predefinito di questo campo è MQC.MQFB_NONE, che indica che non viene fornito alcun feedback.

**messageId**

```
public byte messageId[]
```

Specifica l'identificativo del messaggio da utilizzare quando viene inserito il messaggio. Se MQC.MQMI_NONE non è specificata, il gestore code genera un identificativo del messaggio univoco quando il messaggio viene inserito. Il valore della variabile di questo membro viene aggiornato dopo l'operazione di inserimento per indicare che è stato utilizzato l'identificativo del messaggio.

Il valore predefinito è MQC.MQMI_NONE.

**correlationId**

```
public byte correlationId[]
```

Specifica l'identificativo di correlazione da utilizzare quando viene inserito il messaggio.

Il valore predefinito è MQC.MQCI_NONE.

**accountingToken**

```
public byte accountingToken[]
```

Fa parte del contesto di identità del messaggio. Consente all'applicazione di completare il lavoro come conseguenza del corretto caricamento del messaggio.

Il valore predefinito è MQC.MQACT_NONE.

**groupId**

```
public byte[] groupId
```

Una stringa di byte che identifica il gruppo di messaggi a cui appartiene il messaggio fisico.

Il valore predefinito è MQC.MQGI_NONE.

# MQPoolServices

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPoolServices
```

classe pubblica **MQPoolServices**
estende **Object**

**Nota:** Di norma, le applicazioni non utilizzano questa classe.

La classe MQPoolServices può essere utilizzata dalle implementazioni di ConnectionManager destinate a un utilizzo come ConnectionManager predefinito per le connessioni MQSeries.

Un ConnectionManager può costruire un oggetto MQPoolServices e tramite questo oggetto, registrare un listener. Il listener riceve gli eventi relativi alla serie di MQPoolTokens gestite da MQEnvironment. Il ConnectionManager può utilizzare queste informazioni per eseguire tutte le operazioni necessarie di avvio o pulizia.

Vedere anche "MQPoolServicesEvent" a pagina 139 e "MQPoolServicesEventListener" a pagina 172.

## Costruttori

**MQPoolServices**

```
public MQPoolServices()
```

Costruire un nuovo oggetto MQPoolServices.

## Metodi

**addMQPoolServicesEventListener**

```
public void addMQPoolServicesEventListener
                              (MQPoolServicesEventListener listener)
```

Aggiungere una MQPoolServicesEventListener. Il listener riceve un evento ogni volta che un token viene aggiunto o rimosso dalla serie di MQPoolTokens controllate da MQEnvironment oppure ogni volta che il ConnectionManager predefinito cambia.

**removeMQPoolServicesEventListener**

```
public void removeMQPoolServicesEventListener
                                 (MQPoolServicesEventListener listener)
```

Rimuovere una MQPoolServicesEventListener.

**getTokenCount**

```
public int getTokenCount()
```

Restituisce il numero di MQPoolToken al momento registrate con MQEnvironment.

# MQPoolServicesEvent

```
java.lang.Object
   |
   +-- java.util.EventObject
          |
          +-- com.ibm.mq.MQPoolServicesEvent
```

**Nota:** Di norma, le applicazioni non utilizzano questa classe.

Una MQPoolServicesEvent viene generata ogni volta che una MQPoolToken viene aggiunta o rimossa dalla serie di token controllati da MQEnvironment. Un evento viene generato anche quando si modifica il ConnectionManager predefinito.

Vedere anche "MQPoolServices" a pagina 138 e "MQPoolServicesEventListener" a pagina 172.

## Variabili

**TOKEN_ADDED**
`public static final int TOKEN_ADDED`

L'ID evento utilizzato quando una MQPoolToken viene aggiunta alla serie.

**TOKEN_REMOVED**
`public static final int TOKEN_REMOVED`

L'ID evento utilizzato quando una MQPoolToken viene eliminata dalla serie.

**DEFAULT_POOL_CHANGED**
`public static final int DEFAULT_POOL_CHANGED`

L'ID evento utilizzato quando il ConnectionManager predefinito viene modificato.

**ID** `protected int ID`

L'ID evento. I valori validi sono:

TOKEN_ADDED
TOKEN_REMOVED
DEFAULT_POOL_CHANGED

**token** `protected MQPoolToken token`

Il token. Quando l'ID evento è DEFAULT_POOL_CHANGED, risulta null.

## Costruttori

**MQPoolServicesEvent**
`public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)`

Costruisce una MQPoolServicesEvent in base all'ID evento e al token.

**MQPoolServicesEvent**
`public MQPoolServicesEvent(Object source, int eid)`

Costruisce una MQPoolServicesEvent basata sull'ID evento.

## Metodi

**getId** `public int getId()`

Richiama l'ID evento.

**Restituisce**

L'ID evento, con uno dei seguenti valori:

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

**getToken**

`public MQPoolToken getToken()`

Restituisce il token aggiunto o rimosso dalla serie. Se l'ID evento è DEFAULT_POOL_CHANGED, risulta null.

# MQPoolToken

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPoolToken
```

classe pubblica **MQPoolToken**
estende **Object**

Una MQPoolToken può essere utilizzata per abilitare il pool di connessioni predefinito. Le MQPoolToken sono registrate con la classe MQEnvironment prima che un componente dell'applicazione si connetta a MQSeries. Successivamente ne viene annullata la registrazione quando il componente ha terminato di utilizzare MQSeries. Generalmente la ConnectionManager predefinita è attiva mentre la serie di MQPoolToken registrate non è vuota.

MQPoolToken non fornisce metodi né variabili. I provider di ConnectionManager possono scegliere di estendere MQPoolToken in modo che i suggerimenti possono essere trasmessi al ConnectionManager.

Vedere "MQEnvironment.addConnectionPoolToken" a pagina 102 e "MQEnvironment.removeConnectionPoolToken" a pagina 102.

## Costruttori

**MQPoolToken**

```
public MQPoolToken()
```

Costruire un nuovo oggetto MQPoolToken.

# MQProcess

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQProcess
```

classe pubblica **MQProcess**
estende **MQManagedObject**. (Vedere pagina 110.)

MQProcess fornisce operazioni di ricerca per i processi MQSeries.

## Costruttori

**MQProcess**

```
public MQProcess(MQQueueManager qMgr,
                                 String processName,
                                 int openOptions,
                                 String queueManagerName,
                                 String alternateUserId)
                  throws MQException
```

Accedere un processo sul gestore code `qMgr`. Vedere `accessProcess` in "MQQueueManager" a pagina 158 per informazioni dettagliate sui parametri restanti.

## Metodi

**getApplicationId**

```
public String getApplicationId()
```

-Una stringa di caratteri che identifica l'applicazione da avviare. Queste informazioni sono destinate all'utilizzo da parte di un'applicazione di controllo trigger che elabora i messaggi sulla coda di iniziazione. Le informazioni vengono inviate alla coda di iniziazione come parte del messaggio trigger.

Produce MQException se il metodo viene richiamato dopo aver chiuso il processo.

**getApplicationType**

```
public int getApplicationType()
```

Produce MQException (vedere pagina 103).

In questo viene identificata la natura del programma da avviare in risposta alla ricevuta di un messaggio trigger. Il tipo di applicazione può assumere qualsiasi valore, ma per i tipi standard si consigliano i seguenti valori:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT

- MQC.MWQAT_USER_FIRST (il valore più basso per il tipo di applicazione definito dall'utente)
- MQC.MQAT_USER_LAST (il valore più alto per il tipo di applicazione definito dall'utente)

**getEnvironmentData**

```
public String getEnvironmentData()
```

Produce MQException.

Una stringa contenente informazioni relative all'ambiente che riguarda l'applicazione da avviare.

**getUserData**

```
public String getUserData()
```

Produce MQException.

Una stringa contenente informazioni sull'utente relative all'applicazione da avviare.

**close**

```
public synchronized void close()
```

Produce MQException.

In sostituzione di "MQManagedObject.close" a pagina 112.

# MQPutMessageOptions

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPutMessageOptions
```

classe pubblica **MQPutMessageOptions**
estende **Object**

Questa classe contiene opzioni che controllano il comportamento di MQQueue.put().

**Nota:** Il comportamento di alcune delle opzioni disponibili in questa classe dipende dall'ambiente in cui vengono utilizzate. Questi elementi sono contrassegnati con un *. Consultare la sezione "Estensioni Versione 5 in funzione in altri ambienti" a pagina 83 per informazioni più dettagliate.

## Variabili

**options**

```
public int options
```

Opzioni che controllano l'azione di MQQueue.put. E' possibile specificare tutti o nessuno dei valori che seguono. Se è richiesta più di un'opzione, i valori possono essere aggiunti o combinati utilizzando l'operatore bitwise OR.

**MQC.MQPMO_SYNCPOINT**
Inserire un messaggio con controllo syncpoint. Il messaggio non è visibile al di fuori dell'unità di lavoro fino al commit di quest'ultima. Se l'unità di lavoro viene respinta, il messaggio viene eliminato.

**MQC.MQPMO_NO_SYNCPOINT**
Inserire un messaggio senza controllo syncpoint. Se l'opzione di controllo syncpoint non è specificata, si presuppone un'opzione predefinita di 'no syncpoint', valida per tutte le piattaforme supportate, compreso OS/390.

**MQC.MQPMO_NO_CONTEXT**
Nessun contesto deve essere associato al messaggio.

**MQC.MQPMO_DEFAULT_CONTEXT**
Associare il contesto predefinito al messaggio.

**MQC.MQPMO_SET_IDENTITY_CONTEXT**
Impostare il contesto dell'identità dall'applicazione.

**MQC.MQPMO_SET_ALL_CONTEXT**
Impostare tutto il contesto dall'applicazione.

**MQC.MQPMO_FAIL_IF_QUIESCING**
L'operazione non riesce se il gestore code è in fase di chiusura.

**MQC.MQPMO_NEW_MSG_ID***
Generare un nuovo id del messaggio per ogni messaggio inviato.

**MQC.MQPMO_NEW_CORREL_ID***
Generare un nuovo id della correlazione per ogni messaggio inviato.

**MQC.MQPMO_LOGICAL_ORDER***
Inserire messaggi logici e segmenti nei gruppi di messaggi nel loro ordine logico.

**MQC.MQPMO_NONE**
Nessuna opzione specificata. Da non utilizzare insieme ad altre opzione.

**MQC.MQPMO_PASS_IDENTITY_CONTEXT**
Passare il contesto dell'identità da un handle della coda di input.

**MQC.MQPMO_PASS_ALL_CONTEXT**
Passare tutto il contesto da un handle della coda di input.

**contextReference**
`public MQQueue ContextReference`

Questo è un campo di immissione che indica la fonte delle informazioni sul contesto.

Se il campo `options` include MQC.MQPMO_PASS_IDENTITY_CONTEXT o MQC.MQPMO_PASS_ALL_CONTEXT, impostare questo campo per fare riferimento alla MQQueue da cui le informazioni sul contesto dovrebbero essere tratte.

Il valore iniziale di questo campo è null.

**recordFields ***
`public int recordFields`

Flag che indicano i campi da personalizzare in base alla coda durante l'inserimento di un messaggio in un elenco di distribuzione. E' possibile specificare uno o più flag tra quelli che seguono:

**MQC.MQPMRF_MSG_ID**
Utilizzare l'attributo messageId nella MQDistributionListItem.

**MQC.MQPMRF_CORREL_ID**
Utilizzare l'attributo correlationId nella MQDistributionListItem.

**MQC.MQPMRF_GROUP_ID**
Utilizzare l'attributo groupId nella MQDistributionListItem.

**MQC.MQPMRF_FEEDBACK**
Utilizzare l'attributo feedback nella MQDistributionListItem.

**MQC.MQPMRF_ACCOUNTING_TOKEN**
Utilizzare l'attributo accountingToken nella MQDistributionListItem.

Il valore speciale MQC.MQPMRF_NONE indica che non è necessario personalizzare alcun campo.

**resolvedQueueName**
`public String resolvedQueueName`

Questo è un campo di output impostato dal gestore code sul nome della coda in cui è inserito il messaggio. Può essere diverso dal nome utilizzato per aprire la coda se la coda aperta era una coda alias o modello.

**resolvedQueueManagerName**
`public String resolvedQueueManagerName`

Questo è un campo di output impostato dal gestore code sul nome del gestore code che possiede la coda specificata dal nome della coda remota.

Può essere diverso dal nome del gestore code da cui si accede alla coda se la coda è una coda remota.

**knownDestCount ***

```
public int knownDestCount
```

Questo è un campo di output impostato dal gestore code sul numero di messaggi che la chiamata corrente ha inviato correttamente alle code che risolvono nelle code locali. Questo campo è impostato anche all'apertura di una singola coda che non fa parte di un elenco di distribuzione.

**unknownDestCount ***

```
public int unknownDestCount
```

Questo è un campo di output impostato dal gestore code sul numero di messaggi che la chiamata corrente ha inviato correttamente alle code che risolvono nelle code remote. Questo campo è impostato anche all'apertura di una singola coda che non fa parte di un elenco di distribuzione.

**invalidDestCount ***

```
public int invalidDestCount
```

Questo è un campo di output impostato dal gestore code sul numero di messaggi che non è stato possibile inviare alle code in un elenco di distribuzione. Il conteggio include le code che non è stato possibile aprire, nonché le code che sono state aperte correttamente, ma per le quali l'operazione di inserimento non è riuscita. Questo campo è impostato anche all'apertura di una singola coda che non fa parte di un elenco di distribuzione.

## Costruttori

**MQPutMessageOptions**

```
public MQPutMessageOptions()
```

Costruire un nuovo oggetto MQPutMessageOptions senza alcuna opzione impostata e un resolvedQueueName e un resolvedQueueManagerName vuoto.

# MQQueue

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
            |
            +-- com.ibm.mq.MQQueue
```

classe pubblica **MQQueue**
estende **MQManagedObject**. (Vedere pagina 110.)

MQQueue fornisce operazioni di tipo inquire, set, put e get per le code MQSeries. Le capacità inquire e set sono ereditate da MQ.MQManagedObject.

Vedere anche la sezione “MQQueueManager.accessQueue” a pagina 163.

## Costruttori

**MQQueue**

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
               String alternateUserId )
               throws MQException
```

Accedere a una coda sul gestore code `qMgr`.

Vedere “MQQueueManager.accessQueue” a pagina 163 per informazioni dettagliate sui metodi restanti.

## Metodi

**get**

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions,
                             int MaxMsgSize)
```

Throws MQException.

Richiama un messaggio dalla coda fino a una dimensione massima dei messaggi specificata.

Questo metodo prende un oggetto MQMessage come parametro. Utilizza alcuni dei campi nell’oggetto come parametri di input, in particolare messageId e correlationId, pertanto è importante assicurarsi che che siano impostati nel modo richiesto. (Vedere “Message” a pagina 290.)

Se l’operazione di richiamo non riesce, l’oggetto MQMessage resta immutato. Se invece l’operazione riesce, il descrittore del messaggio (variabili dei membri) e le porzioni dei dati del messaggio della MQMessage vengono completamente sostituite dal descrittore dei messaggi e dai dati dei messaggi provenienti dal messaggio in arrivo.

Tutte le chiamate a MQSeries da una determinata MQQueueManager sono sincrone. Pertanto se si esegue un’operazione di tipo get con attesa, a tutti gli altri thread che utilizzano la stessa MQQueueManager viene impedito di effettuare ulteriori chiamate MQSeries fino al completamento

dell'operazione get. Se è necessario che più thread accedano a MQSeries contemporaneamente, ogni thread deve creare il proprio oggetto MQQueueManager.

**Parametri**

*messaggio*
Un parametro di input/output contenente informazioni sul descrittore del messaggio e i dati sul messaggio restituiti.

*getMessageOptions*
Opzioni che controllano l'operazione get. (Vedere "MQGetMessageOptions" a pagina 105.)

*MaxMsgSize*
Il messaggio più grande che questa chiamata sarà in grado di ricevere. Se il messaggio sulla coda supera questa dimensione, può verificarsi uno dei due fenomeni:

1. Se il flag MQC.MQGMO_ACCEPT_TRUNCATED_MSG è impostato nella variabile dei membri delle opzioni dell'oggetto MQGetMessageOptions, il messaggio viene riempito con il maggior numero di dati del messaggio che la dimensione del buffer specificato può contenere e viene prodotta un'eccezione con il codice di completamento MQException.MQCC_WARNING e il codice motivo MQException.MQRC_TRUNCATED_MSG_ACCEPTED.
2. Se il flag MQC.MQGMO_ACCEPT_TRUNCATED_MSG non è impostato, il messaggio viene lasciato sulla coda e viene prodotta una MQException con il codice di completamento MQException.MQCC_WARNING e il codice motivo MQException.MQRC_TRUNCATED_MSG_FAILED.

Produce MQException se l'operazione get non riesce.

**get**

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions)
```

Produce MQException.

Richiama un messaggio dalla coda, indipendentemente dalla dimensione. Per i messaggi di grandi dimensioni, è probabile che il metodo get esegua due chiamate a MQSeries da parte dell'utente, uno per stabilire la dimensione del buffer richiesto e uno per richiamare i dati stessi del messaggio.

Questo metodo prende un oggetto MQMessage come parametro. Utilizza alcuni dei campi nell'oggetto come parametri di input, in particolare messageId e correlationId, pertanto è importante assicurarsi che che siano impostati nel modo richiesto. (Vedere "Message" a pagina 290.)

Se l'operazione di richiamo non riesce, l'oggetto MQMessage resta immutato. Se invece l'operazione riesce, il descrittore del messaggio (variabili dei membri) e le porzioni dei dati del messaggio della MQMessage vengono completamente sostituite dal descrittore dei messaggi e dai dati dei messaggi provenienti dal messaggio in arrivo.

Tutte le chiamate a MQSeries da una determinata MQQueueManager sono sincrone. Pertanto se si esegue un'operazione di tipo get con attesa, a tutti gli altri thread che utilizzano la stessa MQQueueManager viene impedito di effettuare ulteriori chiamate MQSeries fino al completamento

dell'operazione get. Se è necessario che più thread accedano a MQSeries contemporaneamente, ogni thread deve creare il proprio oggetto MQQueueManager.

**Parametri**

*messaggio*
Un parametro di input/output contenente informazioni sul descrittore del messaggio e i dati sul messaggio restituiti.

*getMessageOptions*
Opzioni che controllano l'operazione get. (Vedere "MQGetMessageOptions" a pagina 105 per informazioni dettagliate.)

Produce MQException se l'operazione get non riesce.

**get**

```
public synchronized void get(MQMessage message)
```

Questa è una versione semplificata del metodo get precedentemente descritto.

**Parametri**

*MQMessage*
Un parametro di input/output contenente informazioni sul descrittore del messaggio e i dati sul messaggio restituiti.

Questo metodo utilizza un'istanza predefinita di MQGetMessageOptions per eseguire l'operazione get. L'opzione del messaggio utilizzata è MQGMO_NOWAIT.

**put**

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions )
```

Produce MQException.

Inserisce un messaggio nella coda.

Questo metodo prende un oggetto MQMessage come parametro. Le proprietà del descrittore del messaggio di questo oggetto può essere modificato come risultato di questo metodo. I valori ottenuti subito dopo il completamento di questo metodo sono i valori inseriti nella coda MQSeries.

Le modifiche all'oggetto MQMessage al termine dell'inserimento non influiscono sul messaggio che si trova nella coda MQSeries.

Un'operazione di tipo put aggiorna messageId e correlationId. E' necessario tenerne conto nell'eseguire ulteriori chiamate di tipo put/get utilizzando lo stesso oggetto MQMessage. Inoltre il richiamo di put non cancella i dati dei messaggi. pertanto:

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

inserisce due messaggi. Il primo contiene ″a″ e il secondo ″ab″.

**Parametri**

*message*
Message Buffer contenente i dati di Message Descriptor e il messaggio da inviare.

*putMessageOptions*
Opzioni che controllano l'azione di inserimento. (Vedere "MQPutMessageOptions" a pagina 144)

Produce MQException se l'inserimento non riesce.

**put**

```
public synchronized void put(MQMessage message)
```

Questa è una versione semplificata del metodo put precedentemente descritto.

**Parametri**

*MQMessage*
Message Buffer contenente i dati di Message Descriptor e il messaggio da inviare.

Questo metodo utilizza un'istanza predefinita di MQPutMessageOptions per eseguire l'operazione put.

**Nota:** Tutti i metodi che seguono producono MQException se il metodo viene richiamato dopo aver chiuso la coda.

**getCreationDateTime**

```
public GregorianCalendar getCreationDateTime()
```

Produce MQException.

La data e l'ora di creazione di questa coda.

**getQueueType**

```
public int getQueueType()
```

Produce MQException

**Restituisce**
Il tipo di questa coda con uno dei seguenti valori:
- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

**getCurrentDepth**

```
public int getCurrentDepth()
```

Produce MQException.

Richiamare il numero di messaggi al momento presenti nella coda. Questo valore viene incrementato durante una chiamata di put o il rifiuto di una chiamata get. Viene diminuito durante un get non-browse e il rifiuto di una chiamata put.

**getDefinitionType**

```
public int getDefinitionType()
```

Throws MQException.

Indica in che modo la coda è stata definita.

**Restituisce**

Una delle seguenti opzioni:

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

**getMaximumDepth**

```
public int getMaximumDepth()
```

Produce MQException.

Il numero massimo di messaggi che possono esistere sulla coda in un determinato momento. Un tentativo di inserire un messaggio in na coda che contiene già questo numero di messaggi non riesce con un codice motivo MQException.MQRC_Q_FULL.

**getMaximumMessageLength**

```
public int getMaximumMessageLength()
```

Throws MQException.

Questa è la lunghezza massima dei dati dell'applicazione che può esistere in ciascun messaggio della coda. Un tentativo di inserire un messaggio più grande di questo valore non riesce con un codice motivo MQException.MQRC_MSG_TOO_BIG_FOR_Q.

**getOpenInputCount**

```
public int getOpenInputCount()
```

Produce MQException.

Il numero di handle al momento validi per la rimozione dei messaggi dalla coda. Questo è il numero *totale* degli handle in questione noti al gestore code locale, non solo i quelli creati da MQSeries classi per Java (utilizzando accessQueue).

**getOpenOutputCount**

```
public int getOpenOutputCount()
```

Produce MQException.

Il numero di handle al momento validi per l'aggiunta dei messaggi dalla coda. Questo è il numero *totale* degli handle in questione noti al gestore code locale, non solo i quelli creati da MQSeries classi per Java (utilizzando accessQueue).

**getShareability**

```
public int getShareability()
```

Produce MQException.

Indica se la coda può essere aperta più volte per immissioni.

**Restituisce**

Una delle seguenti opzioni:
- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

**getInhibitPut**

```
public int getInhibitPut()
```

Produce MQException.

Indica se le operazioni di tipo put sono consentite o meno per questa coda.

**Restituisce**

Una delle seguenti opzioni:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

**setInhibitPut**

```
public void setInhibitPut(int inhibit)
```

Throws MQException.

Controlla se le operazioni di tipo put sono consentite o meno per questa coda. I valori consentiti sono:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

**getInhibitGet**

```
public int getInhibitGet()
```

Produce MQException.

Indica se le operazioni di tipo get sono consentite o meno per questa coda.

**Restituisce**

I valori possibili sono:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

**setInhibitGet**

```
public void setInhibitGet(int inhibit)
```

Throws MQException.

Controlla se le operazioni di tipo get sono consentite o meno per questa coda. I valori consentiti sono:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

**getTriggerControl**

```
public int getTriggerControl()
```

Produce MQException.

Indica se i messaggi trigger vengono scritti o meno in una coda di iniziazione per determinare l'avvio di un'applicazione in riferimento al funzionamento della coda.

**Restituisce**

I valori possibili sono:
- MQC.MQTC_OFF
- MQC.MQTC_ON

**setTriggerControl**

```
public void setTriggerControl(int trigger)
```

Throws MQException.

Controlla se i messaggi trigger vengono scritti o meno in una coda di iniziazione per determinare l'avvio di un'applicazione in riferimento al funzionamento della coda. I valori consentiti sono:
- MQC.MQTC_OFF
- MQC.MQTC_ON

**getTriggerData**

```
public String getTriggerData()
```

Produce MQException.

I dati di formato libero che il gestore code inserisce nel messaggio trigger quando un messaggio in arrivo su questa coda determina la scrittura di un messaggio trigger sulla coda di iniziazione.

**setTriggerData**

```
public void setTriggerData(String data)
```

Throws MQException.

Imposta i dati di formato libero che il gestore code inserisce nel messaggio trigger quando un messaggio in arrivo su questa coda determina la scrittura di un messaggio trigger sulla coda di iniziazione. La lunghezza consentita massima della stringa è determinata da MQC.MQ_TRIGGER_DATA_LENGTH.

**getTriggerDepth**

```
public int getTriggerDepth()
```

Produce MQException.

Il numero di messaggi che devono essere presenti sulla coda prima che un messaggio trigger venga scritto quando il tipo di trigger è impostato su MQC.MQTT_DEPTH.

**setTriggerDepth**

```
public void setTriggerDepth(int depth)
```

Throws MQException.

Imposta il numero di messaggi che devono essere presenti sulla coda prima che un messaggio trigger venga scritto quando il tipo di trigger è impostato su MQC.MQTT_DEPTH.

**getTriggerMessagePriority**

```
public int getTriggerMessagePriority()
```

Throws MQException.

Questa è la priorità al di sotto della quale i messaggi non contribuiscono alla creazione di messaggi trigger. In altre parole, il gestore code ignora questi messaggi quando decide se è necessario generare un trigger. Un valore pari a zero fa sì che tutti i messaggi contribuiscano alla generazione di messaggi trigger.

**setTriggerMessagePriority**

```
public void setTriggerMessagePriority(int priority)
```

Produce MQException.

Imposta la priorità al di sotto della quale i messaggi non contribuiscono alla creazione di messaggi trigger. In altre parole, il gestore code ignora questi messaggi quando decide se è necessario generare un trigger. Un valore pari a zero fa sì che tutti i messaggi contribuiscano alla generazione di messaggi trigger.

**getTriggerType**

```
public int getTriggerType()
```

Produce MQException.

Le condizioni in base alle quali i messaggi trigger vengono scritti in conseguenza dell'arrivo di messaggi su questa coda.

**Restituisce**

I valori possibili sono:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

**setTriggerType**

```
public void setTriggerType(int type)
```

Throws MQException.

Imposta le condizioni in base alle quali i messaggi trigger vengono scritti in conseguenza dell'arrivo di messaggi su questa coda. I valori possibili sono:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

**close**

```
public synchronized void close()
```

Produce MQException.

In sostituzione di “MQManagedObject.close” a pagina 112.

# MQQueueManager

```
java.lang.Object
   |
   +- com.ibm.mq.MQManagedObject
         |
         +- com.ibm.mq.MQQueueManager
```

classe pubblica **MQQueueManager**
estende **MQManagedObject**. (Vedere pagina 110.)

**Nota:** Il comportamento di alcune delle opzioni disponibili in questa classe dipende dall'ambiente in cui vengono utilizzate. Questi elementi sono contrassegnati con un *. Consultare la sezione "Capitolo 8. Comportamento dipendente dall'ambiente" a pagina 81 per informazioni più dettagliate.

## Variabili

**isConnected**

```
public boolean isConnected
```

True se la connessione al gestore code è ancora aperta.

## Costruttori

**MQQueueManager**

```
public MQQueueManager(String queueManagerName)
```

Throws MQException.

Creare una connessione al gestore code in questione.

**Nota:** Durante l'utilizzo di MQSeries classi per Java, il nome host, il nome del canale e la porta da utilizzare durante la richiesta di connessione sono specificati nella classe MQEnvironment. Questa operazione deve essere effettuata *prima* di richiamare il costruttore.

Nell'esempio che segue viene illustrata una connessione a un gestore code "MYQM", in esecuzione su una macchina con il nome host fred.mq.com.

```
MQEnvironment.hostname = "fred.mq.com";  // host to connect to
MQEnvironment.port     = 1414;           // port to connect to.
                                         // If I don't set this,
                                         // it defaults to 1414
                                         // (the default MQSeries port)
MQEnvironment.channel  = "channel.name"; // the CASE-SENSITIVE
                                         //  name of the
                                         // SVR CONN channel on
                                         // the queue manager
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

Se il nome del gestore code viene lasciato vuoto (null o ""), verrà effettuata una connessione al gestore code predefinito.

Vedere anche la sezione "MQEnvironment" a pagina 97.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager cxManager)
```

Throws MQException.

Questo costruttore si connette al gestore code specificato, utilizzando le proprietà in MQEnvironment. Il MQConnectionManager specificato gestisce la connessione.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager cxManager)
```

Throws MQException.

Questo costruttore si connette al gestore code specificato, utilizzando le proprietà in MQEnvironment. Il ConnectionManager specificato gestisce la connessione.

Questo metodo presuppone l'installazione di una JVM su Java 2 v1.3 o versione successiva, con JAAS 1.0 o versione successiva.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      int options)
```

Throws MQException.

Questa versione del costruttore è destinata all'utilizzo solo in modalità binding e utilizza l'API della connessione estesa (MQCONNX) per connettersi al gestore code. Il parametro *options* consente di scegliere binding veloci o normali. I valori possibili sono:

- MQC.MQCNO_FASTPATH_BINDING per i binding veloci *.
- MQC.MQCNO_STANDARD_BINDING per i binding normali.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      int options,
                      MQConnectionManager cxManager)
```

Throws MQException.

Questo costruttore esegue un MQCONNX, che passa le opzioni fornite. Il MQConnectionManager specificato gestisce la connessione.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

Throws MQException.

Questo costruttore esegue un MQCONNX, che passa le opzioni fornite. Il ConnectionManager specificato gestisce la connessione.

Questo metodo presuppone l'installazione di una JVM su Java 2 v1.3 o versione successiva, con JAAS 1.0 o versione successiva.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      java.util.Hashtable properties)
```

Il parametro delle proprietà prende una serie di coppie chiave/valore che descrivono l'ambiente MQSeries relativo a questo determinato gestore code. Tali proprietà, laddove specificato, sostituiscono i valori impostati dalla classe MQEnvironment e consentono l'impostazione delle singole proprietà su un gestore code in base al gestore code. Consultare il ""MQEnvironment.properties"" a pagina 99.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      MQConnectionManager cxManager)
```

Throws MQException.

Questo costruttore si connette al gestore code specificato, utilizzando l'Hashtable delle proprietà fornita per sostituirle in MQEnvironment. Il MQConnectionManager specificato gestisce la connessione.

**MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      ConnectionManager cxManager)
```

Throws MQException.

Questo costruttore si connette al gestore code specificato, utilizzando l'Hashtable delle proprietà fornita per sostituirle in MQEnvironment. Il ConnectionManager specificato gestisce la connessione.

Questo metodo presuppone l'installazione di una JVM Java 2 v1.3 o versione successiva, con JAAS 1.0 o versione successiva.

## Metodi

**getCharacterSet**

```
public int getCharacterSet()
```

Throws MQException.

Restituisce il CCSID (Coded Character Set Identifier) del codeset del gestore code. Esso definisce il set dei caratteri utilizzati dal gestore code per tutti i campi della stringa di caratteri nell'API.

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**getMaximumMessageLength**

```
public int getMaximumMessageLength()
```

Throws MQException.

Restituisce la lunghezza massima di un messaggio (in byte) che può essere gestita dal gestore code. Non è possibile definire una lunghezza massima dei messaggi superiore a questa in nessuna coda.

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**getCommandLevel**

```
public int getCommandLevel()
```

Throws MQException.

Indica il livello dei comandi di controllo del sistema supportati dal gestore code. La serie di comandi di controllo del sistema che corrispondono a un determinato livello di comandi varia in base all'architettura della piattaforma su cui è in esecuzione il gestore code. Consultare la documentazione di MQSeries per ulteriori informazioni.

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**Restituisce**

Una delle costanti MQC.MQCMDL_LEVEL_xxx

**getCommandInputQueueName**

```
public String getCommandInputQueueName()
```

Throws MQException.

Restituisce il nome della coda di input comandi definita sul gestore code. Si tratta di una coda a cui le applicazioni possono inviare comandi, se autorizzate a eseguire questa operazione.

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**getMaximumPriority**

```
public int getMaximumPriority()
```

Throws MQException.

Restituisce la priorità dei messaggi massima supportata dal gestore code. Le priorità vanno da zero (il valore più basso) a questo valore.

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**getSyncpointAvailability**

```
public int getSyncpointAvailability()
```

Throws MQException.

Indica se il gestore code supporta unità di lavoro e operazioni di syncpoint con i metodi MQQueue.get e MQQueue.put.

**Restituisce**

- MQC.MQSP_AVAILABLE se è disponibile il syncpointing.
- MQC.MQSP_AVAILABLE se non è disponibile il syncpointing.

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**getDistributionListCapable**

```
public boolean getDistributionListCapable()
```

Indica se il gestore code supporta le liste di distribuzione.

**disconnect**

```
public synchronized void disconnect()
```

Throws MQException.

Termina la connessione al gestore code. Tutte le code aperte e i processi a cui accede il gestore code in questione vengono chiusi e diventano quindi inutilizzabili. Una volta disconnessi dal gestore code, l'unico modo per riconnettersi consiste nel creare un nuovo oggetto MQQueueManager.

Generalmente viene eseguito il commit di tutti i lavori eseguiti come parte di un'unità di lavoro. Tuttavia, se questa connessione è gestita da un ConnectionManager, piuttosto che da un MQConnectionManager, è possibile che venga eseguito il rollback dell'unità di lavoro.

**commit**

```
public synchronized void commit()
```

Throws MQException.

Il richiamo di questo metodo indica al gestore code che l'applicazione ha raggiunto un syncpoint e che tutte le operazioni di ottenimento e inserimento dei messaggi effettuate dall'ultimo syncpoint devono essere rese permanenti, I messaggi inseriti come parte di un'unità di lavoro (con il flag MQC.MQPMO_SYNCPOINT impostato nel campo delle opzioni di MQPutMessageOptions) sono rese disponibili a altre applicazioni. I messaggi richiamati come parte di un'unità di lavoro (con il flag MQC.MQGMO_SYNCPOINT impostato nel campo delle opzioni di MQGetMessageOptions) verranno eliminati.

Vedere anche la descrizione di ″backout″ che segue.

**backout**

```
public synchronized void backout()
```

Throws MQException.

Il richiamo di questo metodo indica al gestore code che tutte le operazioni di ottenimento e inserimento dei messaggi effettuate dall'ultimo syncpoint devono essere rifiutate. I messaggi inseriti come parte di un'unità di lavoro (con il flag MQC.MQPMO_SYNCPOINT impostato nel campo delle opzioni di MQPutMessageOptions) vengono eliminati. I messaggi richiamati come parte di un'unità di lavoro (con il flag MQC.MQGMO_SYNCPOINT impostato nel campo delle opzioni di MQGetMessageOptions) vengono immessi nuovamente sulla coda.

Vedere anche la descrizione di ″commit″ sopra riportata.

**accessQueue**

```
public synchronized MQQueue accessQueue
                          (
                          String queueName, int openOptions,
                          String queueManagerName,
                          String dynamicQueueName,
                          String alternateUserId
                          )
```

Throws MQException.

Stabilisce l'accesso a una coda di MQSeries su questo gestore code per ottenere, consultare o inserire i messaggi, richiedere informazioni sugli attributi della coda o impostarne gli attributi.

Se la coda in questione è una coda del modello, verrà creata una coda locale dinamica. Il nome della coda creata può essere determinato esaminando l'attributo `name` dell'oggetto MQQueue restituito.

**Parametri**

*queueName*
Nome della coda da aprire.

*openOptions*
Opzioni che controllano l'apertura della coda. Le opzioni valide sono:

**MQC.MQOO_BROWSE**
Aprire per consultare i messaggi.

**MQC.MQOO_INPUT_AS_Q_DEF**
Aprire per richiamare i messaggi utilizzando l'impostazione predefinita definita dalla coda.

**MQC.MQOO_INPUT_SHARED**
Aprire per richiamare i messaggi con l'accesso condiviso.

**MQC.MQOO_INPUT_EXCLUSIVE**
Aprire per richiamare i messaggi con l'accesso esclusivo.

**MQC.MQOO_OUTPUT**
Aprire per inserire i messaggi.

**MQC.MQOO_INQUIRE**
Aperto per esaminare più attentamente - necessario se si desidera interrogare le proprietà.

**MQC.MQOO_SET**
Aprire per impostare gli attributi.

**MQC.MQOO_SAVE_ALL_CONTEXT**
Salvare il contesto quando il messaggio viene richiamato*.

**MQC.MQOO_SET_IDENTITY_CONTEXT**
Consente l'impostazione del contesto dell'identità.

**MQC.MQOO_SET_ALL_CONTEXT**
Consente l'impostazione di tutto il contesto.

**MQC.MQOO_ALTERNATE_USER_AUTHORITY**
Convalidare con l'identificativo utente specificato.

**MQC.MQOO_FAIL_IF_QUIESCING**
L'operazione non riesce se il gestore code è in fase di chiusura.

**MQC.MQOO_BIND_AS_QDEF**
Utilizzare il binding predefinito per la coda.

**MQC.MQOO_BIND_ON_OPEN**
Eseguire il bind dell'handle alla destinazione all'apertura della coda.

**MQC.MQOO_BIND_NOT_FIXED**
Non eseguire il bind a una destinazione specifica.

**MQC.MQOO_PASS_ALL_CONTEXT**
Consente la trasmissione di tutto il contesto.

**MQC.MQOO_PASS_IDENTITY_CONTEXT**
Consente la trasmissione di tutto il contesto dell'identità.

Se sono richieste più opzioni, i valori possono essere aggiunti insieme o combinati utilizzando l'operatore OR. Fare riferimento a MQSeries *MQSeries Application Programming Reference* per una descrizione più completa di queste opzioni.

*queueManagerName*
Nome del gestore code su cui è definita la coda. Un nome interamente vuoto o null denota il gestore code a cui questo oggetto MQQueueManager è connesso.

*dynamicQueueName*
Questo parametro viene ignorato a meno che queueName specifichi il nome di una coda modello. In caso affermativo, questo parametro specifica il nome della coda dinamica da creare. Un nome vuoto o null non è valido se queueName specifica il nome della coda modello. Se l'ultimo carattere non vuoto nel nome è un asterisco (*), il gestore code sostituisce l'asterisco con una stringa di caratteri per garantire che il nome generato per la coda sia univoco su questo gestore code.

*alternateUserId*
Se MQOO_ALTERNATE_USER_AUTHORITY è specificato nel parametro openOptions, quest'ultimo specifica l'identificativo utente alternativo utilizzato per verificare l'apertura dell'autorizzazione. Se MQOO_ALTERNATE_USER_AUTHORITY non è specificato, questo parametro può essere lasciato vuoto (o null).

**Restituisce**
MQQueue che è stato correttamente aperto.

Produce MQException se l'apertura non riesce.

Vedere anche la sezione ""accessProcess"" a pagina 165.

**accessQueue**

```
public synchronized MQQueue accessQueue
                          (
                          String queueName,
                          int openOptions
                          )
```

Produce MQException se il metodo viene richiamato dopo la disconnessione dal gestore code.

**Parametri**

*queueName*
Nome della coda da aprire

*openOptions*
Opzioni che controllano l'apertura della coda

Consultare la sezione "MQQueueManager.accessQueue" a pagina 163 per informazioni più dettagliate sui parametri.

*queueManagerName*, *dynamicQueueName* e *alternateUserId* sono impostati su *""*.

**accessProcess**

```
public synchronized MQProcess accessProcess
                           (
                         String processName,
                         int openOptions,
                         String queueManagerName,
                         String alternateUserId
                         )
```

Throws MQException.

Stabilisce l'accesso a un processo MQSeries su questo gestore code per ottenere ulteriori informazioni sugli attributi del processo.

**Parametri**

*processName*
Nome del processo da aprire.

*openOptions*
Opzioni che controllano l'apertura del processo. La ricerca di ulteriori informazioni viene aggiunta automaticamente alle opzioni specificate, in modo che non sia necessario specificarla esplicitamente.

Le opzioni valide sono:

**MQC.MQOO_ALTERNATE_USER_AUTHORITY**
Convalidare con l'id utente specificato

**MQC.MQOO_FAIL_IF_QUIESCING**
L'operazione non riesce se il gestore code è in fase di chiusura

Se sono richieste più opzioni, i valori possono essere aggiunti insieme o combinati utilizzando l'operatore OR. Fare riferimento a *MQSeries Application Programming Reference* per una descrizione più completa di queste opzioni.

*queueManagerName*
Nome del gestore code su cui è definito il processo. Le applicazioni dovrebbero lasciare questo parametro vuoto o null.

*alternateUserId*
Se MQOO_ALTERNATE_USER_AUTHORITY è specificato nel parametro openOptions, quest'ultimo specifica l'identificativo utente alternativo utilizzato per verificare l'apertura dell'autorizzazione. Se MQOO_ALTERNATE_USER_AUTHORITY non è specificato, questo parametro può essere lasciato vuoto (o null).

**Restituisce**
MQProcess che è stato correttamente aperto.

Produce MQException se l'apertura non riesce.

Vedere anche la sezione "MQQueueManager.accessQueue" a pagina 163.

**accessProcess**
Questa è una versione semplificata del metodo AccessProcess precedentemente descritto.

```
public synchronized MQProcess accessProcess
                      (
                     String processName,
                     int openOptions
                     )
```

Questa è una versione semplificata del metodo AccessQueue precedentemente descritto.

**Parametri**

*processName*
Il nome del processo da aprire.

*openOptions*
Opzioni che controllano l'apertura del processo.

Fare riferimento a ""accessProcess"" a pagina 165 per una descrizione dettagliata delle opzioni.

*ueueManagerName* e *alternateUserId* sono impostati su "".

**accessDistributionList**

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems, int openOptions,
           String alternateUserId
          )
```

Throws MQException.

**Parametri**

*litems* Le voci da includere nell'elenco di distribuzione.

*openOptions*
Opzioni che controllano l'apertura dell'elenco di distribuzione.

*alternateUserId*
Se MQOO_ALTERNATE_USER_AUTHORITY è specificato nel parametro openOptions, quest'ultimo specifica l'identificativo utente alternativo utilizzato per verificare l'apertura dell'autorizzazione. Se MQOO_ALTERNATE_USER_AUTHORITY non è specificato, questo parametro può essere lasciato vuoto (o null).

**Restituisce**
Un MQDistributionList appena creato aperto e pronto per operazioni di inserimento.

Produce MQException se l'apertura non riesce.

Vedere anche la sezione "MQQueueManager.accessQueue" a pagina 163.

**accessDistributionList**
Questa è una versione semplificata del metodo AccessDistributionList precedentemente descritto.

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems,
           int openOptions,
          )
```

**Parametri**

*litems* Le voci da includere nell'elenco di distribuzione.

*openOptions*
Opzioni che controllano l'apertura dell'elenco di distribuzione.

Consultare la sezione "accessDistributionList" a pagina 166 per informazioni più dettagliate sui parametri.

*alternateUserId* è impostato su "".

**begin* (solo connessione binding)**

```
public synchronized void begin()
```

Throws MQException.

Questo metodo viene supportato solo da MQSeries classi per Java in modalità binding e segnala al gestore code che sta iniziando una nuova unità di lavoro.

Non utilizzare questo metodo per applicazioni che utilizzano transazioni locali a una fase.

**isConnected**

```
public boolean isConnected()
```

Restituisce il valore della variabile isConnected.

# MQSimpleConnectionManager

```
java.lang.Object      com.ibm.mq.MQConnectionManager
   |                          |
   +- com.ibm.mq.MQSimpleConnectionManager
```

classe pubblica **MQSimpleConnectionManager**
implementa **MQConnectionManager** (Vedere pagina 173.)

Una MQSimpleConnectionManager fornisce una funzionalità di base del pooling delle connessioni. E' possibile utilizzare un MQSimpleConnectionManager come Connection Manager predefinito o come parametro di un costruttore MQQueueManager. Quando viene creato un MQQueueManager, viene utilizzata la connessione usata più di recente nel pool.

Le connessioni vengono distrutte (da un thread separato) quando non vengono utilizzate per un periodo specifico oppure quando nel pool è presente un numero superiore di connessioni inutilizzate rispetto a quello specificato. E' possibile specificare il periodo di timeout e il numero massimo di connessioni inutilizzate.

## Variabili

**MODE_AUTO**
`public static final int MODE_AUTO`. Consultare il "setActive".

**MODE_ACTIVE**
`public static final int MODE_ACTIVE`. Consultare il "setActive".

**MODE_INACTIVE**
`public static final int MODE_INACTIVE`. Consultare il "setActive".

## Costruttori

**MQSimpleConnectionManager**
`public MQSimpleConnectionManager()`

Costruisce un MQSimpleConnectionManager.

## Metodi

**setActive**
`public void setActive(int mode)`

Imposta la modalità attiva del pool di connessioni.

**Parametri**

*mode* La modalità attiva richiesta del pool di connessioni. I valori validi sono:

**MODE_AUTO**
Il pool di connessioni è attivo mentre il Connection Manager è quello predefinito e c'è almeno un token nel set di MQPoolTokens gestiti da MQEnvironment. Questa è la modalità predefinita.

**MODE_ACTIVE**
Il pool di connessioni è sempre attivo. Quando viene richiamato il metodo MQQueueManager.disconnect(), la connessione sottostante viene inserita nel pool e potenzialmente riutilizzata alla successiva costruzione di un oggetto MQQueueManager. Le connessioni saranno distrutte da un thread separato se non vengono riutilizzate per un periodo superiore a quello di Timeout oppure se la dimensione del pool supera HighThreshold.

**MODE_INACTIVE**
Il pool di connessioni è sempre inattivo. In questa modalità il pool di connessioni a MQSeries è cancellato. Quando viene richiamato il metodo MQQueueManager.disconnect(), la connessione alla base di qualsiasi oggetto MQQueueManager attivo viene distrutta.

**getActive**

```
public int getActive()
```

Imposta la modalità del pool di connessioni.

**Restituisce**

La modalità attiva corrente del pool di connessioni, con uno dei seguenti valori (vedere "setActive" a pagina 169):

MODE_AUTO

MODE_ACTIVE

MODE_INACTIVE

**setTimeout**

```
public void setTimeout(long timeout)
```

Imposta il valore Timeout in cui le connessioni che restano inutilizzate per questo lasso di tempo vengono distrutte da un thread separato.

**Parametri**

*timeout*
Il valore del timeout in millisecondi.

**getTimeout**

```
public long getTimeout()
```

Restituisce il valori Timeout.

**setHighThreshold**

```
public void setHighThreshold(int threshold)
```

Imposta il HighThreshold. Se il numero di connessioni inutilizzate nel pool supera questo valore, la connessione inutilizzata più vecchia nel pool viene distrutta.

**Parametri**

*threshold*
Il numero massimo di connessioni inutilizzate nel pool.

**getHighThreshold**

```
public int getHighThreshold ()
```

Restituisce il valore HighThreshold.

# MQC

interfaccia pubblica **MQC**
estende **Object**

L'interfaccia MQC definisce tutte le costanti utilizzate dall'interfaccia di programmazione MQ Java (ad eccezione delle costanti del codice di completamento e del codice di errore). Per fare riferimento a una di queste costanti all'interno dei programmi, far precedere il nome della costante dal prefisso "MQC.". E' possibile, ad esempio, impostare le opzioni di chiusura relative a una coda nel modo seguente:

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // delete the
                                      // queue when
                                      // it is closed
 ...
```

Una descrizione completa di queste costanti si trova in *MQSeries Application Programming Reference*.

Le costanti del codice di completamento e del codice di errore sono definite nella classe MQException. Consultare il "MQException" a pagina 103.

# MQPoolServicesEventListener

interfaccia pubblica **MQPoolServicesEventListener**
estende **Object**

**Nota:** Di norma, le applicazioni non utilizzano quest'interfaccia.

MQPoolServicesEventListener è destinata all'implementazione da parte dei provider di ConnectionManager predefiniti. Quando una MQPoolServicesEventListener viene registrata con un oggetto MQPoolServices, il listener degli eventi riceve un evento ogni volta che una MQPoolToken viene aggiunta o rimossa dalla serie di MQPoolToken gestite da MQEnvironment. Riceve inoltre un evento ogni volta che il ConnectionManager predefinito cambia.

Vedere anche "MQPoolServices" a pagina 138 e "MQPoolServicesEvent" a pagina 139.

## Metodi

**tokenAdded**

```
public void tokenAdded(MQPoolServicesEvent event)
```

Chiamato quando una MQPoolToken viene aggiunta alla serie.

**tokenRemoved**

```
public void tokenRemoved(MQPoolServicesEvent event)
```

Chiamato quando una MQPoolToken è rimossa dalla serie.

**defaultConnectionManagerChanged**

```
public void defaultConnectionManagerChanged(MQPoolServicesEvent event)
```

Chiamato quando viene impostato il ConnectionManager predefinito. La serie di MQPoolToken sarà stata cancellata.

# MQConnectionManager

Questa è un'interfaccia privata che non può essere implementata dalle applicazioni. MQSeries classi per Java fornisce un'implementazione di questa interfaccia (MQSimpleConnectionManager), che è possibile specificare sul costruttore MQQueueManager oppure tramite MQEnvironment.setDefaultConnectionManager.

Consultare il "MQSimpleConnectionManager" a pagina 169.

Le applicazioni o i middleware che desiderano fornire i propri ConnectionManager devono implementare javax.resource.spi.ConnectionManager. E' necessario che sia installato Java 2 v1.3 con JAAS 1.0.

# MQReceiveExit

interfaccia pubblica **MQReceiveExit**
estende **Object**

L'interfaccia di uscita ricezione consente di esaminare e possibilmente modificare i dati ricevuti dal gestore code da MQSeries classi per Java.

**Nota:** Questa interfaccia non è valida quando si esegue una connessione diretta a MQSeries in modalità binding.

Per fornire la propria receive exit, definire una classe che implementi questa interfaccia. Creare una nuova istanza della classe e assegnare ad essa la variabile MQEnvironment.receiveExit prima di costruire il proprio oggetto MQQueueManager. Ad esempio:

```
// in MyReceiveExit.java
class MyReceiveExit implements MQReceiveExit {
  // è necessario fornire un'implementazione
  // of the receiveExit method
  public byte[] receiveExit(
   MQChannelExit       channelExitParms,
   MQChannelDefinition channelDefinition,
   byte[]              agentBuffer)
  {
   // il codice di uscita va inserito in questo punto
  }
}
// nel programma principale
MQEnvironment.receiveExit = new MyReceiveExit();
 ...    // altra inizializzazione
MQQueueManager qMgr        = new MQQueueManager("");
```

## Metodi

**receiveExit**

```
public abstract byte[] receiveExit(MQChannelExit channelExitParms,
                                   MQChannelDefinition channelDefinition,
                                   byte agentBuffer[])
```

Il metodo di receive exit che la classe deve fornire. Questo metodo verrà richiamato ogni volta che il MQSeries classi per Java riceve alcuni dati dal gestore code.

**Parametri**

*channelExitParms*
Contiene informazioni relative al contesto in cui l'uscita viene richiamata. La variabile del membro exitResponse è un parametro di output utilizzato per indicare a MQSeries classi per Java l'azione successiva da intraprendere. Per ulteriori informazioni, consultare la sezione "MQChannelExit" a pagina 90.

*channelDefinition*
Contiene dettagli del canale tramite il quale avvengono tutte le comunicazioni con il gestore code.

*agentBuffer*
Se il channelExitParms.exitReason è MQChannelExit.MQXR_XMIT, agentBuffer contiene i dati ricevuti dal gestore code, altrimenti agentBuffer è null.

**Restituisce**

Se il codice di exit response (in channelExitParms) è impostato in modo che MQSeries classi per Java possa elaborare i dati (MQXCC_OK), il proprio metodo receive exit deve restituire i dati da elaborare. Il receive exit più semplice, quindi, è composto dalla singola riga ″return agentBuffer;″.

Vedere anche:

- “MQC” a pagina 171
- “MQChannelDefinition” a pagina 88

# MQSecurityExit

interfaccia pubblica **MQSecurityExit**
estende **Object**

L'interfaccia dell'uscita di sicurezza consente di personalizzare i flussi di sicurezza che hanno luogo quando viene effettuato un tentativo di connettersi a un gestore code.

**Nota:** Questa interfaccia non è valida quando si esegue una connessione diretta a MQSeries in modalità binding.

Per fornire la propria security exit, definire una classe che implementi questa interfaccia. Creare una nuova istanza della classe e assegnare ad essa la variabile MQEnvironment.securityExit prima di costruire il proprio oggetto MQQueueManager. Ad esempio:

```
// in MySecurityExit.java
class MySecurityExit implements MQSecurityExit {
  // è necessario fornire un'implementazione
  // del metodo securityExit
  public byte[] securityExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // il codice di uscita va inserito in questo punto
  }
}
// nel programma principale
MQEnvironment.securityExit = new MySecurityExit();
 ...    // altra inizializzazione
MQQueueManager qMgr        = new MQQueueManager("");
```

## Metodi

**securityExit**

```
public abstract byte[] securityExit(MQChannelExit channelExitParms,
                                    MQChannelDefinition channelDefinition,
                                    byte agentBuffer[])
```

Il metodo di security exit che la classe deve fornire.

**Parametri**

*channelExitParms*
Contiene informazioni relative al contesto in cui l'uscita viene richiamata. La variabile del membro exitResponse è un parametro di output utilizzato per indicare a MQSeries Client per Java l'azione successiva da intraprendere. Vedere "MQChannelExit" a pagina 90 per ulteriori dettagli.

*channelDefinition*
Contiene dettagli del canale tramite il quale avvengono tutte le comunicazioni con il gestore code.

*agentBuffer*
Se il channelExitParms.exitReason è

MQChannelExit.MQXR_SEC_MSG, agentBuffer contiene il messaggio d sicurezza ricevuto dal gestore code, altrimenti agentBuffer è null.

**Restituisce**

Se il codice exit response (in channelExitParms) è impostato in modo che un messaggio debba essere trasmesso al gestore code, il metodo security exit deve restituire i dati da trasmettere.

Vedere anche:
- "MQC" a pagina 171
- "MQChannelDefinition" a pagina 88

# MQSendExit

interfaccia pubblica **MQSendExit**
estende **Object**

L'interfaccia di uscita utente consente di esaminare e possibilmente modificare i dati inviati al gestore code da MQSeries Client per Java.

**Nota:** Questa interfaccia non è valida quando si esegue una connessione diretta a MQSeries in modalità binding.

Per fornire la propria send exit, definire una classe che implementi questa interfaccia. Creare una nuova istanza della classe e assegnare ad essa la variabile MQEnvironment.sendExit prima di costruire il proprio oggetto MQQueueManager. Ad esempio:

```
// in MySendExit.java
class MySendExit implements MQSendExit {
  // è necessario fornire un'implementazione del metodo sendExit
  public byte[] sendExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // il codice di uscita va inserito in questo punto
  }
}
// nel programma principale
MQEnvironment.sendExit = new MySendExit();
 ...    // altra inizializzazione
MQQueueManager qMgr        = new MQQueueManager("");
```

## Metodi

**sendExit**

```
public abstract byte[] sendExit(MQChannelExit channelExitParms,
                                MQChannelDefinition channelDefinition,
                                byte agentBuffer[])
```

Il metodo di send exit che la classe deve fornire. Questo metodo viene richiamato ogni volta che MQSeries classi per Java desidera trasmettere alcuni dati al gestore code.

**Parametri**

*channelExitParms*
Contiene informazioni relative al contesto in cui l'uscita viene richiamata. La variabile del membro exitResponse è un parametro di output utilizzato per indicare a MQSeries classi per Java l'azione successiva da intraprendere. Per ulteriori informazioni, consultare la sezione "MQChannelExit" a pagina 90.

*channelDefinition*
Contiene dettagli del canale tramite il quale avvengono tutte le comunicazioni con il gestore code.

*agentBuffer*
Se il channelExitParms.exitReason è MQChannelExit.MQXR_XMIT, agentBuffer contiene i dati da trasmettere al gestore code, altrimenti agentBuffer è null.

**Restituisce**

Se il codice exit response (in channelExitParms) è impostato in modo che un messaggio debba essere trasmesso al gestore code (MQXCC_OK), il metodo send exit deve restituire i dati da trasmettere. Il send exit più semplice, quindi, è composto dalla singola riga ″return agentBuffer;″.

Vedere anche:
- “MQC” a pagina 171
- “MQChannelDefinition” a pagina 88

# ManagedConnection

public interface **javax.resource.spi.ManagedConnection**

**Nota:** Di norma le applicazioni non utilizzano questa classe, che dovrebbe essere utilizzata dalle implementazioni di ConnectionManager.

MQSeries classi per Java fornisce un'implementazione di ManagedConnection restituita da ManagedConnectionFactory.createManagedConnection. Quest'oggetto rappresenta una connessione a un gestore code di MQSeries.

## Metodi

**getConnection**

```
public Object getConnection(javax.security.auth.Subject subject,
                            ConnectionRequestInfo cxRequestInfo)
```

Produce ResourceException.

Crea un nuovo handle della connessione per la connessione fisica rappresentata dall'oggetto ManagedConnection. Per MQSeries classi per Java, questo restituisce un oggetto MQQueueManager. Il ConnectionManager generalmente restituisce questo oggetto da allocateConnection.

Il parametro viene ignorato. Se il parametro cxRequestInfo non è adatto, viene prodotta una ResourceException. E' possibile utilizzare più handle della connessione contemporaneamente per ogni singola ManagedConnection.

**destroy**

```
public void destroy()
```

Produce ResourceException.

Distrugge la connessione fisica sul gestore code MQSeries e viene eseguito il commit di tutte le transazioni locali in sospeso. Per ulteriori dettagli, consultare la sezione "getLocalTransaction" a pagina 181.

**cleanup**

```
public void cleanup()
```

Produce ResourceException.

Chiude tutti gli handle selle connessioni aperte e reimposta la connessione fisica sullo stato iniziale pronto all'organizzazione in pool. Viene eseguito il rollback di tutte le transazioni locali in sospeso. Per ulteriori dettagli, consultare la sezione "getLocalTransaction" a pagina 181.

**associateConnection**

```
public void associateConnection(Object connection)
```

Produce ResourceException.

MQSeries classi per Java al momento non supporta questo metodo. Viene prodotta una javax.resource.NotSupportedException.

**addConnectionEventListener**

```
public void addConnectionEventListener(ConnectionEventListener listener)
```

Aggiunge una ConnectionEventListener all'istanza di ManagedConnection.

Il listener viene notificato se si verifica un errore grave sul ManagedConnection oppure quando il metodo MQQueueManager.disconnect() viene richiamato sull'handle della connessione associata a questa ManagedConnection. Non viene invece notificato sugli eventi della transazione locale (vedere "getLocalTransaction").

**removeConnectionEventListener**

```
public void removeConnectionEventListener(ConnectionEventListener listener)
```

Rimuove un ConnectionEventListener registrato.

**getXAResource**

```
public javax.transaction.xa.XAResource getXAResource()
```

Produce ResourceException.

MQSeries classi per Java al momento non supporta questo metodo. Viene prodotta una javax.resource.NotSupportedException.

**getLocalTransaction**

```
public LocalTransaction getLocalTransaction()
```

Produce ResourceException.

MQSeries classi per Java al momento non supporta questo metodo. Viene prodotta una javax.resource.NotSupportedException.

Al momento un ConnectionManager non è in grado di gestire la transazione locale MQSeries e i ConnectionEventListener registrati non vengono informati degli eventi relativi alla transazione locale. Quando si verifica il cleanup(), viene eseguito il rollback di tutte le unità di lavoro in corso. Quando si verifica il destroy(), viene eseguito il commit di tutte le unità di lavoro in corso.

Il comportamento dell'API esistente è che viene eseguito il commit di un'unità di lavoro in corso al MQQueueManager.disconnect(). Il comportamento esistente viene conservato solo quando un MQConnectionManager (anziché un ConnectionManager) gestisce la connessione.

**getMetaData**

```
public ManagedConnectionMetaData getMetaData()
```

Produce ResourceException.

Richiama le informazioni sui metadati per il gestore code sottostante. Consultare il "ManagedConnectionMetaData" a pagina 185.

## ManagedConnection

### setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

Produce ResourceException.

Imposta l'autore della registrazione per questa ManagedConnection. Quando viene creata una ManagedConnection, eredita l'autore della registrazione dal suo ManagedConnectionFactory.

MQSeries classi per Java al momento non utilizza l'autore della registrazione. Consultare la sezione "MQException.log" a pagina 103 per ulteriori dettagli sulla registrazione.

### getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

Produce ResourceException.

Restituisce l'autore della registrazione per questa ManagedConnection.

MQSeries classi per Java al momento non utilizza l'autore della registrazione. Consultare la sezione "MQException.log" a pagina 103 per ulteriori dettagli sulla registrazione.

# ManagedConnectionFactory

interfaccia pubblica **javax.resource.spi.ManagedConnectionFactory**

**Nota:** Di norma, le applicazioni non utilizzano questa classe.

MQSeries classi per Java fornisce un'implementazione di questa interfaccia ai ConnectionManager. Un ManagedConnectionFactory viene utilizzato per creare ManagedConnection e per selezionare ManagedConnection adatte da una serie di possibili opzioni. Per informazioni più dettagliate su questa interfaccia, vedere la specifica J2EE Connector Architecture nel sito Web della Sun all'indirizzo `http://java.sun.com`.

## Metodi

**createConnectionFactory**

```
public Object createConnectionFactory()
```

Produce ResourceException.

MQSeries classi per Java al momento non supporta i metodi createConnectionFactory. Questo metodo produce una javax.resource.NotSupportedException.

**createConnectionFactory**

```
public Object createConnectionFactory(ConnectionManager cxManager)
```

Produce ResourceException.

MQSeries classi per Java al momento non supporta i metodi createConnectionFactory. Questo metodo produce una javax.resource.NotSupportedException.

**createManagedConnection**

```
public ManagedConnection createManagedConnection
                              (javax.security.auth.Subject subject,
                               ConnectionRequestInfo cxRequestInfo)
```

Produce ResourceException.

Crea una nuova connessione fisica a un gestore code MQSeries e restituisce un oggetto ManagedConnection che rappresenta questa connessione. MQSeries ignora il parametro.

**matchManagedConnection**

```
public ManagedConnection matchManagedConnection
                              (java.util.Set connectionSet,
                               javax.security.auth.Subject subject,
                               ConnectionRequestInfo cxRequestInfo)
```

Produce ResourceException.

Ricerca all'interno della serie fornita di possibili ManagedConnection la ManagedConnection appropriata. Restituisce null o una ManagedConnection appropriata dalla serie che risponde ai criteri relativi alla connessione.

## ManagedConnectionFactory

**setLogWriter**

```
public void setLogWriter(java.io.PrintWriter out)
```

Produce ResourceException.

Imposta l'autore della registrazione per questa ManagedConnectionFactory. Quando viene creata una ManagedConnection, eredita l'autore della registrazione dal suo ManagedConnectionFactory.

MQSeries classi per Java al momento non utilizza l'autore della registrazione. Consultare la sezione "MQException.log" a pagina 103 per ulteriori dettagli sulla registrazione.

**getLogWriter**

```
public java.io.PrintWriter getLogWriter()
```

Produce ResourceException.

Restituisce l'autore della registrazione per questa ManagedConnectionFactory.

MQSeries classi per Java al momento non utilizza l'autore della registrazione. Consultare la sezione "MQException.log" a pagina 103 per ulteriori dettagli sulla registrazione.

**hashCode**

```
public int hashCode()
```

Restituisce l'hashCode per questa ManagedConnectionFactory.

**equals**

```
public boolean equals(Object other)
```

Verifica se questa ManagedConnectionFactory è pari a un'altra ManagedConnectionFactory. Restituisce true se entrambe le ManagedConnectionFactory descrivono lo stesso gestore code di destinazione.

# ManagedConnectionMetaData

interfaccia pubblica **javax.resource.spi.ManagedConnectionMetaData**

**Nota:** Di norma le applicazioni non utilizzano questa classe, che dovrebbe essere utilizzata dalle implementazioni di ConnectionManager.

Un ConnectionManager può utilizzare questa classe per richiamare i metadati relativi a una connessione fisica sottostante a un gestore code. Un'implementazione di questa classe viene restituita da ManagedConnection.getMetaData().

## Metodi

**getEISProductName**

```
public String getEISProductName()
```

Produce ResourceException.

Restituisce "IBM MQSeries".

**getProductVersion**

```
public String getProductVersion()
```

Produce ResourceException.

Restituisce una stringa che descrive il livello di comando del gestore code MQSeries a cui è connessa la ManagedConnection.

**getMaxConnections**

```
public int getMaxConnections()
```

Produce ResourceException.

Restituisce 0.

**getUserName**

```
public String getUserName()
```

Produce ResourceException.

Se la ManagedConnection rappresenta una connessione Client a un gestore code, viene restituito l'ID utente utilizzato per la connessione. Altrimenti viene restituita una stringa vuota.

## ManagedConnectionMetaData

# Parte 3. Programmazione con MQ JMS

# Capitolo 10. Scrittura di programmi MQ JMS

In questo programma verranno fornite informazioni utili per lo sviluppo di applicazioni MQ JMS. A una breve introduzione del modello JMS, seguono informazioni dettagliate sulla programmazione di alcune attività comuni che i programmi potrebbero avere la necessità di eseguire.

## Il modello JMS

JMS definisce una panoramica generica di un servizio di trasmissione messaggi. E' importante comprendere questa panoramica e come si associa al trasporto MQSeries sottostante.

Il modello JMS generico è basato sulle seguenti interfacce definite nel pacchetto `javax.jms`:

**Connection**
Fornisce l'accesso al trasporto sottostante e viene utilizzato per creare **Session**.

**Session**
Fornisce un contesto per la produzione e il consumo di messaggi, inclusi i metodi utilizzati per creare **MessageProducer** e **MessageConsumer**.

**MessageProducer**
Utilizzata per inviare messaggi.

**MessageConsumer**
Utilizzata per ricevere messaggi.

Si noti che una **Connection** è a prova di thread, a differenza di **Session**, **MessageProducer** e **MessageConsumer**. La strategia consigliata è di utilizzare una Session per thread dell'applicazione.

In termini MQSeries:

**Connection**
Fornisce un ambito per le code temporanee. Inoltre, fornisce uno spazio in cui contenere i parametri che controllano la modalità di connessione a MQSeries. Esempi di questi parametri sono il nome del gestore code ed il nome dell'host remoto se si utilizza lka connettività client MQSeries Java.

**Session**
Contiene un `HCONN` e definisce pertanto un ambito transazionale.

**MessageProducer e MessageConsumer**
Contiene un `HOBJ` che definisce una determinata coda in cui scrivere o da cui leggere.

Vengono applicate le normali regole di MQSeries:

- Una sola operazione può essere in corso per HCONN in qualsiasi momento. Pertanto le interfacce MessageProducer o MessageConsumer associate a una Session non possono essere chiamate contemporaneamente. La cosa è coerente con la restrizione JMS di un singolo thread a Session.
- Le operazioni PUT possono utilizzare le code remote, ma le GET possono essere applicate solo alle code sul gestore code locale.

Le interfacce JMS generiche sono sottoclassi in versioni più specifiche per il comportamento 'Point-to-Point' e 'Publish/Subscribe'.

Le versioni point-to-point sono:
- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

Una delle idee chiave in JMS è che è possibile, e vivamente consigliato, scrivere programmi che utilizzano solo riferimenti alle interfacce in `javax.jms`. Tutte le informazioni specifiche del fornitore vengono incapsulate nelle implementazioni di:
- QueueConnectionFactory
- TopicConnectionFactory
- Coda
- Argomento

Tali oggetti sono noti come "aggetti amministrati", vale a dire oggetti che possono essere incorporati mediante uno strumento di amministrazione del fornitore e memorizzati in uno spazio dei nomi JNDI. Un'applicazione JMS può richiamare questi oggetti dallo spazio dei nomi e utilizzarli senza che sia necessario sapere quale fornitore ha fornito l'implementazione.

## Creazione di una connessione

Le connessioni non vengono create direttamente, bensì sviluppate utilizzando un'impostazione predefinita della connessione. Gli oggetti predefiniti possono essere memorizzati in uno spazio dei nomi JNDI, isolando in questo modo l'applicazione JMS da informazioni specifiche del fornitore. Informazioni dettagliate su come creare e memorizzare oggetti predefiniti sono riportate nel "Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS" a pagina 35.

Se non è disponibile uno spazio dei nomi JNDI, consultare la sezione "Creazione di oggetti predefiniti in fase di runtime" a pagina 191.

### Richiamo dell'oggetto predefinito da JNDI

Per richiamare un oggetto da uno spazio dei nomi JNDI, è necessario impostare un contesto iniziale, come viene illustrato in questo frammento tratto dal file di esempio `IVTRun`:

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

dove:

**icf** definisce una classe predefinita per il contesto iniziale

**url** definisce un URL specifico del contesto

Per ulteriori informazioni sull'utilizzo JNDI, consultare la documentazione JNDI della Sun.

**Nota:** Alcune combinazioni dei pacchetti JNDI e dei provider di servizi LDAP possono determinare un errore LDAP 84. Per risolvere il problema, inserire la seguente riga prima della chiamata `InitialDirContext`.

```
environment.put(Context.REFERRAL, "throw");
```

Una volta ottenuto il contesto iniziale, gli oggetti vengono richiamati dallo spazio dei nomi utilizzando il metodo `lookup()`. Il codice che segue richiama un QueueConnectionFactory denominato `ivtQCF` da uno spazio dei nomi basato su LDAP:

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

## Utilizzo dell'impostazione predefinita per la creazione di una connessione

Il metodo `createQueueConnection()` sull'oggetto predefinito viene utilizzato per creare una 'Connection', come viene mostrato nel codice che segue:

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

## Creazione di oggetti predefiniti in fase di runtime

Se uno spazio dei nomi JNDI non è disponibile, è possibile creare oggetti predefiniti in fase di runtime. Tuttavia l'utilizzo di questo metodo riduce la portabilità delle applicazioni JMS in quanto richiede riferimenti a classi specifiche di MQSeries.

Il codice che segue crea una QueueConnectionFactory con tutte le impostazioni predefinite:

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

(E' possibile omettere il prefisso `com.ibm.mq.jms.` se si importa invece il pacchetto `com.ibm.mq.jms`.)

Una connessione creata dall'impostazione predefinita sopra indicata utilizza il binding Java per connettersi al gestore code predefinito sulla macchina locale. I metodi `set` illustrati nella Tabella 14 possono essere utilizzati per personalizzare l'impostazione predefinita con informazioni specifiche MQSeries.

### Avvio della connessione

La specifica JMS definisce che le connessioni dovrebbero essere create nello stato 'arrestato'. Fino all'avvio della connessione, i MessageConsumer associati alla connessione non possono ricevere alcun messaggio. Per avviare la connessione, eseguire il seguente comando:

```
connection.start();
```

*Tabella 14. Impostare i metodi su MQQueueConnectionFactory*

| Metodo | Descrizione |
|---|---|
| setCCSID(int) | Utilizzato per impostare la proprietà `MQEnvironment.CCSID` |
| setChannel(String) | Il nome del canale per una connessione client |
| setHostName(String) | Il nome dell'host per una connessione client |
| setPort(int) | La porta per una connessione client |
| setQueueManager(String) | Il nome del gestore code |

*Tabella 14. Impostare i metodi su MQQueueConnectionFactory (Continua)*

| Metodo | Descrizione |
|---|---|
| setTemporaryModel(String) | Il nome di una coda modello utilizzata per generare una destinazione temporanea come risultato di una chiamata a `QueueSession.createTemporaryQueue()`. Si consiglia di fare in modo che questo sia il nome di una coda dinamica temporanea anziché di una coda dinamica permanente. |
| setTransportType(int) | Specificare come connettersi a MQSeries. Le opzioni attualmente disponibili sono:<br>• JMSC.MQJMS_TP_BINDINGS_MQ (l'impostazione predefinita)<br>• JMSC.MQJMS_TP_CLIENT_MQ_TCPIP.<br><br>`JMSC` si trova nel pacchetto `com.ibm.mq.jms` |
| setReceiveExit(String)<br>setSecurityExit(String)<br>setSendExit(String)<br>setReceiveExitInit(String)<br>setSecutityExitInit(String)<br>setSendExitInit(String) | Lo scopo di questi metodi è di consentire l'utilizzo delle uscite di invio, ricezione e sicurezza fornite dall'applicazione MQSeries Classes per Java sottostante. I metodi `set*Exit` prendono il nome di una classe che implementa i metodi di uscita attinenti. (Consultare la documentazione del prodotto MQSeries 5.1 per maggiori dettagli.)<br><br>Inoltre la classe deve implementare un costruttore con un solo parametro `String`. Questa stringa fornisce tutti i dati di inizializzazione richiesti dall'uscita e viene impostata sul valore fornito nel metodo `set*ExitInit` corrispondente. |

## Scelta del trasporto client o binding

MQ JMS può comunicare Con MQSeries utilizza il client o i trasporti binding[2]. Se si utilizzano i binding Java, l'applicazione JMS ed il gestore code MQSeries devono trovarsi sulla stessa macchina. Se si utilizza il client, il gestore code può essere su una macchina diversa rispetto all'applicazione.

Il contenuto dell'oggetto predefinito della connessione determina il tipo di trasporto da utilizzare. Nel "Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS" a pagina 35 viene descritto come definire un oggetto predefinito per un utilizzo con il trasporto client o binding.

Il frammento di codice che segue illustra come definire il trasporto all'interno di un'applicazione:

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

2. Tuttavia, il trasporto client non è supportato sulla piattaforma z/OS & OS/390.

## Come ottenere una sessione

Una volta effettuata una connessione, utilizzare il metodo `createQueueSession` sul QueueConnection per ottenere una sessione.

Il metodo assume due parametri:

1. Un metodo booleano che determina se la sessione è 'negoziata' o 'non negoziata'.
2. Un parametro che determina la modalità di 'riconoscimento'.

### come ottenere una sessione

Il caso più semplice è quello della sessione 'non negoziata' con AUTO_ACKNOWLEDGE, come viene mostrato nel frammento di codice che segue:

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

**Nota:** Una connessione è a prova di thread, a differenza delle sessioni e degli oggetti da esse creati. La pratica raccomandata per le applicazioni è di utilizzare una sessione separata per ciascun thread.

## Invio di un messaggio

I messaggi vengono inviati tramite un MessageProducer. Per il point-to-point si tratta di un QueueSender creato mediante il metodo createSender su QueueSession. Un QueueSender viene generalmente creato per una coda specifica, in modo che tutti i messaggi inviati utilizzando quel mittente siano inviati alla stessa destinazione. La destinazione viene specificata utilizzando un oggetto Queue. Gli oggetti Queue possono essere creati in fase di runtime oppure creati e memorizzati in uno spazio dei nomi JNDI.

Gli oggetti Queue vengono richiamati da JNDI nel modo seguente:

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

MQ JMS fornisce un'implementazione dell'oggetto Queue in com.ibm.mq.jms.MQQueue. Contiene proprietà che controllano i dettagli del comportamento specifico di MQSeries, ma in molti casi è possibile utilizzare i valori predefiniti. JMS definisce un modo standard per specificare la destinazione che riduce al minimo il codice specifico di MQSeries nell'applicazione. Questo meccanismo utilizza il metodo QueueSession.createQueue che prende un parametro string che descrive la destinazione. La stringa stessa è ancora in formato specifico del fornitore ma si tratta di un approccio più flessibile rispetto al riferimento diretto alle classi del fornitore.

MQ JMS accetta due forme per il parametro string di createQueue().

- Il primo è il nome della coda MQSeries, come viene illustrato nel seguente frammento tratto dal programma IVTRun nella directory esempi:

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- La seconda e più potente forma si basa su "URI" (uniform resource identifiers). Questa forma consente di specificare code remote, vale a dire code su un gestore code diverso da quello a cui si è connessi. Consente inoltre di impostare le altre proprietà contenute in un oggetto com.ibm.mq.jms.MQQueue.

  L'URI di una coda inizia con la sequenza queue://, seguita dal nome del gestore code su cui risiede la coda. Tale nome è a sua volta seguito da un ulteriore "/", dal nome della coda e, facoltativamente, da un elenco di coppie nome-valore che impostano le proprietà Queue restanti. Ad esempio, l'URI equivalente all'esempio precedente è:

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

Si noti che il nome del gestore code è omesso e viene interpretato come il gestore code a cui il QueueConnection di appartenenza è connesso nel momento in cui è utilizzato l'oggetto Queue.

L'esempio che segue si connette alla coda Q1 sul gestore code HOST1.QM1, e determina l'invio di tutti i messaggi come non persistenti e di priorità 5:

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

Nella Tabella 15 vengono elencati i nomi che è possibile utilizzare nella parte nome-valore dell'URI. Uno svantaggio di questo formato è che non supporta i nomi simbolici per i valori. Pertanto, laddove possibile, la tabella indica anche i valori 'speciali'. Questi valori speciali possono essere soggetti a cambiamenti. (Vedere la sezione "Impostazione delle proprietà con il metodo 'set'" per un metodo alternativo di impostazione delle proprietà.)

*Tabella 15. Nomi delle proprietà per gli URI delle code*

| Proprietà | Descrizione | Valori |
|---|---|---|
| expiry | Durata del messaggio in millisecondi | 0 per illimitata, interi positivi per il timeout (ms) |
| priority | Priorità del messaggio | da 0 a 9, -1=QDEF, -2=APP |
| persistence | Se il messaggio deve essere 'fissato' su disco | 1=non persistente, 2=persistente, -1=QDEF, -2=APP |
| CCSID | Set di caratteri della destinazione | interi - valori validi elencati nella documentazione di base MQSeries |
| targetClient | Eventuale conformità a JMS dell'applicazione ricevente | 0=JMS, 1=MQ |
| encoding | Come rappresentare campi numerici | Un valore intero così come viene descritto nella documentazione di base MQSeries |
| **QDEF** - un valore speciale che indica che la proprietà deve essere determinata dalla configurazione della coda MQSeries. | | |
| **APP** - un valore speciale che indica che l'applicazione JMS può controllare questa proprietà. | | |

Una volta ottenuto l'oggetto Queue, utilizzando `createQueue` come sopra o da JNDI, deve essere passato al metodo `createSender` per creare un QueueSender:

```
QueueSender queueSender = session.createSender(ioQueue);
```

L'oggetto `queueSender` ottenuto viene utilizzato per inviare messaggi attraverso il metodo `send`:

```
queueSender.send(outMessage);
```

## Impostazione delle proprietà con il metodo 'set'

E' possibile impostare le proprietà Queue creando prima un'istanza di `com.ibm.mq.jms.MQQueue` utilizzando il costruttore predefinito. Quindi è possibile inserire i valori richiesti utilizzando metodi pubblici `set`. Questo metodo indica che è possibile utilizzare nomi simbolici per i valori delle proprietà. Tuttavia, dal momento che questi valori sono specifici del fornitore, e sono incorporati nel codice, l'applicazione diventa meno portabile.

Il frammento di codice che segue mostra l'impostazione di una proprietà della coda con un metodo `set`.

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

Nella Tabella 16 vengono riportati i valori delle proprietà simbolici forniti con MQ JMS per un utilizzo con i metodi `set`.

*Tabella 16. Valori simbolici con le proprietà della coda*

| **Proprietà** | **Parola chiave dello strumento di amministrazione** | **Valori** |
|---|---|---|
| expiry | UNLIM<br>APP | JMSC.MQJMS_EXP_UNLIMITED<br>JMSC.MQJMS_EXP_APP |
| priority | APP<br>QDEF | JMSC.MQJMS_PRI_APP<br>JMSC.MQJMS_PRI_QDEF |
| persistence | APP<br>QDEF<br>PERS<br>NON | JMSC.MQJMS_PER_APP<br>JMSC.MQJMS_PER_QDEF<br>JMSC.MQJMS_PER_PER<br>JMSC.MQJMS_PER_NON |
| targetClient | JMS<br>MQ | JMSC.MQJMS_CLIENT_JMS_COMPLIANT<br>JMSC.MQJMS_CLIENT_NONJMS_MQ |
| encoding | Integer(N)<br>Integer(R)<br>Decimal(N)<br>Decimal(R)<br>Float(N)<br>Float(R)<br>Native | JMSC.MQJMS_ENCODING_INTEGER_NORMAL<br>JMSC.MQJMS_ENCODING_INTEGER_REVERSED<br>JMSC.MQJMS_ENCODING_DECIMAL_NORMAL<br>JMSC.MQJMS_ENCODING_DECIMAL_REVERSED<br>JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL<br>JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED<br>JMSC.MQJMS_ENCODING_NATIVE |

Consultare la sezione “La proprietà ENCODING” a pagina 48 per una discussione sulla codifica.

## Tipi di messaggi

JMS fornisce diversi tipi di messaggi, ciascuno dei quali con una certa conoscenza del relativo contenuto. Per evitare il riferimento a nomi di classi specifiche del fornitore per i tipi di messaggio, vengono forniti dei metodi sull’oggetto Session per la creazione dei messaggi.

Nel programma di esempio un messaggio di testo viene creato nel seguente modo:

```
System.out.println( "Creazione di unTextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Aggiunta di testo");
outMessage.setText(outString);
```

I tipi di messaggi che è possibile utilizzare sono:
- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Informazioni dettagliate sui vari tipi sono riportate nel “Capitolo 14. Interfacce e classi JMS” a pagina 253.

## Ricezione di un messaggio

I messaggi vengono ricevuti tramite un QueueReceiver, creato da un oggetto Session attraverso il metodo `createReceiver()`. Questo metodo prende un parametro Queue che definisce da dove vengono ricevuti i messaggi. Consultare la sezione “Invio di un messaggio” a pagina 194 per informazioni dettagliate su come creare un oggetto Queue.

Il programma di esempio crea un receiver e rilegge il testo del messaggio con il seguente codice:

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

Il parametro nella chiamata di ricezione è un timeout in millisecondi. Questo parametro definisce la durata dell'attesa da parte del metodo se non è disponibile alcun messaggio immediatamente. E' possibile omettere il parametro e in tal caso la chiamata si blocca per un tempo indefinito. per evitare ritardi, utilizzare il metodo `receiveNoWait()`.

I metodi di ricezione restituiscono un messaggio del tipo appropriato. Se, ad esempio, un `TextMessage` viene inserito su una coda, quando il messaggio viene ricevuto, l'oggetto restituito è un'istanza di `TextMessage`.

Per estrarre il contenuto dal corpo del messaggio, è necessario eseguire il cast dalla classe Message generica (che rappresenta il tipo di restituzione dichiarata dei metodi di ricezione) su sottoclassi più specifiche, come `TextMessage`. Se il tipo di messaggio ricevuto non è conosciuto, è possibile utilizzare l'operatore “instanceof ” per determinare di che tipo si tratta. E' buona norma verificare sempre la classe del messaggio prima di eseguire il casting, in modo che gli errori imprevisti possano essere gestiti correttamente.

Il codice seguente illustra l'utilizzo di “instanceof” e l'estrazione del contenuto da un TextMessage:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Stampare messaggio di errore se il messaggio non era un messaggio di testo.
  System.out.println("Il messaggio di risposta non era un messaggio di testo");
}
```

## Selettori dei messaggi

JMS fornisce un meccanismo per la selezione di un sottoinsieme di messaggi su una coda in modo che possa essere restituito da una chiamata di ricezione. Durante la creazione di un QueueReceiver, può essere fornita una stringa contenente un'espressione SQL (Structured Query Language) per determinare i messaggi da richiamare. Il selettore può fare riferimento ai campi nell'intestazione dei messaggi JMS nonché ai campi nelle proprietà del messaggio. Si tratta di campi dell'intestazione definiti dall'applicazione in modo efficace. I dettagli dei nomi dei campi dell'intestazione, nonché la sintassi del selettore SQL sono riportati nel “Capitolo 12. Messaggi JMS” a pagina 213.

### ricezione di un messaggio

Nell'esempio che segue viene illustrato come effettuare una selezione per una proprietà definita dall'utente denominata myProp:

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

**Nota:** La specifica JMS non consente la modifica del selettore associato a un receiver. Una volta creato un receiver, il selettore viene fissato per la durata di quel receiver. Ci significa che se sono richiesti selettori differenti, è necessario creare nuovi receiver.

## Consegna asincrona

Un'alternativa all'esecuzione di chiamate a `QueueReceiver.receive()` consiste nel registrare un metodo richiamato automaticamente quando è disponibile un messaggio adatto. Il frammento che segue illustra il meccanismo:

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // Il metodo che verrà richiamato da JMS quando un messaggio
  // è disponibile.
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // elaborazione specifica dell'applicazione in questa posizione
    .
    .
    .
  }
}

 .
 .
 .
  // Nel programma principale (possibilmente di qualche altra classe)
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // il programma principale può ora continuare con il comportamento specifico
  // di qualche altra applicazione.
```

**Nota:** L'utilizzo della consegna asincrona con un QueueReceiver caratterizza l'intera sessione come asincrona. E' sbagliato effettuare una chiamata esplicita ai metodi `receive` di un QueueReceiver associato a un Session che utilizza la consegna asincrona.

# Chiusura

La sola raccolta di dati inutili (Garbage collection) non è in grado di rilasciare tempestivamente tutte le risorse MQSeries. Questo è valido soprattutto se l'applicazione deve creare molti oggetti JMS di breve durata a livello di Session o inferiore. Pertanto è importante richiamare i metodi `close()` delle varie classi (QueueConnection, QueueSession, QueueSender e QueueReceiver) quando le risorse non sono più necessarie.

## Blocco di Java Virtual Machine nella chiusura

Se un'applicazione MQ JMS termina senza chiamare `Connection.close()`, alcune JVM sembreranno bloccarsi. Se si verifica questo problema, modificare l'applicazione per includere una chiamata a `Connection.close()` oppure terminare la JVM utilizzando i tasti `Ctrl-C`.

# Gestione degli errori

Tutti gli errori che si verificano in fase di runtime in un'applicazione JMS vengono riportati dalle eccezioni. La maggior parte dei metodi in JMS produce JMSExceptions per indicare gli errori. Una buona regola della programmazione consiste nel raccogliere queste eccezioni e visualizzarle in un output adatto.

A differenza delle normali Exception Java, una JMSException può contenere al suo interno una ulteriore eccezione. Per JMS questo può essere un modo prezioso per trasmettere dettagli importanti dal trasporto sottostante. Nel caso di MQ JMS, quando MQSeries produce una MQException, tale eccezione viene generalmente inclusa come eccezione incorporata in una JMSException.

L'implementazione di JMSException non include l'eccezione incorporata nell'output del relativo metodo `toString()`. Pertanto è necessario verificare in modo esplicito un'eccezione incorporata e stamparla, come viene illustrato nel frammento che segue:

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

## Listener delle eccezioni

Per la consegna asincrona dei messaggi, il codice dell'applicazione non è in grado di raccogliere eccezioni determinate da errori di ricezione dei messaggi in quanto il codice dell'applicazione non esegue chiamate esplicite ai metodi `receive()`. Per gestire questa situazione, è possibile registrare un `ExceptionListener`, che rappresenta un'istanza di una classe che implementa il metodo `onException()`. Quando si verifica un errore grave, questo metodo viene richiamato con la JMSException trasmessa come unico parametro. Ulteriori dettagli sono disponibili nella documentazione JMS della Sun.

# Capitolo 11. Programmazione di applicazioni Publish/Subscribe

In questa sezione verrà introdotto il modello di programmazione utilizzato per la creazione di applicazioni di tipo Publish/Subscribe che utilizzano MQSeries Classes for Java Message Service.

## Scrittura di una semplice applicazione di tipo Publish/Subscribe

In questa sezione verrà fornito un esempio pratico di una semplice applicazione MQ JMS.

### Importazione dei pacchetti richiesti

Un'applicazione MQSeries classi per Java Message Service si avvia con una serie di istruzioni di importazione che dovrebbero includere almeno quanto segue:

```
import javax.jms.*;                // Interfacce JMS
import javax.naming.*;             // Utilizzate per la ricerca JNDI di
import javax.naming.directory.*;   // oggetti amministrati
```

### Ottenere o creare oggetti JMS

Il passaggio successivo consiste nell'ottenere o creare un certo numero di oggetti JMS:

1. Ottenere un TopicConnectionFactory
2. Creare un TopicConnection
3. Creare un TopicSession
4. Ottenere un Topic da JNDI
5. Creare TopicPublisher e TopicSubscriber

Molti dei processi appena indicati sono simili a quelli che vengono utilizzati per il point-to-point, come viene illustrato di seguito:

**Ottenere un TopicConnectionFactory**

Il modo preferito per eseguire questa operazione consiste nell'utilizzare la ricerca JNDI, in affinché la portabilità del codice dell'applicazione possa essere mantenuta. Il codice che segue inizializza un contesto JNDI:

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://server.company.com/o=company_us,c=us";

Java.util.Hashtable env = new java.util.Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, CTX_FACTORY );
env.put( Context.PROVIDER_URL,            INIT_URL );
env.put( Context.REFERRAL,                "throw" );

Context ctx = null;
try {
   ctx = new InitialDirContext( env );
} catch( NamingException nx ) {
   // Aggiungere del codice per gestire l'impossibilità di connettersi a un contesto JNDI
}
```

**Nota:** Le variabili `CTX_FACTORY` e `INIT_URL` richiedono una personalizzazione per rispondere alle esigenze specifiche dell'installazione e del provider di servizi JNDI.

Le proprietà richieste dall'inizializzazione JNDI sono in una tabella hash, trasmessa al costruttore InitialDirContext. Se questa connessione non riesce, viene prodotto un errore che indica che gli oggetti amministrati richiesti successivamente nell'applicazione non sono disponibili.

Ottenere ora un TopicConnectionFactory utilizzando una chiave di ricerca definita dall'amministratore:

```
TopicConnectionFactory factory;
factory = (TopicConnectionFactory)lookup("cn=sample.tcf");
```

Se uno spazio dei nomi JNDI non è disponibile, è possibile creare un TopicConnectionFactory in fase di runtime. E' possibile creare un nuovo com.ibm.mq.jms.MQTopicConnectionFactory in modo simile al metodo descritto per un QueueConnectionFactory nella sezione "Creazione di oggetti predefiniti in fase di runtime" a pagina 191.

**Creare un TopicConnection**

Questo oggetto viene creato dall'oggetto TopicConnectionFactory. Le connessioni vengono sempre inizializzate in uno stato `stop` e devono essere avviate con il seguente codice:

```
TopicConnection conn;
conn = factory.createTopicConnection();
conn.start();
```

**Creare un TopicSession**

Questo oggetto viene creato utilizzando l'oggetto TopicConnection. Questo metodo utilizza due parametri, uno per indicare che la sessione è negoziata e uno per specificare la modalità di riconoscimento:

```
TopicSession session = conn.createTopicSession( false,
                                                Session.AUTO_ACKNOWLEDGE );
```

**Ottenere un Topic**

Questo oggetto può essere ottenuto da JNDI, per l'utilizzo con TopicPublishers e TopicSubscribers creati successivamente. Il codice che segue richiama un Topic:

```
Topic topic = null;
 try {
    topic = (Topic)ctx.lookup( "cn=sample.topic" );
 } catch( NamingException nx ) {
    // Aggiungere del codice per gestire l'impossibilità di richiamare Topic da JNDI
 }
```

Se uno spazio dei nomi JNDI non è disponibile, è possibile creare un Topic in fase di runtime, come viene descritto nella sessione "Creazione di argomenti in fase di runtime" a pagina 204.

**Creare consumer e produttori di pubblicazioni**

A seconda del tipo di applicazione client JMS sviluppata, è necessario creare un sottoscrittore, un editore o entrambi. Utilizzare i metodi `createPublisher` e `createSubscriber` nel modo seguente:

```
// Creare un editore, che pubblica su quel
determinato argomento
 TopicPublisher pub = session.createPublisher( topic );
// Creare un sottoscrittore, che effettua la sottoscrizione su quel determinato argomento
 TopicSubscriber sub = session.createSubscriber( topic );
```

## Pubblicare messaggi

L'oggetto TopicPublisher, pub, viene utilizzato per pubblicare messaggi, così come un QueueSender viene utilizzato nel dominio point-to-point. Il frammento che segue crea un TextMessage utilizzando la sessione, quindi pubblica il messaggio:

```
// Creare il TextMessage e inserirvi alcuni dati
TextMessage outMsg = session.createTextMessage();
outMsg.setText( "Questa è una stringa di testo breve!" );

// Utilizzare l'editore per pubblicare il messaggio
pub.publish( outMsg );
```

## Ricevere sottoscrizioni

I sottoscrittori devono essere in grado di leggere le sottoscrizioni a loro consegnate, come nel codice che segue:

```
// Richiamare la sottoscrizione successiva in attesa
TextMessage inMsg = (TextMessage)sub.receive();

// Ottenere il contenuto del messaggio
String payload = inMsg.getText();
```

Questo frammento di codice esegue un'operazione di tipo 'get-with-wait', che indica che la chiamata di ricezione si bloccherà fino a quando è disponibile un messaggio. Sono disponibili versioni alternative della chiamata di ricezione, quali ad esempio 'receiveNoWait'. Per ulteriori dettagli, consultare la sezione "TopicSubscriber" a pagina 373.

## Liberare le risorse non desiderate

E' importante liberare tutte le risorse utilizzate dall'applicazione di tipo Publish/Subscribe quando essa termina. Utilizzare il metodo close() sugli oggetti che è possibile chiudere (editori, sottoscrittori, sessioni e connessioni):

```
// Chiudere editore e sottocrittore
pub.close();
sub.close();

// Chiudere sessioni e connessioni
session.close();
conn.close();
```

# Utilizzo degli argomenti

In questa sezione verrà illustrato l'utilizzo di oggetti JMS Topic nelle applicazioni MQSeries classi per Java Message Service.

## Nomi di argomenti

In questa sezione verrà descritto l'uso dei nomi di argomenti all'interno di MQSeries classi per Java Message Service.

**Nota:** La specifica JMS non specifica dettagli esatti sull'uso e la gestione delle gerarchie degli argomenti. Pertanto quest'area può variare ampiamente da un provider all'altro.

I nomi degli argomenti in MQ JMS sono organizzati in una struttura gerarchica ad albero, di cui viene illustrato un esempio nella Figura 3 a pagina 204.

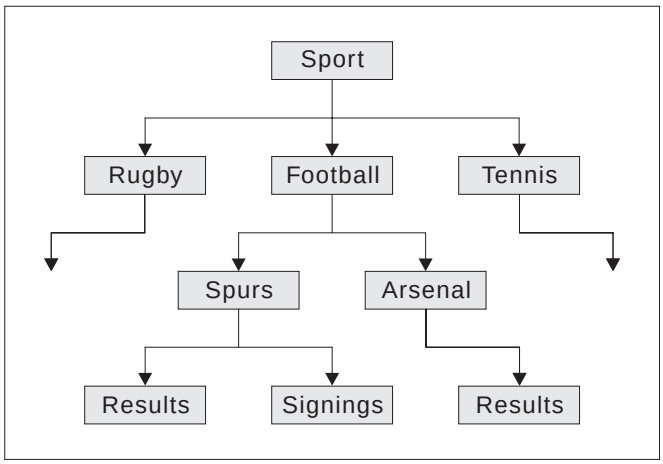


*Figura 3. gerarchia del nome argomento MQSeries classi per Java Message Service*

In un nome argomento, i livelli nella struttura ad albero sono separati dal carattere "/". Ciò significa che il nodo "Signings" nella Figura 3 viene identificato dal nome argomento:

```
Sport/Football/Spurs/Signings
```

Una potente funzione del sistema di argomenti in MQSeries classi per Java Message Service è rappresentata dall'utilizzo dei caratteri jolly che consentono ai sottoscrittori di eseguire la sottoscrizione a più di un argomento alla volta. Il carattere jolly "*" corrisponde a zero o a più caratteri, mentre "?" corrisponde ad un solo carattere.

Se un sottoscrittore effettua la sottoscrizione a un argomento rappresentato dal seguente nome di argomento:

```
Sport/Football/*/Risultati
```

riceverà le pubblicazioni relative agli argomenti che includono:
- Sport/Football/Campionato/Risultati
- Sport/Football/Juventus/Risultati

Se l'argomento della sottoscrizione è:

```
Sport/Football/Campionato/*
```

riceverà le pubblicazioni relative ad argomenti che includono:
- Sport/Football/Campionato/Risultati
- Sport/Football/Campionato/Marcatori

Non è necessario gestire esplicitamente le gerarchie di argomenti utilizzati sul lato broker. Quando il primo editore o sottoscrittore su un determinato argomento comincia a esistere, il broker crea automaticamente lo stato degli argomenti su cui è attualmente in corso la pubblicazione e la sottoscrizione.

**Nota:** Un editore non può pubblicare su un argomento il cui nome contiene caratteri jolly.

## Creazione di argomenti in fase di runtime

Esistono quattro modi per creare oggetti Topic in fase di runtime:

1. Costruire un Topic utilizzando il costruttore `MQTopic` a un argomento

2. Costruire un argomento utilizzando il costruttore predefinito MQTopic, quindi chiamare il metodo setBaseTopicName(..)
3. Utilizzare il metodo createTopic(..) della sessione
4. Utilizzare il metodo createTemporaryTopic(..) della sessione

**Metodo 1: Utilizzo di MQTopic(..)**

Questo metodo richiede un riferimento all'implementazione MQSeries dell'interfaccia JMS Topic, pertanto rende il codice non portabile.

Il costruttore prende un argomento, che dovrebbe essere un URI (uniform resource identifier). Per i Topic MQSeries classi per Java Message Service, dovrebbe avere la forma:

```
topic://TopicName[?property=value[&property=value]*]
```

Per ulteriori informazioni sugli URI e sulle coppie nome-valore consentite, consultare la sezione "Invio di un messaggio" a pagina 194.

Il codice che segue crea un argomento per i messaggi non persistenti a priorità 5:

```
// Creare un Topic utilizzando il costruttore MQTopic a un argomento
String tSpec = "Sport/Football/Campionato/Risultati?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

**Metodo 2: Utilizzo di MQTopic(), quindi setBaseTopicName(..)**

Questo metodo utilizza il costruttore predefinito MQTopic, pertanto rende il codice non portabile.

Dopo la creazione dell'oggetto, impostare la proprietà baseTopicName utilizzando il metodo setBaseTopicName, passando il nome di argomento richiesto.

**Nota:** Il nome di argomento utilizzato in questo esempio è nel formato non URI e non può includere coppie nome-valore. Impostarle utilizzando i metodi 'set', come viene descritto nella sezione "Impostazione delle proprietà con il metodo 'set'" a pagina 195. Il codice che segue utilizza questo metodo per creare un argomento:

```
// Creare un Topic utilizzando il costruttore predefinito MQTopic
Topic rtTopic = new MQTopic();

// Impostare le proprietà dell'oggetto utilizzando i metodi setter
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Campionato/Risultati" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

**Metodo 3: Utilizzo di session.createTopic(..)**

Un oggetto Topic può essere creato anche utilizzando il metodo createTopic di TopicSession, che utilizza un URI dell'argomento nel modo seguente:

```
// Creare un Topic
utilizzando il metodo predefinito della sessione
Topic rtTopic = session.createTopic( "topic://Sport/Football/Campionato/Risultati" );
```

**Metodo 4: Utilizzo di session.createTemporaryTopic()**

Un TemporaryTopic è un Topic che può essere utilizzato solo dai sottoscrittori creati dallo stesso TopicConnection. Un TemporaryTopic viene creato nel modo seguente:

```
// Creare un TemporaryTopic utilizzando il metodo predefinito della sessione
Topic rtTopic = session.createTemporaryTopic();
```

# Opzioni del sottoscrittore

Sono disponibili diversi metodi per l'utilizzo dei sottoscrittori JMS. In questa sezione verranno riportati degli esempi che ne illustrano l'utilizzo.

JMS fornisce due tipi di sottoscrittori:

**Sottoscrittori non durevoli**
Questi sottoscrittori ricevono i messaggi sull'argomento che hanno scelto, ma solo se i messaggi vengono pubblicati mentre il sottoscrittore è attivo.

**Sottoscrittori durevoli**
Tali sottoscrittori ricevono tutti i messaggi pubblicati su un argomento, inclusi quelli che vengono pubblicati mentre il sottoscrittore è inattivo.

## Creazione di sottoscrittori non durevoli

Il sottoscrittore creato nella sezione "Creare consumer e produttori di pubblicazioni" a pagina 202 non è durevole e viene creato con il seguente codice:

```
// Creare un sottoscrittore, che effettua la sottoscrizione su quel determinato argomento
 TopicSubscriber sub = session.createSubscriber( topic );
```

## Creazione di sottoscrittori durevoli

La creazione di un sottoscrittore durevole è molto simile alla creazione di un sottoscrittore non durevole, ma in aggiunta è necessario fornire anche un nome che identifichi in modo univoco il sottoscrittore:

```
// Creare un sottoscrittore durevole, fornendo un nome che identifichi in modo univoco
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

I sottoscrittori non durevoli annullano automaticamente la registrazione quando il metodo `close()` viene richiamato oppure quando non sono più compresi nell'ambito. Tuttavia, se si desidera terminare una sottoscrizione durevole, è necessario segnalarlo esplicitamente al sistema. A tale scopo, utilizzare il metodo `unsubscribe()` della sessione e passare il nome univoco che ha creato il sottoscrittore:

```
// Annullare la sottoscrizione del sottoscrittore durevole creata in precedenza
session.unsubscribe( "D_SUB_000001" );
```

Un sottoscrittore durevole viene creato nel gestore code specificato nei parametri del gestore code MQTopicConnectionFactory. Se si verifica un tentativo successivo di creare un sottoscrittore durevole con lo stesso nome in un gestore code differente, verrà restituito un sottoscrittore durevole nuovo e completamente indipendente.

## Utilizzo dei selettori dei messaggi

E' possibile utilizzare i selettori dei messaggi per filtrare i messaggi che non soddisfano determinati criteri. Per informazioni dettagliate sui selettori dei messaggi, consultare la sezione "Selettori dei messaggi" a pagina 197. I selettori dei messaggi sono associati a un sottoscrittore nel modo seguente:

```
// Associare un selettore dei messaggi a un sottoscrittore non durevole
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

## Eliminazione delle pubblicazioni locali

E' possibile creare un sottoscrittore che ignora le pubblicazioni pubblicate sulla connessione del sottoscrittore. Impostare il terzo parametro della chiamata `createSubscriber` su true, nel modo seguente:

```
// Creare un sottoscrittore non durevole con l'opzione noLocal impostata
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

## Combinazione delle opzioni del sottoscrittore

E' possibile combinare le variazioni di sottoscrittore in modo da poter creare un sottoscrittore durevole che si applica a un selettore e che ignora le pubblicazioni locali, qualora lo si desideri. Il frammento di codice che segue mostra l'utilizzo delle opzioni combinate:

```
// Creare un sottoscrittore durevole, noLocal con un selettore applicato
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                        selector, true );
```

## Configurazione della coda del sottoscrittore di base

Con MQ JMS V5.2 è possibile configurare i sottoscrittori in due modi:

- Approccio a più code

  Ogni sottoscrittore dispone di una coda esclusiva assegnata da cui richiama tutti i messaggi. JMS crea una nuova coda per ciascun sottoscrittore. Questo è l'unico approccio disponibile con MQ JMS V1.1.
- Approccio di coda condivisa

  Un sottoscrittore utilizza una coda condivisa da cui richiama, così come altri sottoscrittori, i propri messaggi. Questo approccio richiede l'utilizzo di una sola coda che possa servire più sottoscrittori. Si tratta dell'approccio predefinito utilizzato con MQ JMS V5.2.

In MQ JMS V5.2 è possibile scegliere l'approccio da utilizzare e configurare le code da utilizzare.

In generale l'approccio della coda condivisa non offre grandissimi vantaggi in termini di miglioramento delle prestazioni. Per i sistemi caratterizzati da velocità elevate si riscontrano anche grandi vantaggi in termini di architettura e gestione grazie alla significativa riduzione del numero di code richieste.

In alcune situazioni l'approccio a più code resta comunque quello da utilizzare:

- La capacità fisica teorica per la memorizzazione dei messaggi è superiore.

  Una coda MQSeries non può contenere più di 640000 messaggi e nell'approccio della coda condivisa, deve essere divisa tra tutti i sottoscrittori che condividono la coda. La questione è più significativa per i sottoscrittori durevoli, poiché la durata di un sottoscrittore durevole è generalmente superiore a quella di un sottoscrittore non durevole. Pertanto, è possibile che più messaggi si accumulino per un sottoscrittore durevole.
- La gestione esterna delle code di sottoscrizione è più semplice.

  Per determinati tipi di applicazione gli amministratori possono controllare lo stato e la profondità di determinate code di sottoscrittori. Questa attività è molto più semplice quando c'è una corrispondenza uno a uno tra un sottoscrittore e una coda.

### Configurazione predefinita

La configurazione predefinita utilizza le seguenti code di sottoscrizione condivise:

- SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE per le sottoscrizioni non durevoli
- SYSTEM.JMS.D.SUBSCRIPTION.QUEUE per le sottoscrizioni durevoli

Esse vengono create quando si esegue lo script MQJMS_PSQ.MQSC.

Se viene richiesto, è possibile specificare code fisiche alternative. E' possibile anche cambiare la configurazione per utilizzare l'approccio a più code.

## Configurazione di sottoscrittori non durevoli

E' possibile impostare la proprietà del nome della coda del sottoscrittore non durevole in uno dei seguenti modi:

- Utilizzare lo strumento di amministrazione MQ JMS (per gli oggetti richiamati JNDI) per impostare la proprietà BROKERSUBQ
- Utilizzare il metodo setBrokerSubQueue() nel proprio programma

Per le sottoscrizioni non durevoli il nome della coda fornito dovrebbe iniziare con i seguenti caratteri:

SYSTEM.JMS.ND.

Per selezionare un approccio della coda condivisa, specificare un nome di coda esplicito in cui la coda in questione sia quella da utilizzare per la coda condivisa. La coda specificata deve già esistere fisicamente prima di creare la sottoscrizione.

Per selezionare l'approccio a più code, specificare un nome di coda che termini con il carattere *. Di conseguenza, ciascun sottoscrittore creato con questo nome di coda crea una coda dinamica appropriata, destinata a un utilizzo esclusivo da parte di quel determinato sottoscrittore. MQ JMS utilizza la propria coda modello interna per creare questo tipo di code. Pertanto, con l'approccio a più code, tutte le code richieste vengono create dinamicamente.

Quando si utilizza l'approccio a più code, non è possibile specificare un nome di coda esplicito, bensì il solo prefisso. In questo modo si possono creare diversi domini delle code del sottoscrittore. E' possibile ad esempio utilizzare:

SYSTEM.JMS.ND.MYDOMAIN.*

I caratteri che precedono * vengono utilizzati come prefisso in modo che tutte le code dinamiche associate a questa sottoscrizione abbiano nomi di coda che iniziano con SYSTEM.JMS.ND.MYDOMAIN.

## Configurazione di sottoscrittori durevoli

Come è già stato illustrato in precedenza, l'utilizzo dell'approccio a più code per le sottoscrizioni durevoli ha comunque la sua validità. E' molto probabile infatti che le sottoscrizioni durevoli abbiano una durata maggiore e che sulla coda si accumulino quindi parecchi messaggi non recuperati.

Pertanto la proprietà del nome della coda del sottoscrittore durevole viene impostata nell'oggetto Topic, vale a dire a un livello più gestibile rispetto a TopicConnectionFactory. Ciò consente di specificare un certo numero di nomi di coda di sottoscrittori diversi, senza la necessità di ricreare più oggetti a partire da TopicConnectionFactory.

E' possibile impostare la proprietà del nome della coda del sottoscrittore durevole in uno dei seguenti modi:

- Utilizzare lo strumento di amministrazione MQ JMS (per gli oggetti richiamati JNDI) per impostare la proprietà BROKERDURSUBQ
- Utilizzare il metodo setBrokerDurSubQueue() nel proprio programma:

```
// Impostare il nome della coda del sottoscrittore durevole MQTopic utilizzando
// l'approccio a più code
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

Una volta inizializzato, l'oggetto Topic viene trasmesso al metodo TopicSession createDurableSubscriber() per creare la sottoscrizione specificata:

```
// Creare un sottoscrittore durevole utilizzando il Topic presentato in precedenza
TopicSubscriber sub = new session.createDurableSubscriber
                                          (sportsTopic, "D_SUB_SPORT_001");
```

Per le sottoscrizioni durevoli il nome della coda fornito dovrebbe iniziare con i seguenti caratteri:

SYSTEM.JMS.D.

Per selezionare un approccio della coda condivisa, specificare un nome di coda esplicito in cui la coda in questione sia quella da utilizzare per la coda condivisa. La coda specificata deve già esistere fisicamente prima di creare la sottoscrizione.

Per selezionare l'approccio a più code, specificare un nome di coda che termini con il carattere *. Di conseguenza, ciascun sottoscrittore creato con questo nome di coda crea una coda dinamica appropriata, destinata a un utilizzo esclusivo da parte di quel determinato sottoscrittore. MQ JMS utilizza la propria coda modello interna per creare questo tipo di code. Pertanto, con l'approccio a più code, tutte le code richieste vengono create dinamicamente.

Quando si utilizza l'approccio a più code, non è possibile specificare un nome di coda esplicito, bensì il solo prefisso. In questo modo si possono creare diversi domini delle code del sottoscrittore. E' possibile ad esempio utilizzare:

SYSTEM.JMS.D.MYDOMAIN.*

I caratteri che precedono * vengono utilizzati come prefisso in modo che tutte le code dinamiche associate a questa sottoscrizione abbiano nomi di coda che iniziano con SYSTEM.JMS.D.MYDOMAIN.

### Problemi di ricreazione e migrazione per i sottoscrittori durevoli

Per un sottoscrittore durevole, non tentare di riconfigurare il nome della coda del sottoscrittore fino all'eliminazione di quest'ultimo. In altre parole, eseguire un'operazione di unsubscribe(), quindi ricreare la coda, tenendo presente che i messaggi del sottoscrittore precedente sono stati eliminati.

Tuttavia, il sottoscrittore creato mediante MQ JMS V1.1 verrà riconosciuto nella migrazione al livello corrente. Non è necessario eliminare la sottoscrizione che continuerà a funzionare utilizzando un approccio a più code.

## Risoluzione dei problemi relativi alle operazioni di Publish/Subscribe

In questa sezione verranno descritti alcuni problemi che possono verificarsi quando si sviluppano applicazioni client JMS che utilizzano il dominio publish/subscribe. I problemi illustrati sono specifici del dominio publish/subscribe. Fare riferimento alle sezioni "Gestione degli errori" a pagina 199 e "Risoluzione dei problemi" a pagina 33 per ulteriori informazioni più generiche sulla risoluzione dei problemi.

### Chiusura incompleta Publish/Subscribe

E' importante che le applicazioni client JMS chiudano tutte le risorse esterne quando vengono terminate. Per eseguire questa operazioni, chiamare il metodo `close()` su tutti gli oggetti che possono essere chiusi quando non sono più richiesti. Per il dominio publish/subscribe, gli oggetti sono:

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

L'implementazione MQSeries classi per Java Message Service facilita l'esecuzione di questa attività attraverso l'utilizzo ad una "chiusura a cascata". Con questo processo, una chiamata "close" su un TopicConnection determina chiamate a "close" su ciascuno dei TopicSessions creati. A sua volta ciò determina una serie di chiamate a "close" su tutti gli oggetti TopicSubscribers e TopicPublishers creati dalla sessione.

Pertanto, per assicurare il rilascio appropriato delle risorse esterne, è importante richiamare `connection.close()` per ciascuna delle connessioni create da un'applicazione.

In alcune circostanze la procedura "close" non è completa. Tali circostanze includono:
- Perdita di una connessione client MQSeries
- Chiusura inaspettata dell'applicazione

In queste circostanze, il metodo `close()` non viene richiamato e le risorse esterne restano aperte da parte dell'applicazione terminata. Le principali conseguenze sono:

**Inconsistenza dello stato del broker**
Il broker messaggi MQSeries può contenere informazioni sulla registrazione relative a sottoscrittori ed editori che non esistono più. Ciò significa che il broker può continuare a inoltrare i messaggi a sottoscrittori che non li riceveranno mai.

**I messaggi e le code dei sottoscrittori non vengono eliminati**
Parte della procedura di annullamento della registrazione consiste nell'eliminazione dei messaggi del sottoscrittore. Qualora sia necessario, viene eliminata anche la coda MQSeries sottostante utilizzata per ricevere le sottoscrizioni. Se non è stata eseguita una chiusura normale, questi messaggi e queste code restano. In presenza di uno stato di incoerenza dello stato del broker, le code continuano a riempirsi di messaggi che non verranno mai letti.

## Programma di utilità per la pulizia dei sottoscrittori

Per evitare i problemi associati a una chiusura non corretta degli oggetti del sottoscrittore, MQ JMS un programma di utilità per pulizia dei sottoscrittori. Tale utilità si esegue su un gestore code al momento dell'inizializzazione del primo TopicConnection che deve utilizzare quel gestore code fisco. Se tutti i TopicConnections su un determinato gestore code vengono chiusi, quando il successivo TopicConnection viene inizializzato per quel gestore code, il programma di utilità viene eseguito nuovamente.

Tenta di rilevare gli eventuali problemi precedenti di publish/subscribe di MQ JMS che possono essersi verificati in altre applicazioni. Se i problemi vengono rilevati, il programma di utilità esegue una pulizia delle risorse associate attraverso le seguenti operazioni:
- annullamento della registrazione al broker dei messaggi MQSeries
- pulizia degli eventuali messaggi non recuperati e delle code associate alla sottoscrizione

Il programma di utilità viene eseguito in modo trasparente in background e persiste solo per un breve periodo. Non dovrebbe interferire con le altre operazioni di MQ JMS. Se viene rilevato un gran numero di problemi su un determinato gestore code, è possibile che si verifichi un breve ritardo in fase di inizializzazione durante la pulizia delle risorse.

**Nota:** Si consiglia di chiudere in modo corretto tutti gli oggetti del sottoscrittore quando è possibile per evitare che i problemi si accumulino.

## Gestione dei prospetti del broker

L'implementazione di MQ JMS utilizza dei messaggi di prospetto dal broker per confermare i comandi di registrazione e annullamento della registrazione. Questi prospetti vengono generalmente consumati dall'implementazione di MQSeries classi per Java Message Service, ma in presenza di alcune condizioni di errore potrebbero rimanere sulla coda. Tali messaggi vengono inviati alla coda `SYSTEM.JMS.REPORT.QUEUE` sul gestore code locale.

Un'applicazione Java, `PSReportDump`, viene fornita con MQSeries classi per Java Message Service, che scarica il contenuto di questa coda in formato di solo testo. Le informazioni possono essere quindi analizzate, dall'utente stesso o dal personale tecnico IBM. E' possibile anche utilizzare l'applicazione per cancellare il contenuto della coda di messaggi dopo aver diagnosticato e risolto un problema.

La forma compilata dello strumento viene installata nella directory `<MQ_JAVA_INSTALL_PATH>/bin`. Per richiamare lo strumento, passare a questa directory, quindi utilizzare il seguente comando:

```
java PSReportDump [-m queueManager] [-clear]
```

dove:

**-m queueManager**
= specifica il nome del gestore code da utilizzare

**-clear** cancella la coda dei messaggi dopo averne scaricato il contenuto

L'output viene visualizzato sullo schermo o può essere reindirizzato in un file.

# Capitolo 12. Messaggi JMS

I messaggi JMS sono composti dalle seguenti parti:

**Intestazione** Tutti i messaggi supportano lo stesso gruppo di campi di intestazione. I Campi di intestazione contengono valori utilizzati sia dai client che dai provider per identificare ed instradare i messaggi.

**Proprietà** Ciascun messaggio contiene una funzione integrata per supportare i valori di proprietà definiti dalle applicazioni. Le proprietà forniscono un efficace meccanismo per filtrare i messaggi definiti dalle applicazioni.

**Corpo** JMS definisce vari tipi di corpo di messaggio che abbracciano la maggior parte degli stili di messaggistica attualmente utilizzati.

JMS definisce cinque tipi di corpo di messaggio:

**Flusso** un flusso di valori primitiviJava. Esso viene compilato e letto in modo sequenziale.

**Associazione** Un gruppo di coppie nome-valore, dove i nomi sono stringhe ed i valori sono tipi primitivi Java E' possibile accedere a questi elementi in modo sequenziale oppure in modo casuale per nome. L'ordine degli elementi non è definito.

**Testo** Un messaggio che contiene una java.util.String.

**Oggetto** un messaggio che contiene un oggetto Java serializzabile

**Byte** un flusso di messaggi non convertito. Questo tipo di messaggio serve a codificare letteralmente un corpo di messaggio in modo che corrisponda ad un formato di messaggio esistente.

Il campo di intestazione JMSCorrelationID viene utilizzato per collegare un messaggio a un altro messaggio. Esso di norma collega un messaggio di risposta al relativo messaggio richiedente. JMSCorrelationID può contenere un ID di messaggio specifico per il provider, una stringa specifica per l'applicazione oppure un valore byte [] nativo del provider.

## Selettori di messaggi

Un messaggio contiene una funzione integrata per supportare i valori di proprietà definiti dalle applicazioni. In effetti, questo fornisce un meccanismo per aggiungere campi di intestazione specifici per le applicazioni ad un messaggio. Tramite i selettori dei messaggi, un'applicazione può utilizzare un provider JMS per selezionare o filtrare messaggi utilizzando criteri specifici per le applicazioni. Le proprietà definite dalle applicazioni devono rispettare le seguenti regole:

- I nomi di proprietà devono rispettare le regole esistenti per un ID di selettore di messaggi.
- I valori delle proprietà possono essere di tipo boolean, byte, short, int, long, float, double e string.

- I seguenti prefissi di nomi sono riservati: JMSX, JMS_.

I valori di proprietà vengono impostati prima dell'invio di un messaggio. Quando un client riceve un messaggio, le proprietà di messaggio sono di sola lettura. Se un client tenta di impostare le proprietà a questo punto, viene prodotto un errore MessageNotWriteableException. Se viene eseguita la chiamata di clearProperties, è possibile eseguire operazioni di lettura e di scrittura sulle proprietà.

Un valore di proprietà può essere o meno un duplicato di un valore in un corpo di messaggio. JMS non definisce una regola per cosa può essere eseguito o meno in una proprietà. Tuttavia, gli sviluppatori di applicazioni devono tener presente che i provider JMS gestiranno probabilmente meglio i dati in un corpo di messaggio che i dati nelle proprietà di un messaggio. Per ottenere prestazioni migliori, le applicazioni devono utilizzare le proprietà di messaggio solo quando devono eseguire la personalizzazione di un'intestazione di messaggio in quanto questa operazione consente di supportare la selezione di messaggi personalizzata.

Un selettore di messaggi JMS consente ad un client di specificare i messaggi cui è interessato utilizzando l'intestazione del messaggio. Vengono infatti recapitati solo i messaggi le cui intestazioni corrispondono al selettore.

I selettori dei messaggi non possono fare riferimento ai valori del corpo del messaggio.

Un selettore di messaggio corrisponde ad un messaggio quando viene soddisfatta con condizione ″true″ la sostituzione dei valori delle proprietà e dei campi dell'intestazione del messaggio con i corrispettivi identificativi nel selettore.

Un selettore di messaggio è un valore String, la cui sintassi è basata su un sottoinsieme della sintassi di espressione condizionale SQL92. L'ordine in cui viene valutato un selettore di messaggi è da sinistra verso destra in un livello di precedenza. E' possibile utilizzare delle parentesi per modificare quest'ordine. I valori letterali ed i nomi di operatore di selettore predefiniti sono qui scritti in caratteri maiuscoli; essi non sono tuttavia sensibili al maiuscolo/minuscolo.

Un selettore può contenere:

- Valori letterali
  - Un valore letterale di stringa è racchiuso tra apici. Un apice singolo raddoppiato all'interno della stringa rappresenta un apostrofo. Esempi sono 'dove' e 'dov''è'. In modo analogo ai valori letterali di stringa Java, utilizzano la codifica di caratteri Unicode.
  - Un valore letterale numerico esatto è un valore numerico senza un punto decimale, come ad esempio 57, -957, +62. I numeri compresi nell'intervallo dei valori long Java sono supportati.
  - Un valore letterale numerico approssimativo è un valore numerico in notazione scientifica, come ad esempio 7E3 o -57.9E2 oppure un valore numerico con una parte decimale, come ad esempio 7., -95.7 o +6.2. I numeri compresi nell'intervallo dei valori double Java sono supportati.
  - I valori letterali di tipo boolean TRUE e FALSE.
- Identificativi:
  - Un identificativo è una sequenza a lunghezza indefinita di lettere Java e numeri Java; il primo carattere deve essere una lettera Java. Una lettera è un carattere per cui il metodo Character.isJavaLetter restituisce una condizione

″true″. Ciò include “_” e “$”. Una lettera oppure un numero è un carattere per cui il metodo Character.isJavaLetterOrDigit restituisce una condizione ″true″.

  - Gli identificativi non possono essere i nomi NULL, TRUE o FALSE.
  - Gli identificativi non possono essere NOT, AND, OR, BETWEEN, LIKE, IN e IS.
  - Gli identificativi sono riferimenti a campi di intestazione oppure riferimenti a proprietà.
  - Gli identificativi sono sensibili al maiuscolo/minuscolo.
  - I riferimenti ai campi di intestazione di messaggio sono limitati a:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    I valori JMSMessageID, JMSTimestamp, JMSCorrelationID e JMSType possono essere nulli e, in questo caso, vengono trattati come un valore NULL.
  - Un nome che inizia con “JMSX” è un nome di proprietà definito JMS.
  - Un nome che inizia con “JMS_” è un nome di proprietà di un provider specifico.
  - Un nome che non inizia con “JMS” è un nome di proprietà di un’applicazione specifica. Se esiste un riferimento ad una proprietà che non esiste in un messaggio, il suo valore è NULL. Se esiste, il suo valore è il corrispettivo valore di proprietà.
- Uno spazio vuoto ha una definizione simile ai caratteri Java: spazio, tabulazione orizzontale, modulo continuo e terminatore riga.
- Espressioni:
  - Un selettore è un’espressione condizionale. Se selettore rileva una condizione ″true″ significa che ha trovato una corrispondenza; se rileva una condizione ″false″ o ″unknown″ non ha trovato una corrispondenza.
  - Le espressioni aritmetiche sono formate da espressioni aritmetiche, operazioni aritmetiche, identificativi (il cui valore viene trattato come un valore letterale numerico) e valori letterali numerici.
  - Le espressioni condizionali sono formate da espressioni condizionali, operazioni di confronto e operazioni logiche.
- E’ supportato l’utilizzo standard delle parentesi, (), per impostare l’ordine in cui vengono valutate le espressioni.
- Gli operatori logici, in ordine di precedenza: NOT, AND, OR.
- Gli operatori di messa a confronto: =, >, >=, <, <=, <> (non uguale).
  - E’ possibile mettere a confronto solo valori dello stesso tipo. L’unica eccezione è rappresentata dal fatto che è possibile mettere a confronto valori numerici esatti con valori numerici approssimativi. (Il tipo di conversione richiesto è definito dalle regole della promozione numerica Java). Se si tenta una messa a confronto di tipi diversi, il selettore rileva sempre una condizione ″false″.
  - La messa a confronto di valori di tipo ″string″ e ″boolean″ è limitata ai valori = e <>. Due stringhe sono uguali solo se contengono la stessa sequenza di caratteri.
- Gli operatori aritmetici in ordine di precedenza:

  - +, - unario.
  - *, /, moltiplicazione e divisione.
  - +, -, addizione e sottrazione.
  - Le operazioni aritmetiche su un valore NULL non sono supportate. Se viene eseguita un'operazione aritmetica su un valore NULL, il selettore di completamento rileva sempre una condizione ″false″.
  - Le operazioni aritmetiche devono utilizzare la promozione numerica Java.
- L'operatore di messa a confronto espr-aritmetica1 [NOT] BETWEEN espr-aritmetica2 e espr-aritmetica3:
  - età BETWEEN 15 and 19 è equivalente a età >= 15 AND età <= 19.
  - età NOT BETWEEN 15 and 19 è equivalente a età < 15 OR età > 19.
  - Se una delle espressioni di un'operazione BETWEEN ha un valore 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 String o NULL.
  - Paese IN (' RU', 'US', 'Francia') è ″true″ per 'RU' ma ″false″ per 'Peru'. E' equivalente all'espressione (Paese = ' RU') OR (Paese = ' US') OR (Paese = ' Francia').
  - Paese NOT IN ('RU', 'US', 'Francia') è ″false″ per 'UK' e ″true″ per 'Peru'. E' equivalente all'espressione NOT ((Paese = ' RU') OR (Paese = ' US') OR (Paese = ' Francia')).
  - Se l'identificativo di un'operazione IN o NOT IN è NULL, il valore dell'operazione è ″unknown″.
- L'operatore di messa a confronto tipo identificativo [NOT] LIKE valore-schema [ESCAPE carattere-di-escape], dove l'identificativo ha un valore ″string″- Valore schema è un valore letterale stringa, dove '_' rappresenta un singolo carattere e '%' rappresenta una sequenza di caratteri (inclusa la sequenza vuota). Tutti gli altri caratteri rappresentano se stessi. Il carattere-di-escape facoltativo è un valore letterale stringa a un carattere, il cui carattere viene utilizzato per eseguire l'escape del valore speciale di '_' e '%' nel valore-schema.
  - telefono LIKE '12%3' è ″true″ per '123' '12993' e ″false″ per '1234'.
  - parola LIKE 'l_sa' è ″true″ per 'lisa' e ″false″ per 'luisa'.
  - sottolineato LIKE '_%' ESCAPE '\' è ″true″ per '_foo' e ″false″ per 'bar'.
  - telefono NOT LIKE '12%3' è ″false″ per '123' '12993' e ″true″ per '1234'.
  - Se l'identificativo di un'operazione LIKE o NOT LIKE è NULL, il valore dell'operazione è ″unknown″.
- L'operatore di messa a confronto tipo identificativo IS NULL controlla se è presente un valore di campo di intestazione null oppure un valore di proprietà mancante.
  - nome_prop IS NULL.
- L'operatore di messa a confronto tipo identificativo IS NOT NULL controlla se esiste un valore di proprietà o un valore di campo di intestazione non null.
  - nome_prop IS NOT NULL.

Il seguente selettore di messaggi seleziona i messaggi con un tipo di messaggio ″auto″, colore ″blu″ e peso superiore ai 1200 Kg:

```
"JMSType
= 'auto' AND color = 'blu' AND weight > 1200"
```

Come notato in precedenza, i valori di proprietà possono essere NULL. La valutazione delle espressioni dei selettori che contengono valori NULL è definita dalla semantica SQL 92 NULL. Viene qui di seguito riportata una breve descrizione di questa semantica:

- SQL tratta un valore NULL come ″unknown″.
- La messa a confronto o la valutazione aritmetica con un valore ″unknown″ produce sempre un valore ″unknown″.
- Gli operatori IS NULL e IS NOT NULL convertono un valore ″unknown″ nei rispettivi valori TRUE e FALSE.

A differenza di SQL, i selettori dei messaggi JMS non supportano il confronto decimale e aritmetico. Pertanto i valori letterali numerici esatti sono limitati a quelli senza decimali. E' anche questo il motivo per cui ci sono dei valori numerici con un decimale come rappresentazione alternativa di un valore numerico approssimato.

I commenti SQL non sono supportati.

# Associazione di messaggi JMS ai messaggi MQSeries

In questa sezione verrà illustrato in che modo la struttura dei messaggi JMS descritta nella prima parte del capitolo viene associata a un messaggio MQSeries. L'argomento è destinato fondamentalmente ai programmatori che desiderano trasmettere i messaggi tra JMS e le tradizionali applicazioni di MQSeries ma risulta utile anche agli utenti che desiderano manipolare i messaggi trasmessi tra due applicazioni JMS, ad esempio nell'implementazione di un broker di messaggi.

I messaggi MQSeries sono costituiti da tre componenti:
- MQSeries Message Descriptor (MQMD)
- Un'intestazione MQRFH2MQSeries MQRFH2
- Il corpo del messaggio.

L'intestazione MQRFH2 è facoltativa e la sua inclusione in un messaggio in uscita è governata da un flag nella classe JMS Destination. E' possibile impostare questo flag utilizzando lo strumento di amministrazione MQSeries JMS. Dal momento che MQRFH2 contiene informazioni specifiche di JMS, le include sempre nel messaggio quando il mittente sa che la destinazione che riceve è un'applicazione JMS. Generalmente, è necessario omettere l'intestazione MQRFH2 nell'invio di un messaggio direttamente ad una applicazione non JMS (applicazione nativa MQSeries). Ciò accade in quanto un'applicazione del genere non prevede una MQRFH2 nel proprio messaggio MQSeries. Figura 4 indica come la struttura di un messaggio JMS viene trasformata in un messaggio MQSeries e viceversa:

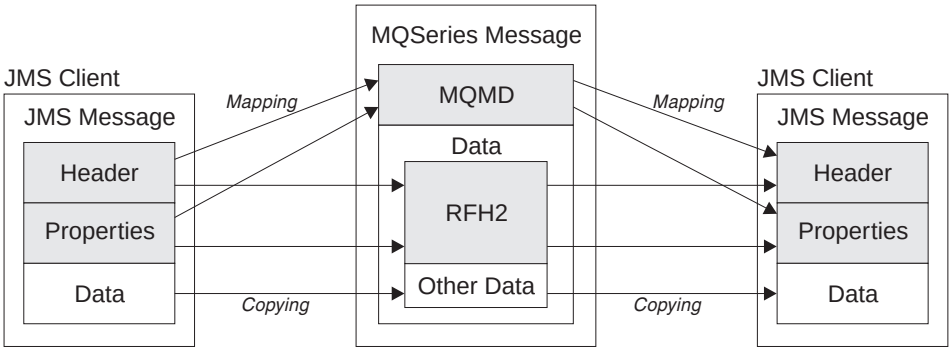


*Figura 4. Modello di associazione JMS MQSeries*

Le strutture vengono trasformate in due modi:

**Associazione**
Laddove il MQMD include un campo equivalente al campo JMS, il campo JMS viene associato al campo MQMD. I campi MQMD aggiuntivi vengono esposti come proprietà JMS in quanto è probabile che un'applicazione JMS debba richiamare o impostare questi campi durante la comunicazione con un'applicazione non JMS.

**Copia** Laddove non esiste alcun equivalente MQMD, un campo dell'intestazione JMS o la proprietà viene passata, possibilmente trasformata, come campo all'interno della MQRFH2.

## L'intestazione MQRFH2

In questa sezione viene descritta l'intestazione MQRFH Versione 2, che contiene dati specifici di JMS associati al contenuto del messaggio. MQRFH2 Versione 2 è un'intestazione estensibile e contiene inoltre informazioni aggiuntive che non sono direttamente associate a JMS. In questa sezione ne verrà comunque illustrato solo l'utilizzo da parte di JMS.

L'intestazione è composta da due parti: una parte fissa e una parte variabile.

**Parte fissa**

La parte fissa ricalca il modello dell'intestazione "standard" MQSeries ed è composta dai seguenti campi:

**StrucId (MQCHAR4)**
Identificativo della struttura.

Deve essere MQRFH_STRUC_ID (valore: "RFH ") (valore iniziale).

MQRFH_STRUC_ID_ARRAY (valore "R","F","H"," ") è anche definita nel solito modo.

**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 NameValueData.

Il valore impostato in StrucLength deve essere un multiplo di 4 (i dati nei campi NameValueData possono essere accompagnati da caratteri spazio per raggiungere questo risultato).

**Encoding (MQLONG)**
Codifica dei dati.

La codifica di tutti i dati numerici nella porzione del messaggio che segue la MQRFH2 (l'intestazione successiva o i dati del messaggio che seguono questa intestazione).

**CodedCharSetId (MQLONG)**
Identificativo del set di caratteri codificati.

Rappresentazione di tutti i dati dei caratteri nella porzione del messaggio che segue la MQRFH2 (l'intestazione successiva o i dati del messaggio che seguono questa intestazione).

**Format (MQCHAR8)**
Nome del formato.

Nome del formato relativo alla porzione del messaggio che segue la MQRFH2.

**Flags (MQLONG)**
Flag.

MQRFH_NO_FLAGS =0. Nessun flag impostato.

**NameValueCCSID (MQLONG)**
Il CCSID (coded character set identifier) per le stringhe dei caratteri NameValueData contenute in questa intestazione. NameValueData può essere codificato in un set di caratteri che differisce dalle altre stringhe di caratteri contenute nell'intestazione (StrucID e Format).

Se NameValueCCSID è un CCSID Unicode a 2 byte (1200, 13488 o 17584), l'ordine dei byte dell'Unicode equivale all'ordine dei campi numerici nella MQRFH2. Ad esempio, Version, StrucLength, NameValueCCSID stesso.

NameValueCCSID potrebbe richiedere soltanto valori dalla seguente tabella:

*Tabella 17. Valori possibili per i NameValueCCSID*

| Valore | Significato |
|---|---|
| 1200 | UCS2 open-ended |
| 1208 | UTF8 |
| 13488 | UCS2 2.0 subset |
| 17584 | UCS2 2.1 subset (include il simbolo dell'Euro) |

**Parte variabile**

La parte variabile segue la parte fissa e contiene un numero variabile di cartelle MQRFH2. Ogni cartella contiene a sua volta un variabile di elementi o proprietà. Le cartelle raggruppano insieme proprietà correlate. Le intestazioni MQRFH2 create da JMS possono contenere fino a tre cartelle:

**La cartella <mcd>**

La cartella contiene le proprità che descrivono la "forma" o il "formato" del messaggio. Per esempio, la proprietà `Msd` identifica ilmessaggio come Text, Bytes, Stream, Map, Object, o "Null". E' sempre presente in una MQRFH2 JMS.

**La cartella <jms>**

Questa cartella viene utilizzata per trasportare i campi dell'intestazione JMS e le proprietà che non possono essere espresse pienamente nel MQMD. E' sempre presente in una MQRFH2 JMS.

**La cartella <usr>**

Questa cartella viene utilizzata per trasportare tutte le proprietà definite dall'applicazione associate al messaggio. E' presente solo se l'applicazione ha impostato alcune proprietà definite dall'applicazione.

Nella Tabella 18 viene riportato un elenco completo dei nomi di proprietà.

*Tabella 18. Cartelle e proprietà MQRFH2 utilizzate da JMS*

| Nome Campo JMS | Tipo Java | Nome Cartella MQRFH2 | Nome proprietà | Tipo/valori |
|---|---|---|---|---|
| JMSDestination | Destination | jms | Dst | string |
| JMSExpiration | long | jms | Exp | i8 |
| JMSPriority | int | jms | Pri | i4 |
| JMSDeliveryMode | int | jms | Dlv | i4 |
| JMSCorrelationID | String | jms | Cid | string |
| JMSReplyTo | Destination | jms | Rto | string |
| JMSType | String | mcd | Tipo | string |
| JMSXGroupID | String | jms | Gid | string |
| JMSXGroupSeq | int | jms | Seq | i4 |

*Tabella 18. Cartelle e proprietà MQRFH2 utilizzate da JMS (Continua)*

| Nome Campo JMS | Tipo Java | Nome Cartella MQRFH2 | Nome proprietà | Tipo/valori |
|---|---|---|---|---|
| xxx (Definito dall'utente) | Any | usr | xxx | qualsiasi |
| | | mcd | Msd | jms_none<br>jms_text<br>jms_bytes<br>jms_map<br>jms_stream<br>jms_object |

La sintassi utilizzata per esprimere le proprietà nella parte variabile è la seguente:

**NameValueLength (MQLONG)**
Lunghezza espressa in byte della stringa NameValueData che segue questo campo di lunghezza (non include la propria lunghezza). Il valore impostato in NameValueLength deve essere un multiplo di 4 (nel campo NameValueData vengono aggiunti degli spazi per raggiungere questo risultato).

**NameValueData (MQCHARn)**
Una stringa di un solo carattere, la cui lunghezza in byte è determinata dal campo NameValueLength che la precede. Contiene una "cartella" che a sua volta contiene una sequenza delle "proprietà". Qgni proprietà è una tripletta "name/type/value", contenuta all'interno di un elemento XML che porta il nome dela cartella, nel modo seguente:

```
 <nomecartella> tripletta1
tripletta2 .....   triplettan </nomecartella>
```

Il tag di chiusura`</nomecartella>` può essere seguito da spazi come caratteri di riempimento. Ogni tripletta viene codificata utilizzando una sintassi simile a quella XML:

```
<nome dt='datatype'>valore</nome>
```

L'elemento `dt='datatype'` è facoltativo e viene omesso per molte proprietà il cui tipo di dati è predefinito. Se viene incluso, è necessario includere uno o più caratteri spazio prima del tag `dt=`.

`nome` è il nome della proprietà - vedere Tabella 18 a pagina 220.

`datatype` deve corrispondere,ad uno dei valori letterali riportati nella Tabella 19.

`valore` è una rappresentazione di stringa del valore da trasmettere, utilizzando la Definizione inTabella 19.

Un valore null viene codificato utilizzando la seguente sintassi:

```
<nome/>
```

*Tabella 19. Definizioni e valori del delle proprietà*

| Valore datatype | Definizione |
|---|---|
| stringa | Qualsiasi sequenza di dati escluso `<` e `&` |
| boolean | Il carattere `0` o `1` (1 = ″true″) |

*Tabella 19. Definizioni e valori del delle proprietà (Continua)*

| Valore datatype | Definizione |
|---|---|
| bin.hex | Cifre esadecimali che rappresentano ottetti |
| i1 | Un numero, espresso utilizzando le cifre `0..9`, con un segno facoltativo (che non sia frazioni o esponente). Deve essere compreso nell'intervallo che va da -128 a 127 incluso |
| i2 | Un numero, espresso utilizzando le cifre `0..9`, con un segno facoltativo (che non sia frazioni o esponente). Deve essere compreso nell'intervallo che va da -32768 a 32767 incluso |
| i4 | Un numero, espresso utilizzando le cifre `0..9`, con un segno facoltativo (che non sia frazioni o esponente). Deve essere compreso nell'intervallo che va da -2147483648 a 2147483647 incluso |
| i8 | Un numero, espresso utilizzando le cifre `0..9`, con un segno facoltativo (che non sia frazioni o esponente). Deve essere compreso nell'intervallo che va da -9223372036854775808 a 9223372036854775807 incluso |
| int | Un numero, espresso utilizzando le cifre 0..9, con un segno facoltativo (che non sia frazioni o esponente). Deve essere compreso nelo stesso intervallo come 'i8'. Può essere utilizzato al posto di uno dei tipi 'i*' se il mittente non desidera associare una determinata precisione alla proprietà |
| r4 | Numero a virgola mobile, ampiezza <= 3.40282347E+38, >= 1.175E-37 espress mediante le cifre 0..9, segno facoltativo, cifre frazionali facoltative, esponente facoltativo |
| r8 | Numero a virgola mobile, ampiezza <= 1.7976931348623E+308, >= 2.225E-307 espress mediante le cifre 0..9, segno facoltativo, cifre frazionali facoltative, esponente facoltativo |

Un valore string può contenere spazi. E' necessario utilizzare la sequenza escape riportata di seguito in un valore string:

`&` per il carattere &
`<` per il carattere <

E' possibile utilizzare le seguenti sequenze escape, ma non sono obbligatorie:

`>` per il carattere >
`&apos;` per il carattere '
`&quot;` per il carattere "

## Campi e proprietà JMS con i corrispondenti campi MQMD

Tabella 20 elenca i campi di intestazione JMS Tabella 21 a pagina 223 e le proprietà JMS direttamente associate ai campi MQMD. Tabella 22 a pagina 223 elenca le proprietà specifiche del provider ed i campi MQMD ai quali sono associati.

*Tabella 20. campi di intestazione JMS associati ai campi MQMD*

| Campi di intestazione JMS | Tipo Java | Campo MQMD | Tipo C |
|---|---|---|---|
| JMSDeliveryMode | int | Persistenza | MQLONG |
| JMSExpiration | long | Expiry | MQLONG |
| JMSPriority | int | Priority | MQLONG |
| JMSMessageID | String | MessageID | MQBYTE24 |

*Tabella 20. campi di intestazione JMS associati ai campi MQMD (Continua)*

| Campi di intestazione JMS | Tipo Java | Campo MQMD | Tipo C |
|---|---|---|---|
| JMSTimestamp | long | PutDate PutTime | MQCHAR8<br>MQCHAR8 |
| JMSCorrelationID | String | CorrelId | MQBYTE24 |

*Tabella 21. Proprietà JMS associate ai campi MQMD*

| Proprietà JMS | Tipo Java | Campo MQMD | Tipo C |
|---|---|---|---|
| JMSXUserID | String | UserIdentifier | MQCHAR12 |
| JMSXAppID | String | PutApplName | MQCHAR28 |
| JMSXDeliveryCount | int | BackoutCount | MQLONG |
| JMSXGroupID | String | GroupId | MQBYTE24 |
| JMSXGroupSeq | int | MsgSeqNumber | MQLONG |

*Tabella 22. Proprietà specifiche del provider JMS associate ai campi MQMD*

| Proprietà specifica del provider JMS | Tipo Java | Campo MQMD | Tipo C |
|---|---|---|---|
| JMS_IBM_Report_Exception | int | Report | MQLONG |
| JMS_IBM_Report_Expiration | int | Report | MQLONG |
| JMS_IBM_Report_COA | int | Report | MQLONG |
| JMS_IBM_Report_COD | int | Report | MQLONG |
| JMS_IBM_Report_PAN | int | Report | MQLONG |
| JMS_IBM_Report_NAN | int | Report | MQLONG |
| JMS_IBM_Report_Pass_Msg_ID | int | Report | MQLONG |
| JMS_IBM_Report_Pass_Correl_ID | int | Report | MQLONG |
| JMS_IBM_Report_Discard_Msg | int | Report | MQLONG |
| JMS_IBM_MsgType | int | MsgType | MQLONG |
| JMS_IBM_Feedback | int | Feedback | MQLONG |
| JMS_IBM_Format | String | Format | MQCHAR8 |
| JMS_IBM_PutApplType | int | PutApplType | MQLONG |
| JMS_IBM_Encoding | int | Encoding | MQLONG |
| JMS_IBM_Character_Set | String | CodedCharacterSetId | MQLONG |

## Associazione dei campi JMS ai campi MQSeries fields (messaggi in uscita)

Nella Tabella 23 a pagina 224 viene illustrato in che modo i campi di intestazione JMS vengono associati ai campi MQMD/RFH2 in fase send() o publish(). Nella Tabella 24 a pagina 224 viene illustrato in che modo le proprietà JMS e le proprietà specifiche del provider JMS illustrate nella Tabella 25 a pagina 224 vengono associate ai campi MQMD in fase send() o publish()

## Associazione di messaggi JMS

Per i campi contrassegnati da 'Impostato dall'oggetto Message', il valore trasmesso è il valore contenuto nel messaggio JMS immediatamente prima di send/publish(). Il valore in JMS Message non viene modificato da send/publish().

Per i campi contrassegnati dall'indicazione 'Impostato dal metodo Send', un valore viene assegnato all'esecuzione di send/publish() e qualsiasi valore contenuto nel JMS Message viene ignorato. Il valore nel messaggio JMS viene aggiornato per mostrare il valore utilizzato.

I campi contrassegnati come 'Sola ricezione' non sono trasmessi e non vengono modificati nel messaggio da send() o publish().

*Tabella 23. Associazione dei campi dei messaggi in uscita*

| Nome campo di intestazione JMS | Campo MQMD utilizzato per la trasmissione | Intestazione | Impostato da |
|---|---|---|---|
| JMSDestination | | MQRFH2 | Metodo Send |
| JMSDeliveryMode | Persistenza | MQRFH2 | Metodo Send |
| JMSExpiration | Expiry | MQRFH2 | Metodo Send |
| JMSPriority | Priority | MQRFH2 | Metodo Send |
| JMSMessageID | MessageID | | Metodo Send |
| JMSTimestamp | PutDate/PutTime | | Metodo Send |
| JMSCorrelationID | CorrelId | MQRFH2 | Oggetto Message |
| JMSReplyTo | ReplyToQ/ReplyToQMgr | MQRFH2 | Oggetto Message |
| JMSType | | MQRFH2 | Oggetto Message |
| JMSRedelivered | | | Sola ricezione |

*Tabella 24. Associazione della proprietà JMS del messaggio in uscita*

| Nome proprietà JMS | Campo MQMD utilizzato per la trasmissione | Intestazione | Impostato da |
|---|---|---|---|
| JMSXUserID | UserIdentifier | | Metodo Send |
| JMSXAppID | PutApplName | | Metodo Send |
| JMSXDeliveryCount | | | Sola ricezione |
| JMSXGroupID | GroupId | MQRFH2 | Oggetto Message |
| JMSXGroupSeq | MsgSeqNumber | MQRFH2 | Oggetto Message |

*Tabella 25. Associazione della proprietà specifica del provider JMS del messaggio in uscita*

| Nome della proprietà specifica del provider JMS | Campo MQMD utilizzato per la trasmissione | Intestazione | Impostato da |
|---|---|---|---|
| JMS_IBM_Report_Exception | Report | | Oggetto Message |
| JMS_IBM_Report_Expiration | Report | | Oggetto Message |
| JMS_IBM_Report_COA/COD | Report | | Oggetto Message |
| JMS_IBM_Report_NAN/PAN | Report | | Oggetto Message |
| JMS_IBM_Report_Pass_Msg_ID | Report | | Oggetto Message |
| JMS_IBM_Report_Pass_Correl_ID | Report | | Oggetto Message |
| JMS_IBM_Report_Discard_Msg | Report | | Oggetto Message |
| JMS_IBM_MsgType | MsgType | | Oggetto Message |

*Tabella 25. Associazione della proprietà specifica del provider JMS del messaggio in uscita (Continua)*

| Nome della proprietà specifica del provider JMS | Campo MQMD utilizzato per la trasmissione | Intestazione | Impostato da |
|---|---|---|---|
| JMS_IBM_Feedback | Feedback | | Oggetto Message |
| JMS_IBM_Format | Format | | Oggetto Message |
| JMS_IBM_PutApplType | PutApplType | | Metodo Send |
| JMS_IBM_Encoding | Encoding | | Oggetto Message |
| JMS_IBM_Character_Set | CodedCharacterSetId | | Oggetto Message |

## Associazione dei campi dell'intestazione JMS in fase di send()/publish()

Le note che seguono riguardano l'associazione dei campi JMS in fase di send()/publish():

- **JMS Destination to MQRFH2:** Viene memorizzata come stringa che serializza le caratteristiche salienti di un oggetto di destinazione in modo che un JMS che riceve possa ricostituire un oggetto di destinazione equivalente. Il campo MQRFH2 è codificato come URI (vedere "uniform resource identifiers" a pagina 194 per informazioni dettagliate sulla notazione URI).
- **JMSReplyTo to MQMD ReplyToQ, ReplyToQMgr, MQRFH2:** I nomi di Queue e QueueManager vengono copiati rispettivamente nei campi MQMD ReplyToQ e ReplyToQMgr. Le informazioni sull'estensione di destinazione (altri dettagli "utili" conservati nell'oggetto di destinazione) vengono copiate nel campo MQRFH2. Il campo MQRFH2 è codificato come URI (vedere "uniform resource identifiers" a pagina 194 per informazioni dettagliate sulla notazione URI).
- **JMSDeliveryMode to MQMD Persistence:** Il valore JMSDeliveryMode viene impostato dal metodo o MessageProducer send/publish(), a meno che l'oggetto di destinazione non lo sostituisca. Il valore JMSDeliveryMode viene associato al campo MQMD Persistence nel modo seguente:
  - Il valore JMS PERSISTENT equivale a MQPER_PERSISTENT
  - Il valore JMS NON_PERSISTENT equivale a MQPER_NOT_PERSISTENT

  Se JMSDeliveryMode è impostato su un valore non predefinito, anche il valore della modalità di consegna viene codificato nella MQRFH2.
- **JMSExpiration to/from MQMD Expiry, MQRFH2:** JMSExpiration memorizza il tempo relativo alla scadenza (la somma del tempo corrente e di quello restante), mentre MQMD memorizza quello restante. Inoltre, JMSExpiration è espresso in millisecondi, mentre MQMD.expiry è espresso in centisecondi.
  - Se il metodo send() imposta una durata illimitata, MQMD Expiry è impostato su MQEI_UNLIMITED e nessuna JMSExpiration viene codificata nella MQRFH2.
  - Se il metodo send() imposta una durata inferiore a 214748364.7 secondi (circa 7 anni), la durata è memorizzata in MQMD. La scadenza e il tempo di scadenza, espresso in millisecondi, viene codificato come valore i8 nella MQRFH2.
  - Se il metodo send() imposta una durata superiore a 214748364.7 secondi, MQMD Expiry è impostato su MQEI_UNLIMITED. L'effettivo tempo di scadenza, espresso in millisecondi, viene codificato come valore i8 nella MQRFH2.

- **JMSPriority to MQMD Priority:** Associare direttamente il valore JMSPriority (0-9) nel valore della priorità MQMD (0-9). Se JMSPriority è impostato su un valore non predefinito, anche il livello di priorità viene codificato nella MQRFH2.
- **JMSMessageID from MQMD MessageID:** Tutti i messaggi inviati da JMS hanno identificativi di messaggio univoci assegnati da MQSeries. Il valore assegnato viene restituito nel campo MQMD messageId dopo la chiamata MQPUT e passato nuovamente all'applicazione nel campo JMSMessageID. L'ID messageID MQSeries è un valore binario a 24-byte mentre JMSMessageID è un valore String. Il JMSMessageID è composto dal valore messageId binario convertito in una sequenza di 48 caratteri esadecimali, preceduti dai caratteri "ID:". JMS fornisce un'indicazione che può essere impostato per disabilitare la produzione degli identificativi dei messaggi. Questa indicazione viene ignorata e in tutti i casi viene assegnato un identificativo univoco. Qualsiasi valore impostato nel campo JMSMessageId prima di un metodo send() viene sovrascritto.
- **JMSTimestamp from MQMD PutDate, PutTime:** Dopo un invio, il campo JMSTimestamp viene impostato pari al valore data/ora fornito dai campi MQMD PutDate e PutTime. Qualsiasi valore impostato nel campo JMSMessageId prima di un metodo send() viene sovrascritto.
- **JMSType to MQRFH2:** Questa stringa viene impostata nella MQRFH2.
- **JMSCorrelationID to MQMD CorrelId, MQRFH2:** JMSCorrelationID può contenere una delle seguenti opzioni:
  - **Un ID messaggio specifico del provider:** Questo è un identificativo del messaggio precedentemente inviato o ricevuto e dovrebbe quindi essere una stringa di 48 cifre esadecimali precedute da "ID:". Il prefisso viene rimosso e i caratteri che restano vengono convertiti in binari, quindi impostati nel campo MQMD CorrelId. Nessun valore CorrelId viene codificato nella MQRFH2.
  - **Un valore provider-native byte[]:** Il valore viene copiato nel campo MQMD CorrelId, riempito con valori null o troncato fino a 24 byte se necessario. Nessun valore CorrelId viene codificato nella MQRFH2.
  - **Una stringa specifica dell'applicazione:** Il valore viene copiato nella MQRFH2. I primi 24 byte della stringa, in formato UTF8, vengono scritti nel MQMD CorrelID.

## Associazione dei campi delle proprietà JMS

Queste note fanno riferimento all'associazione dei campi delle proprietà JMS nei messaggi MQSeries:

- **JMSXUserID from MQMD UserIdentifier:** JMSXUserID è impostato su return from send call.
- **JMSXAppID da MQMD PutApplName:** JSMXAppID è impostato su return from send call.
- **Da JMSXGroupID a MQRFH2 (point-to-point):** Per i messaggi point-to-point, JMSXGroupID viene copiato nel campo MQMD GroupID. Se JMSXGroupID viene avviato con il prefisso "ID:", viene convertito in binario. Altrimenti viene codificato come una stringa UTF8. Il valore viene riempito o troncato, se necessario, per raggiungere una lunghezza di 24 byte. Il flag MQF_MSG_IN_GROUP viene impostato.
- **Da JMSXGroupID a MQRFH2 (publish/subscribe):** Per messaggi di tipo publish/subscribe, JMSXGroupID viene copiato nella MQRFH2 come stringa.
- **JMSXGroupSeq MQMD MsgSeqNumber (point-to-point):** Per i messaggi point-to-point, JMSXGroupSeq viene copiato nel campo MQMD MsgSeqNumber. Il flag MQF_MSG_IN_GROUP viene impostato.

- **JMSXGroupSeq MQMD MsgSeqNumber (publish/subscribe):** Per i messaggi di tipo publish/subscribe, JMSXGroupSeq viene copiato nella MQRFH2 come un i4.

## Associazione di campi specifici del provider JMS

Queste note fanno riferimento all'associazione dei campi specifici del Provider JMS nei messaggi MQSeries:

- Da **JMS_IBM_Report_<nome> a MQMD Report:** Un'applicazione JMS può impostare le opzioni MQMD Report, utilizzando le seguenti proprietà JMS_IBM_Report_XXX. Il singolo MQMD è associato a diverse proprietà JMS_IBM_Report_XXX. L'applicazione dovrebbe impostare il valore di queste proprietà sulle costanti MQSeries MQRO_ standard (incluse in com.ibm.mq.MQC). Pertanto, ad esempio, per richiedere COD con dati completi, l'applicazione dovrebbe impostare JMS_IBM_Report_COD sul valore MQC.MQRO_COD_WITH_FULL_DATA.

  **JMS_IBM_Report_Exception**

  MQRO_EXCEPTION or
  MQRO_EXCEPTION_WITH_DATA or
  MQRO_EXCEPTION_WITH_FULL_DATA

  **JMS_IBM_Report_Expiration**

  MQRO_EXPIRATION o
  MQRO_EXPIRATION_WITH_DATA o
  MQRO_EXPIRATION_WITH_FULL_DATA

  **JMS_IBM_Report_COA**

  MQRO_COA o
  MQRO_COA_WITH_DATA o
  MQRO_COA_WITH_FULL_DATA

  **JMS_IBM_Report_COD**

  MQRO_COD o
  MQRO_COD_WITH_DATA o
  MQRO_COD_WITH_FULL_DATA

  **JMS_IBM_Report_PAN**
  MQRO_PAN

  **JMS_IBM_Report_NAN**
  MQRO_NAN

  **JMS_IBM_Report_Pass_Msg_ID**
  MQRO_PASS_MSG_ID

  **JMS_IBM_Report_Pass_Correl_ID**
  MQRO_PASS_CORREL_ID

  **JMS_IBM_Report_Discard_Msg**
  MQRO_DISCARD_MSG

- **Da JMS_IBM_MsgType a MQMD MsgType:** Il valore viene associato direttamente in MQMD MsgType. Se l'applicazione non ha impostato un valore esplicito di JMS_IBM_MsgType, verrà utilizzato un valore predefinito. Tale valore è determinato nel modo seguente:
  - Se JMSReplyTo è impostato su una destinazione della coda MQSeries, MSGType è impostato sul valore MQMT_REQUEST

  - Se JMSReplyTo non è impostato oppure è impostato su qualsiasi altro valore che non sia una destinazione della coda MQSeries, MsgType viene impostato sul valore MQMT_DATAGRAM
- **Da JMS_IBM_Feedback a MQMD Feedback:** Il valore viene associato direttamente in MQMD Feedback.
- **Da JMS_IBM_Format a MQMD Format:** Il valore viene associato direttamente in MQMD Format.
- **Da JMS_IBM_Encoding a MQMD Encoding:** Se impostata, questa proprietà sostituisce la codifica numerica di Destination Queue o Topic.
- **JMS_IBM_Character_Set to MQMD CodedCharacterSetId:** Se è impostata, questa proprietà sostituisce la proprietà dell'insieme dei caratteri codificati della Destination Queue o Topic.

## Associazione dei campi MQSeries ai campi JMS (messaggi in arrivo)

Nella Tabella 26 viene illustrato in che modo i campi di intestazione JMS ed i campi della proprietà JMS illustrate nella Tabella 27, vengono associati ai campi MQMD/MQRFH2 in fase send() o publish(). Nella Tabella 28 a pagina 229 viene illustrato in che modo le proprietà specifiche del provider JMS vengono associate.

*Tabella 26. Associazione dei campi di intestazione JMS dei messaggi in arrivo*

| Nome campo di intestazione JMS | Campo MQMD richiamato da | Campo MQRFH2 richiamato da |
|---|---|---|
| JMSDestination | | jms.Dst |
| JMSDeliveryMode | Persistenza | |
| JMSExpiration | | jms.Exp |
| JMSPriority | Priority | |
| JMSMessageID | MessageID | |
| JMSTimestamp | PutDate PutTime | |
| JMSCorrelationID | CorrelId | jms.Cid |
| JMSReplyTo | ReplyToQ ReplyToQMgr | jms.Rto |
| JMSType | | mcd.Type |
| JMSRedelivered | BackoutCount | |

*Tabella 27. Associazione della proprietà del messaggio in arrivo*

| Nome proprietà JMS | Campo MQMD richiamato da | Campo MQRFH2 richiamato da |
|---|---|---|
| JMSXUserID | UserIdentifier | |
| JMSXAppID | PutApplName | |
| JMSXDeliveryCount | BackoutCount | |
| JMSXGroupID | GroupId | jms.Gid |
| JMSXGroupSeq | MsgSeqNumber | jms.Seq |

*Tabella 28. Associazione della proprietà JMS specifica del provider del messaggio in arrivo*

| Nome proprietà JMS | Campo MQMD richiamato da | Campo MQRFH2 richiamato da |
|---|---|---|
| JMS_IBM_Report_Exception | Report | |
| JMS_IBM_Report_Expiration | Report | |
| JMS_IBM_Report_COA | Report | |
| JMS_IBM_Report_COD | Report | |
| JMS_IBM_Report_PAN | Report | |
| JMS_IBM_Report_NAN | Report | |
| JMS_IBM_Report_ Pass_Msg_ID | Report | |
| JMS_IBM_Report_Pass_Correl_ID | Report | |
| JMS_IBM_Report_Discard_Msg | Report | |
| JMS_IBM_MsgType | MsgType | |
| JMS_IBM_Feedback | Feedback | |
| JMS_IBM_Format | Format | |
| JMS_IBM_PutApplType | PutApplType | |
| JMS_IBM_Encoding[1] | Encoding | |
| JMS_IBM_Character_Set[1] | CodedCharacterSetId | |
| 1. Impostato solo se il messaggio in arrivo è un Bytes Message. | | |

## Associazione di JMS ad un'applicazione MQSeries nativa

In questa sezione verrà descritto cosa accade se si invia un messaggio da un'applicazione Client JMS ad un'applicazione tradizionale MQSeries che non ha alcuna conoscenza delle intestazioni MQRFH2. La Figura 5 a pagina 230 è un diagramma dell'associazione.

L'amministratore indica che il Client JMS sta comunicando con con un'applicazione del genere impostando il valore TargetClient di Destinazione MQSeries su JMSC.MQJMS_CLIENT_NONJMS_MQ. Ciò indica che non deve essere prodotto alcun campo MQRFH2. Se tale operazione non viene eseguita, l'applicazione di ricezione deve essere in grado di gestire il campo MQRFH2.

L'associazione da JMS a MQMD indirizzata ad un'applicazione MQSeries Nativa equivale all'associazione da JMS a MQMD indirizzata ad un client JMS. Se JMS riceve un messaggio MQSeries con il campo MQMD Format impostato su un valore diverso da MQFMT_RFH2, è evidente che i dati vengono ricevuti da un'applicazione non JMS. Se il Format è MQFMT_STRING, il messaggio viene ricevuto come messaggio JMS Text. Altrimenti viene ricevuto come messaggio JMS Bytes. Dal momento che non c'è alcuna MQRFH2, è possibile ripristinare solo le proprietà JMS trasmesse nella MQMD.

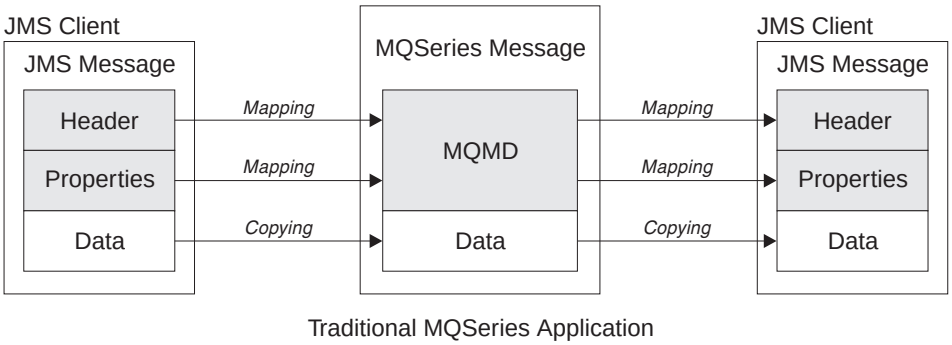


*Figura 5. Modello di associazione JMS MQSeries*

## Corpo del messaggio

In questa sezione verrà illustrata la codifica del corpo del messaggio stesso. La codifica dipende dal tipo di messaggio JMS:

**ObjectMessage**
è un oggetto serializzato dal Runtime Java nel modo consueto.

**TextMessage**
è una stringa codificata. Per un messaggio in uscita, la stringa viene codificata nel set di caratteri fornito dal''oggetto Destination. La codifica predefinita è la UTF8 che inizia con il primo carattere del messaggio. Non c'è nessun campo della lunghezza all'inizio. Tuttavia è possibile specificare qualsiasi altro set di caratteri supportato da MQ Java. Tali set di caratteri vengono utilizzati principalmente quando si invia un messaggio a un'applicazione non JMS.

Se il set di caratteri è di tipo double-byte (incluso UTF16), la specifica della codifica del valore intero dell'oggetto Destination determina l'ordine dei byte.

Un messaggio in arrivo viene interpretato utilizzando un set di caratteri e una codifica specificati nel messaggio stesso. Tali specifiche si trovano nell'intestazione MQSeries posta sull'estrema destra (o MQMD se non ci sono intestazioni). Per i messaggi JMS, l'intestazione posta sull'estrema destra sarà generalmente la MQRFH2.

**BytesMessage**
è, in base all'impostazione predefinita, una sequenza di byte definita dalla specifica JMS 1.0.2 e la documentazione Java associata.

Per un messaggio in uscita assemblato dall'applicazione stessa, la proprietà di codifica dell'oggetto Destination può essere utilizzato per sostituire le codifiche dei campi dei valori interi e a virgola mobile contenuti nel messaggio. Ad esempio, è possibile richiedere che i valori a virgola mobile siano memorizzati nel formato S/390 anziché IEEE.

Un messaggio in arrivo viene interpretato utilizzando la codifica numerica specificata nel messaggio stesso. Tale specifica si trova nell'intestazione MQSeries posta sull'estrema destra (o MQMD se non ci sono intestazioni). Per i messaggi JMS, l'intestazione posta sull'estrema destra sarà generalmente la MQRFH2.

Se si riceve un BytesMessage, e viene inviato di nuovo senza modifiche, il corpo viene trasmesso byte per byte, nel modo in cui è stato ricevuto. La

proprietà di codifica dell'oggetto Destination non ha effetto sul corpo. L'unica entità simile a una stringa che può essere inviata esplicitamente in un BytesMessage è una stringa UTF8. Questa è codificata in formato UTF8 Java e inizia con un campo della lunghezza di 2 byte. La proprietà del set di caratteri dell'oggetto Destination non ha effetto sulla codifica di un BytesMessage in uscita. Il valore dell'insieme dei caratteri in un messggio MQSeries in arrivo non ha effetto sull'interpretazione di quel messaggio come un JMS BytesMessage.

E' improbabile che le applicazioni non Java riconoscano la codifica UTF8 Java. Pertanto, affinché un'applicazione JMS invii un BytesMessage contenente dati di testo, è necessario che l'applicazione stessa converta le proprie stringhe in matrici di byte e che scriva queste matrici nel BytesMessage.

**MapMessage**
è una stringa contenente una serie di triplette XML nome/tipo/valore, codificate come:

```
<map><nomeelemento1
dt='datatype'>valore</nomeelemento1>
<nomeelemento2 dt='datatype'>valore</nomeelemento2>.....
</map>
```

dove:
`datatype` può assumere uno dei valori descritti nella Tabella 19 a pagina 221.
`stringa` è il tipo di dati predefinito, pertanto `dt='string'` è omesso.

Il set di caratteri utilizzato per codificare o interpretare la stringa XML che compone il corpo di MapMessage viene determinato seguendo le regole che si applicano a un TextMessage.

**StreamMessage**
è come un'associazione, ma senza nomi di elementi:

```
<stream><elt
dt='datatype'>valore</elt>
<elt dt='datatype'>valore</elt>.....</stream>
```

Ogni elemento viene inviato utilizzando lo stesso nome di tag (elt). Il tipo predefinito è string, pertanto `dt='string'` è omesso per gli elementi stringa.

Il set di caratteri utilizzato per codificare o interpretare la stringa XML che compone il corpo di StreamMessage viene determinato seguendole regole che si applicano a un TextMessage.

Il campo MQRFH2.format viene impostato nel modo seguente:

**MQFMT_NONE**
per ObjectMessage, BytesMessage o messaggi senza corpo.

**MQFMT_STRING**
per TextMessage, StreamMessage o MapMessage.

## Associazione di messaggi JMS

# Capitolo 13. Application Server Facilities MQ JMS

MQ JMS V5.2 supporta le funzioni Application Server Facilities (ASF) specificate nella specifica Java Message Service 1.0.2 (visitare il sito Web della Sun Java all'indirizzo `http://java.sun.com`). Tale specifica identifica tre ruoli all'interno del modello di programmazione:

- **Il provider JMS** fornisce la funzionalità ConnectionConsumer e Session avanzata.
- **Il server applicazioni** fornisce la funzionalità ServerSessionPool e ServerSession.
- **L'applicazione client** utilizza la funzionalità fornita dal provider JMS e dal server applicazioni.

Nelle sezioni che seguono verrà descritta in dettaglio l'implementazione di ASF da parte di MQ JMS:

- In "Classi e funzioni ASF" verrà spiegato come MQ JMS implementa la classe ConnectionConsumer e la funzionalità avanzata nella classe Session.
- In "Codice di esempio del server applicazioni" a pagina 240 verrà illustrato il codice di esempio di ServerSessionPool e ServerSession fornito con MQ JMS.
- In "Esempi di utilizzo di ASF" a pagina 244 verranno descritti gli esempi di ASF forniti ed esempi di utilizzo di ASF dalla prospettiva di un'applicazione client.

**Nota:** La specifica Java Message Service 1.0.2 per ASF descrive inoltre il supporto JMS per le transazioni distribuite mediante il protocollo X/Open XA. Per informazioni dettagliate sul supporto XA fornito da MQ JMS, fare riferimento all'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401.

## Classi e funzioni ASF

MQ JMS implementa la funzione ConnectionConsumer e la funzionalità avanzata nella classe Session. Per ulteriori dettagli, vedere:

- "MQPoolServices" a pagina 138
- "MQPoolServicesEvent" a pagina 139
- "MQPoolToken" a pagina 141
- "MQPoolServicesEventListener" a pagina 172
- "ConnectionConsumer" a pagina 270
- "QueueConnection" a pagina 322
- "Session" a pagina 336
- "TopicConnection" a pagina 355

### ConnectionConsumer

La specifica JMS consente una stretta integrazione tra un server applicazioni e un'implementazione JMS mediante l'interfaccia `ConnectionConsumer`. Questa funzione fornisce un'elaborazione simultanea dei messaggi. Generalmente un server applicazioni crea un gruppo di thread e l'implementazione JMS rende disponibili i messaggi ai thread. Un server applicazioni JMS-compatibile può utilizzare questa funzione per fornire una funzionalità dei messaggi di alto livello, sotto forma ad esempio di bean di elaborazione dei messaggi.

Le normali applicazioni non utilizzano l'interfaccia ConnectionConsumer, a differenza dei client JMS avanzati che potrebbero utilizzarla. Per questo tipo di client, ConnectionConsumer offre un metodo ad elevate prestazioni per la consegna contemporanea dei messaggi a una serie di thread. Quando un messaggio arriva su una coda o su un argomento, JMS seleziona un thread dal pool di thread, consegnando un batch di messaggi. Per farlo, esegue un metodo `onMessage()` di MessageListener associato.

E' possibile ottenere lo stesso effetto creando più oggetti Session e MessageConsumer,ciascuno con un MessageListener registrato. Tuttavia l'interfaccia ConnectionConsumer offre prestazioni migliori, un impiego più limitato di risorse e una maggiore flessibilità. In particolare sono necessari meno oggetti Session.

Per facilitare lo sviluppo di applicazioni che utilizzano ConnectionConsumers, MQ JMS offre un'implementazione di esempio completamente funzionante di un pool. Questa implementazione può essere utilizzata senza alcuna modifica o può essere adattata alle esigenze specifiche dell'applicazione.

## Pianificazione di un'applicazione

### Principi generali della messaggistica point-to-point

Quando un'applicazione crea una ConnectionConsumer da un oggetto QueueConnection, specifica un oggetto Queue JMS e una stringa del selettore. ConnectionConsumer inizia quindi a ricevere messaggi o, ad essere precisi, a fornire messaggi a oggetti Session nel ServerSessionPool associato. I messaggi arrivano sulla coda e, se corrispondono al selettore, vengono consegnati a oggetti Session nel ServerSessionPool associato.

In termini MQSeries, l'oggetto Queue fa riferimento a QLOCAL o a QALIAS nel Gestore code locale. Se si tratta di un QALIAS, esso deve fare riferimento a un QLOCAL. Il QLOCAL MQSeries completamente risolto è conosciuto come *QLOCAL sotostante*. Un'interfaccia ConnectionConsumer viene definita *attiva* se non è chiusa e la QueueConnection principale viene avviata.

E' possibile l'esecuzione di più ConnectionConsumer, ciascuna con diversi selettori, sullo stesso QLOCAL sottostante. Per mantenere le prestazioni, i messaggi non desiderati non devono accumularsi sulla coda. Per messaggi non desiderati si intendono quei messaggi per i quali nessuna ConnectionConsumer ha un selettore corrispondente. E' possibile impostare QueueConnectionFactory in modo che i messaggi non desiderati siano rimossi dalla coda. Per informazioni dettagliate, fare riferimento alla sezione "Eliminazione di messaggi dalla coda" a pagina 237. E' possibile impostare questo comportamento in uno dei seguenti modi:

- Utilizzare lo strumento di amministrazione JMS per impostare QueueConnectionFactory su MRET(NO).
- Nel proprio programma, utilizzare:

  ```
  MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
  ```

Se questa impostazione non viene modificata, i messaggi non desiderati verranno automaticamente mantenuti sulla coda.

E' possibile che le ConnectionConsumers rivolte verso lo stesso QLOCAL sottostante siano state create da più oggetti QueueConnection. Tuttavia, per motivi di prestazione, si consiglia di evitare che più JVM creino ConnectionConsumer sullo stesso QLOCAL sottostante.

Durante l'impostazione del Gestore code MQSeries, tenere conto di quanto segue:

- Il QLOCAL sottostante deve essere abilitato per un input condiviso. Per eseguire questa operazione, utilizzare il comando MQSC:

  ```
  ALTER QLOCAL(nome.qlocal) SHARE GET(ENABLED)
  ```

- E' necessario che il gestore code disponga di una coda delle lettere non consegnate abilitata. Se una classe ConnectionConsumer rileva un problema durante l'inserimento di un messaggio sulla coda delle lettere non consegnate, la consegna dei messaggi sul QLOCAL sottostante si interrompe. Per definire una coda delle lettere non consegnate, utilizzare:

  ```
  ALTER QMGR
  DEADQ(nome.coda.lettere.non.consegnate)
  ```

- L'utente che esegue ConnectionConsumer deve disporre dell'autorizzazione a eseguire MQOPEN con MQOO_SAVE_ALL_CONTEXT e MQOO_PASS_ALL_CONTEXT. Per informazioni dettagliate, consultare la documentazione MQSeries relativa alla piattaforma specifica.
- Se i messaggi non desiderati vengono lasciati sulla coda, influiscono negativamente sulle prestazioni del sistema. Pertanto è necessario programmare i selettori dei messaggi in modo che tra loro le classi ConnectionConsumer rimuovano tutti i messaggi dalla coda.

Per informazioni dettagliate sui comandi MQSC, fare riferimento a *MQSeries MQSC Command Reference*.

## Principi generali sulla messaggistica publish/subscribe

Quando un'applicazione crea una ConnectionConsumer da un oggetto TopicConnection, specifica un oggetto Topic e una stringa del selettore. La classe ConnectionConsumer inizia quindi a ricevere messaggi su quel Topic che corrispondono al selettore.

In alternativa un'applicazione può creare una classe ConnectionConsumer durevole associata a un nome specifico. Questa ConnectionConsumer riceve messaggi pubblicati sul Topic dall'ultimo momento di attività della ConnectionConsumer durevole. Riceve tutti questi messaggi sull'oggetto Topic che corrisponde al selettore.

Per le sottoscrizioni non durevoli, viene utilizzata una coda separata per le sottoscrizioni ConnectionConsumer. L'opzione configurabile CCSUB su TopicConnectionFactory specifica la coda da utilizzare. Generalmente l'opzione CCSUB dovrebbe specificare una singola coda destinata all'utilizzo da parte di tutte le ConnectionConsumers che utilizzano la stessa TopicConnectionFactory. E' tuttavia possibile fare in modo che ciascuna ConnectionConsumer crei una coda temporanea specificando un prefisso del nome della coda seguito da un '*'.

Per sottoscrizioni durevoli, la proprietà CCDSUB dell'oggetto Topic specifica la coda da utilizzare. Può trattarsi di una coda che esiste già o del prefisso di un nome di coda seguito da un '*'. Se si specifica una coda che esiste già, tutte le ConnectionConsumers durevoli che si sottoscrivono all'oggetto Topic utilizzeranno questa coda. Se si specifica un prefisso di nome di coda seguito da un '*', la prima volta che una ConnectionConsumer durevole viene creata con un determinato nome, viene generata una coda. La coda in questione viene riutilizzata successivamente quando una ConnectionConsumer durevole viene creata con lo stesso nome.

Durante l'impostazione del Gestore code MQSeries, tenere conto di quanto segue:

- E' necessario che il gestore code disponga di una coda delle lettere non consegnate abilitata. Se una classe ConnectionConsumer rileva un problema durante l'inserimento di un messaggio sulla coda delle lettere non consegnate, la consegna dei messaggi sul QLOCAL sottostante si interrompe. Per definire una coda delle lettere non consegnate, utilizzare:

  ```
  ALTER QMGR DEADQ(nome.coda.lettere.non
  consegnate)
  ```

- L'utente che esegue ConnectionConsumer deve disporre dell'autorizzazione a eseguire MQOPEN con MQOO_SAVE_ALL_CONTEXT e MQOO_PASS_ALL_CONTEXT. Per informazioni dettagliate, consultare la documentazione MQSeries relativa alla piattaforma specifica.
- E' possibile ottimizzare le prestazioni per una singola ConnectionConsumer creando un'apposita coda dedicata e separata. Questa operazione presuppone però un impiego extra di risorse.

## Gestione di messaggi poison

Talvolta sulla coda arriva un messaggio formattato male che causa un errore nell'applicazione ricevente e il backout della ricezione del messaggio. In questa situazione un messaggio del genere potrebbe essere ripetutamente ricevuto e quindi restituito alla coda. Questo tipo di messaggi è conosciuto come *messaggi poison*. ConnectionConsumer deve essere in grado di riconoscerli e re-indirizzarli verso una destinazione alternativa.

Quando un'applicazione utilizza le classi ConnectionConsumer, le circostanze in cui un messaggi viene respinto dipendono dalla classe Session fornita dal server applicazioni:

- Quando la classe Session non è sottoposta a transazione, con AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE, un messaggio viene respinto solo dopo un errore di sistema o se l'applicazione viene chiusa inaspettatamente.
- Quando la classe Session non è sottoposta a transazione con CLIENT_ACKNOWLEDGE, i messaggi non riconosciuti possono essere respinti dal server applicazioni che richiama `Session.recover()`.

  Generalmente l'implementazione client MessageListener o il server applicazioni richiamano `Message.acknowledge()`. `Message.acknowledge()` riconosce tutti i messaggi consegnati sulla sessione fino a quel momento.
- Quando la classe Session è sottoposta a transazione, generalmente il server applicazioni esegue il commit di Session. Se invece rileva un errore, potrebbe scegliere di respingere uno o più messaggi.
- Se il server applicazioni fornisce una XASession, i messaggi vengono respinti o ne viene eseguito il commit in base a una transazione distribuita. E' responsabilità del server applicazioni il completamento della transazione.

Il Gestore code MQSeries mantiene un record del numero di volte in cui ciascun messaggio è stato respinto. Quando questo numero raggiunge una soglia configurabile, ConnectionConsumer riaccoda il messaggio su una determinata coda dei messaggi respinti. Se per una ragione qualsiasi il riaccodamento non riesce, il messaggio viene eliminato dalla coda e riaccodato alla coda delle lettere non consegnate o cancellato. Consultare la sezione "Eliminazione di messaggi dalla coda" a pagina 237 per informazioni più dettagliate.

Su molte piattaforme, la soglia e la coda del riaccodamento sono proprietà di QLOCAL MQSeries. Per la messaggistica point-to-point, dovrebbe trattarsi del QLOCAL sottostante. Per la messaggistica publish/subscribe, si tratta della coda

CCSUB definita in TopicConnectionFactory o la coda CCDSUB definita nel Topic. Per impostare la soglia e le proprietà della coda di riaccodamento, emettere il seguente comando MQSC:

```
ALTER QLOCAL(nome.coda) BOTHRESH(soglia) BOQUEUE(nome.coda.riaccodamento)
```

Per la messaggistica publish/subscribe, se il sistema crea una coda dinamica per ciascuna sottoscrizione, queste impostazioni sono ottenute dalla coda modello MQ JMS. Per modificare queste impostazioni, è possibile utilizzare:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(soglia) BOQUEUE(nome.coda.riaccodamento)
```

Se la soglia è pari a zero, la gestione dei messaggi poison è disabilitata e i messaggi poison rimarranno sulla coda di input. Altrimenti, quando il numero di rifiuti raggiunge la soglia, il messaggio viene inviato alla coda di riaccodamento. Se il numero di rifiuti raggiunge la soglia, ma il messaggio non riesce a raggiungere la coda di riaccodamento, il messaggio viene inviato alla coda delle lettere non consegnate o cancellato. Questa situazione si verifica se la coda di riaccodamento non è definita oppure se ConnectionConsumer non riesce a inviare il messaggio alla coda di riaccodamento. In alcune piattaforme non è possibile specificare la soglia e le proprietà della coda di riaccodamento. In queste piattaforme i messaggi vengono inviati alla coda delle lettere non consegnate o cancellati, quando il numero di rifiuti raggiunge quota 20. Per ulteriori informazioni, consultare la sezione “Eliminazione di messaggi dalla coda”.

## Eliminazione di messaggi dalla coda

Quando un’applicazione utilizza ConnectionConsumers, è possibile che JMS debba rimuovere i messaggi dalla coda in una serie di situazioni:

**Messaggio formattato in modo errato**
E’ possibile che arrivi un messaggio che JMS non è in grado di analizzare.

**Messaggio poison**
E’ possibile che un messaggio raggiunga la soglia di rifiuto, ma che ConnectionConsumer non riesca a riaccodarlo sulla coda dei rifiuti.

**ConnectionConsumer non interessata**
Per la messaggistica point-to-point, quando QueueConnectionFactory è impostata in modo da non conservare i messaggi non desiderati, arriva un messaggio che non desiderato da alcuna classe ConnectionConsumer.

In queste situazioni ConnectionConsumer tenta di rimuovere il messaggio dalla coda. Le opzioni di disposizione nel campo MQMD del messaggio impostano l’esatto comportamento. Le opzioni sono:

**MQRO_DEAD_LETTER_Q**
Il messaggio viene riaccodato alla coda delle lettere non consegnate del gestore code. Questa è l’impostazione predefinita.

**MQRO_DISCARD_MSG**
Il messaggio viene cancellato.

ConnectionConsumer genera anche un messaggio di prospetto e questo dipende anche dal campo di prospetto del MQMD del messaggio. Questo messaggio viene inviato al ReplyToQ el messaggio su ReplyToQmgr. Se si verifica un errore durante l’invio del messaggio di prospetto, il messaggio viene inviato invece alla coda delle lettere non consegnate. Le opzioni di prospetto di eccezione nel relativo campo del MQMD del messaggio impostano i dettagli del messaggio di prospetto. Le opzioni sono:

**MQRO_EXCEPTION**
Viene generato un messaggio di prospetto che contiene il MQMD del messaggio originale. Non contiene alcun dato del corpo del messaggio.

**MQRO_EXCEPTION_WITH_DATA**
Viene generato un messaggio di prospetto che contiene il MQMD, le eventuali intestazioni MQ e 100 byte di dati del corpo del messaggio.

**MQRO_EXCEPTION_WITH_FULL_DATA**
Viene generato un messaggio di prospetto che contiene tutti i dati dal messaggio originale.

**default**
Non viene generato alcun messaggio di prospetto.

Quando vengono generati i messaggi di prospetto, vengono rispettate le seguenti opzioni:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Se una ConnectionConsumer non riesce a seguire le opzioni di disposizione o le opzioni di prospetto di errore, nel MQMD del messaggio l'azione dipende dalla persistenza del messaggio. Se il messaggio non è persistente, viene cancellato e non viene creato alcun messaggio di prospetto. Se il messaggio è persistente, la consegna di tutti i messaggi dal QLOCAL si interrompe.

E' quindi importante definire una coda delle lettere non consegnate e controllarla periodicamente per assicurarsi che non si verifichi alcun problema. In particolare assicurarsi che la coda delle lettere non consegnate non raggiunga la sua profondità massima e che la dimensione massima del messaggio sia sufficiente per tutti i messaggi.

Quando un messaggio viene riaccodato alla coda delle lettere non consegnate, viene preceduto da un'intestazione delle lettere non consegnate MQSeries (MQDLH). Consultare *MQSeries Application Programming Reference* per informazioni dettagliate sul formato di MQDLH. E' possibile identificare i messaggi che una classe ConnectionConsumer ha inserito nella coda delle lettere non consegnate o i messaggi di prospetto generati da una ConnectionConsumer in base ai seguenti campi:

- PutApplType è `MQAT_JAVA (0x1C)`
- PutApplName è `"MQ JMS ConnectionConsumer"`

Questi campi sono nel MQDLH dei messaggi sulla coda delle lettere non consegnate e il MQMD dei messaggi di prospetto. Il campo di feedback del MQMD e il campo Reason del MQDLH contengono un codice che descrive l'errore. Per informazioni dettagliate su questi codici, consultare la sezione "Gestione degli errori". Altri campi rispettano le descrizioni riportate in *MQSeries Application Programming Reference*.

## Gestione degli errori

### Ripristino da condizioni di errore

Se una classe ConnectionConsumer rileva un errore grave, la consegna dei messaggi a tutte le classi ConnectionConsumers con un interesse nello stesso

QLOCAL si interrompe. Generalmente ciò si verifica se la classe ConnectionConsumer non riesce a riaccodare un messaggio nella coda delle lettere non consegnate o se rileva un errore durante la lettura dei messaggi da QLOCAL.

Quando ciò accade, l'applicazione e il server applicazioni vengono notificati nel modo seguente:

- Viene eseguita la notifica di qualsiasi ExceptionListener registrato con la Connection interessata.

E' possibile utilizzarle per identificare la causa del problema. In alcuni casi, l'amministratore del sistema deve intervenire per risolvere il problema.

Sono disponibili due modi in cui un'applicazione può recuperare da una condizione di errore:

- Richiamare `close()` su tutte le ConnectionConsumer interessate. L'applicazione può creare nuove ConnectionConsumers solo dopo la chiusura di tutte le ConnectionConsumers interessate e la risoluzione di tutti gli eventuali problemi del sistema.
- Richiamare `stop()` su tutte le Connection interessate. Una volta interrotte tutte le Connection e risolti gli eventuali problemi del sistema, l'applicazione dovrebbe essere in grado di avviare (`start()`) tutte le Connection correttamente.

## Codici motivo e feedback

Per determinare la causa di un errore, è possibile utilizzare:

- Il codice di feedback in qualsiasi messaggio di prospetto
- Il codice motivo nel MQDLH di tutti i messaggi nella coda delle lettere non consegnate

### classi e funzioni ASF

Le classi ConnectionConsumer generano i seguenti codici motivo.

**MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)**

**Causa** Il messaggio raggiunge la soglia di backout definita sul QLOCAL, ma non viene definita alcuna coda di backout.

Sulle piattaforme in cui non è possibile definire la coda di backout, il massaggio raggiunge la soglia definita da JMS di 20.

**Azione** Per evitare questa situazione, assicurarsi che le classi ConnectionConsumers che utilizzano la coda forniscono una serie di selettori che gestiscono tutti i messaggi oppure impostare QueueConnectionFactory per mantenere i messaggi.

In alternativa, studiare la fonte del messaggio.

**MQRC_MSG_NOT_MATCHED (0x93B; 2363)**

**Causa** Nella messaggistica point-to-point c'è un messaggio che non corrisponde ad alcun selettore per le ConnectionConsumers che controllano la coda. Per mantenere le prestazioni, il messaggio viene riaccodato alla coda delle lettere non consegnate.

**Azione** Per evitare questa situazione, assicurarsi che le classi ConnectionConsumers che utilizzano la coda forniscono una serie di selettori che gestiscono tutti i messaggi oppure impostare QueueConnectionFactory per mantenere i messaggi.

In alternativa, studiare la fonte del messaggio.

**MQRC_JMS_FORMAT_ERROR (0x93C; 2364)**

**Causa** JMS non è in grado di interpretare il messaggio sulla coda.

**Azione** Studiare la fonte del messaggio. JMS consegna generalmente i messaggi di un formato imprevisto come BytesMessage o TextMessage. Talvolta questa operazione non riesce se il messaggio è formattato male.

La visualizzazione di altri codici in questi campi è determinata da un tentativo fallito di riaccodare il messaggio a una coda di backout. In questa situazione il codice descrive il motivo della mancata riuscita del riaccodamento. Per diagnosticare la causa di questi errori, fare riferimento a *MQSeries Application Programming Reference*.

Se il messaggio di prospetto non può essere inserito in ReplyToQ, verrà inserito nella coda delle lettere non consegnate. In questa situazione il campo del feedback di MQMD viene riempito nel modo illustrato prima. Il campo del motivo in MQDLH spiega la ragione per cui non è stato possibile inserire il messaggio di prospetto in ReplyToQ.

## Codice di esempio del server applicazioni

La Figura 6 a pagina 241 riepiloga i principi delle funzionalità ServerSessionPool e ServerSession.

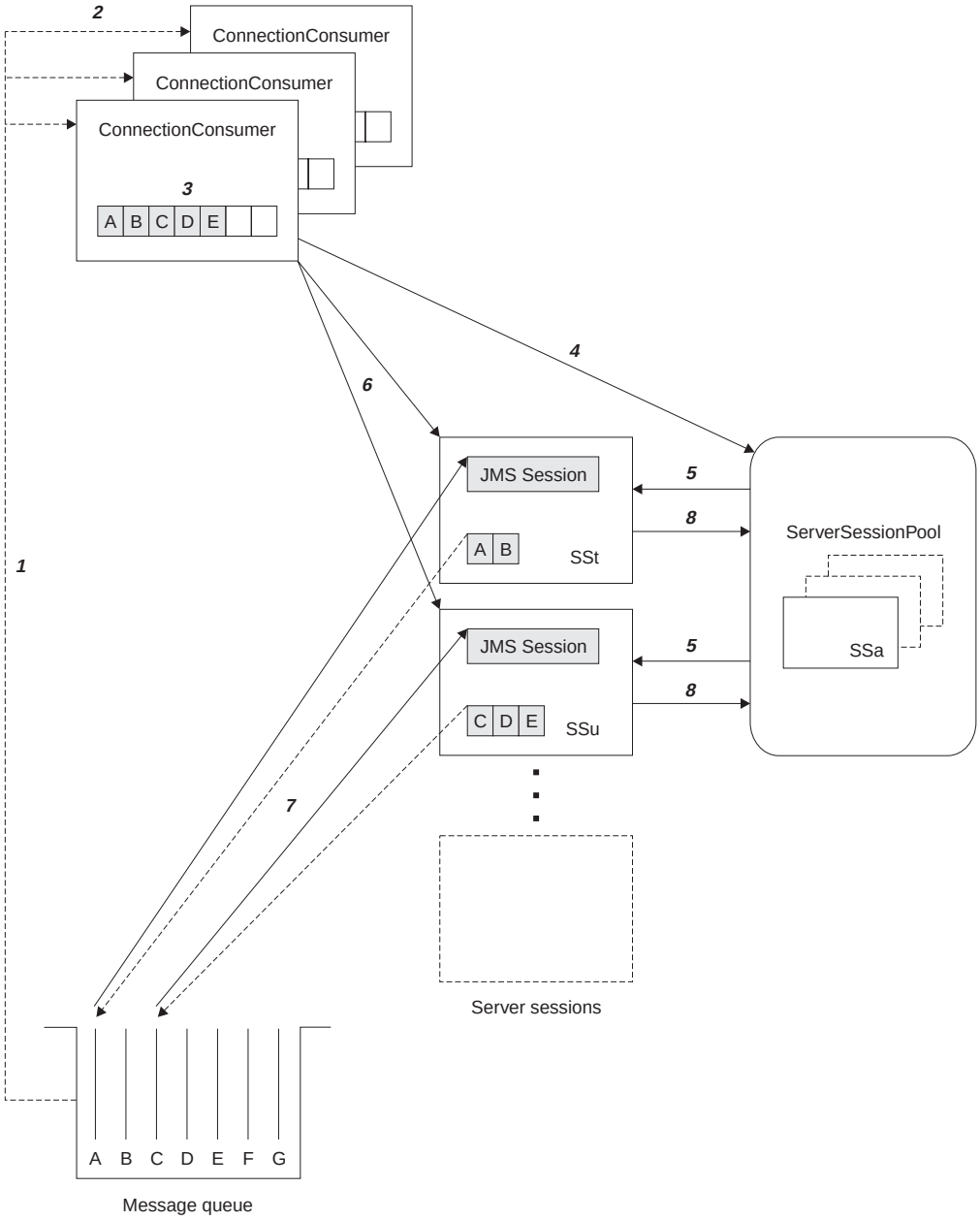


*Figura 6. Funzionalità di ServerSessionPool e ServerSession*

1. Le classi ConnectionConsumer ottengono i riferimenti dei messaggi dalla coda.
2. Ogni ConnectionConsumer seleziona specifici riferimenti dei messaggi.
3. Il buffer di ConnectionConsumer conserva i riferimenti dei messaggi selezionati.
4. ConnectionConsumer richiede una o più ServerSession da ServerSessionPool.
5. Le ServerSession vengono assegnate da ServerSessionPool.
6. ConnectionConsumer assegna i riferimenti dei messaggi alle ServerSession e avvia l'esecuzione dei thread ServerSession.
7. Ogni ServerSession richiama i relativi messaggi a cui fa riferimento dalla coda. Li passa al metodo `onMessage` dalla MessageListener associata alla sessione JMS.
8. Al termine dell'elaborazione, ServerSession viene restituita al pool.

### Codice di esempio del server applicazioni

Generalmente il server applicazioni fornisce la funzionalità di ServerSessionPool e ServerSession. Tuttavia MQ JMS viene fornito con una semplice implementazione di queste interfacce, con l'origine del programma. Questi esempi si trovano nella directory indicata di seguito, dove <dir_install> è la directory di installazione per MQ JMS:

```
<dir_install>/samples/jms/asf
```

Questi esempi consentono di utilizzare ASF MQ JMS in un ambiente standalone rendendo quindi non necessario un server applicazioni apposito. Inoltre forniscono esempi su come implementare queste interfacce e utilizzare al meglio ASF MQ JMS. Questi esempi si rivelano utili sia per gli utenti di MQ JMS che per i fornitori di altri server applicazioni.

## MyServerSession.java

Questa classe implementa l'interfaccia `javax.jms.ServerSession`. La sua funzione di base consiste nell'associare un thread con una sessione JMS. Le istanze di questa classe sono raccolte da un ServerSessionPool (vedere "MyServerSessionPool.java"). Come ServerSession, deve implementare i due metodi seguenti:

- `getSession()`, che restituisce la sessione JMS associata a questo ServerSession
- `start()`, che avvia questo thread di ServerSession e determina il richiamo del metodo `run()` della sessione JMS

MyServerSession implementa inoltre l'interfaccia `Runnable`. Pertanto la creazione del thread di ServerSession può essere basata su questa classe e non necessita di una classe separata.

La classe utilizza un meccanismo `wait()-notify()` basato sui valori di due flag booleani, ready e quit. Questo meccanismo significa che ServerSession crea e avvia il suo thread associato durante la sua costruzione. Tuttavia non esegue automaticamente il corpo del metodo `run()`. Il corpo del metodo `run()` viene eseguito solo quando il flag ready è impostato su true dal metodo `start()`. ASF richiama il metodo `start()` quando è necessario per la consegna dei messaggi alla sessione JMS associata.

Per la consegna, viene richiamato il metodo `run()` della sessione JMS. ASF MQ JMS avrà già caricato il metodo `run()` con i messaggi.

Al termine della consegna il flag ready viene reimpostato su false, e il ServerSessionPool di appartenenza viene notificato dell'avvenuta consegna. ServerSession resta quindi in uno stato di attesa fino a quando il metodo `start()` viene nuovamente richiamato oppure viene richiamato il metodo `close()` e si termina il thread di ServerSession.

## MyServerSessionPool.java

Questa classe implementa l'interfaccia `javax.jms.ServerSessionPool` ed ha la funzione di creare e controllare l'accesso a un pool di ServerSession.

In questa semplice implementazione, il pool è composto da una gamma statica di oggetti ServerSession creati durante la costruzione del pool. I quattro parametri che seguono sono passati al costruttore:

- `javax.jms.Connection` *`connessione`*

  La connessione utilizzata per creare JMS Session.
- `int` *`capacità`*

La dimensione della gamma di oggetti MyServerSession.

- `int ackMode`

  La modalità di riconoscimento richiesta di JMS Session.

- `MessageListenerFactory mlf`

  Il MesssageListenerFactory che crea il listener dei messaggi fornito a JMS Session. Vedere “MessageListenerFactory.java”.

Il costruttore del pool utilizza questi parametri per creare una gamma di oggetti MyServerSession. La connessione fornita viene utilizzata per creare JMS Session della modalità di riconoscimento e per correggere il dominio (QueueSession per point-to-point e TopicSession per pubblish/subscribe). Gli oggetti Session sono forniti con un listener dei messaggi. Infine vengono creati gli oggetti ServerSession in base a JMS Session.

Questa implementazione di esempio è un modello statico. Pertanto tutti i ServerSession del pool vengono creati al momento della creazione del pool, senza che quest'ultimo in seguito possa ridursi o ingrandirsi. Questo approccio viene utilizzato per semplicità. E' comunque possibile che un ServerSessionPool utilizzi un algoritmo sofisticato per creare ServerSession dinamicamente, quando è necessario.

MyServerSessionPool mantiene un record dei ServerSession attualmente in uso mantenendo una gamma di valori booleani chiamati inUse. Tali valori booleani sono tutti inizializzati su false. Quando il metodo `getServerSession` viene richiamato e richiede un ServerSession dal pool, nella matrice inUse viene effettuata una ricerca del primo valore false. Appena ne viene trovato uno, il valore booleano viene impostato su true, con la restituzione del primo ServerSession corrispondente. Se nella matrice inUse non ci sono valori false, il metodo `getServerSession` deve attendere (`wait()`) fino all'avvenuta notifica.

La notifica avviene in una delle seguenti circostanze:

- Viene richiamato il metodo `close()` del pool, che indica che il pool dovrebbe essere chiuso.
- Un ServerSession al momento in uso completa il suo carico di lavoro e richiama il metodo `serverSessionFinished`. Il metodo `serverSessionFinished` restituisce il ServerSession al pool e imposta il flag inUse corrispondente su false. ServerSession può essere quindi riutilizzato.

## MessageListenerFactory.java

In questo esempio, un oggetto predefinito del listener del messaggio viene associato a ogni istanza di ServerSessionPool. La classe MessageListenerFactory rappresenta un'interfaccia molto semplice che viene utilizzata per ottenere un'istanza di una classe che implementa l'interfaccia `javax.jms.MessageListener`. La classe contiene un solo metodo:

```
javax.jms.MessageListener createMessageListener();
```

Un'implementazione di questa interfaccia viene fornita al momento della creazione di ServerSessionPool. Questo oggetto viene creato per la creazione di listener dei messaggi per le singole JMS Session che eseguono il backup di ServerSession nel pool. Questa architettura sta a significare che ogni implementazione separata dell'interfaccia `MessageListenerFactory` deve avere i propri oggetti ServerSessionPool.

### Codice di esempio del server applicazioni

MQ JMS include un'implementazione di MessageListenerFactory di esempio, discussa nella sezione "CountingMessageListenerFactory.java" a pagina 245.

## Esempi di utilizzo di ASF

Esiste una serie di classi, con la propria origine, nella directory `<dir_install>/samples/jms/asf` (dove <dir_install> è la directory di installazione per MQ JMS). Queste classi utilizzano le funzionalità del server applicazioni MQ JMS descritte nella sezione "Classi e funzioni ASF" a pagina 233, all'interno dell'ambiente del server applicazioni standalone di esempio descritto nella sezione "Codice di esempio del server applicazioni" a pagina 240.

Vengono forniti tre esempi di utilizzo di ASF dalla prospettiva di un'applicazione client:

- Un esempio point-to-point semplice utilizza:
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- Un esempio point-to-point più complesso utilizza:
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- Un esempio publish/subscribe semplice utilizza:
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- Un esempio publish/subscribe più complesso utilizza:
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java

Nelle sezioni che seguono verrà descritta ciascuna classe.

### Load1.java

Questa classe è una semplice applicazione JMS generica che carica una determinata coda con un numero di messaggi, quindi termina. Può richiamare gli oggetti amministrati richiesti da uno spazio dei nomi JNDI oppure crearli esplicitamente, utilizzando le classi MQ JMS che implementano questa interfaccia. Gli oggetti amministrati richiesti sono QueueConnectionFactory e Queue. E' possibile utilizzare le opzioni della riga comandi per impostare il numero di messaggi con cui caricare la coda e il tempo di inattività tra i singoli inserimenti dei messaggi.

Questa applicazione dispone di due versioni della sintassi della riga comandi.

Per un utilizzo con JNDI la sintassi è:

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

Per un utilizzo senza JNDI la sintassi è:

```
java Load1 -nojndi [-qm qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

La Tabella 29 descrive i parametri e ne fornisce le impostazioni predefinite.

*Tabella 29. Parametri e impostazioni predefinite Load1*

| Parametro | Significato | Impostazione predefinita |
|---|---|---|
| jndiICF | Classe del valore predefinito del contesto iniziale utilizzata per JNDI | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | URL del provider utilizzato per JNDI | ldap://localhost/o=ibm,c=us |
| qcfLookup | Chiave di ricerca JNDI utilizzata per QueueConnectionFactory | cn=qcf |
| qLookup | Chiave di ricerca JNDI utilizzata per Queue | cn=q |
| qMgrName | Nome del gestore code a cui connettersi | "" (utilizzare il gestore code predefinito) |
| qName | Nome della coda da caricare | SYSTEM.DEFAULT.LOCAL.QUEUE |
| sleepTime | Tempo (in millisecondi) di pausa tra gli inserimenti dei messaggi | 0 (nessuna pausa) |
| numMsgs | Numero di messaggi da inserire | 1000 |

In caso di errori, viene visualizzato un messaggio di errore e l'applicazione viene terminata.

E' possibile utilizzare questa applicazione per simulare il caricamento dei messaggi su una coda MQSeries. A sua volta il caricamento del messaggio può attivare le applicazioni abilitate all'ASF descritte nelle sezioni che seguono. I messaggi da inserire nella coda sono semplici oggetti JMS TextMessage che non contengono proprietà dei messaggi definite dall'utente che potrebbero essere utili per fare uso di diversi listener dei messaggi. Viene anche fornito il codice sorgente in modo da poter modificare questa applicazione di carico se necessario.

## CountingMessageListenerFactory.java

Questo file contiene definizioni per due classi:

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener è un'implementazione molto semplice dell'interfaccia `javax.jms.MessageListener`. Mantiene un record del numero di volte in cui è stato richiamato il metodo `onMessage`, ma non fa nulla con i messaggi trasmessi.

CountingMessageListenerFactory è la classe predefinita per CountingMessageListener. rappresenta un'implementazione dell'interfaccia `MessageListenerFactory` descritta in "MessageListenerFactory.java" a pagina 243. Mantiene un record di tutti i listener dei messaggi che produce. Include anche un metodo, `printStats()`, che visualizza le statistiche dell'utilizzo relative a ciascuno di questi listener.

## ASFClient1.java

Questa applicazione funziona come client dell'ASF MQ JMS. Imposta una sola ConnectionConsumer per consumare i messaggi in una sola coda MQSeries. Visualizza statistiche sulla velocità di esecuzione per ogni listener dei messaggi utilizzato e termina dopo un minuto.

Può richiamare gli oggetti amministrati richiesti da uno spazio dei nomi JNDI oppure crearli esplicitamente, utilizzando le classi MQ JMS che implementano questa interfaccia. Gli oggetti amministrati richiesti sono QueueConnectionFactory e Queue.

Questa applicazione dispone di due versioni della sintassi della riga comandi:

Per un utilizzo con JNDI la sintassi è:

```
java ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

Per un utilizzo senza JNDI la sintassi è:

```
java ASFClient1 -nojndi [-qm qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

La Tabella 30 descrive i parametri e ne fornisce le impostazioni predefinite.

*Tabella 30. Parametri e impostazioni predefinite ASFClient1*

| Parametro | Significato | Impostazione predefinita |
|---|---|---|
| jndiICF | Classe del valore predefinito del contesto iniziale utilizzata per JNDI | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | URL del provider utilizzato per JNDI | ldap://localhost/o=ibm,c=us |
| qcfLookup | Chiave di ricerca JNDI utilizzata per QueueConnectionFactory | cn=qcf |
| qLookup | Chiave di ricerca JNDI utilizzata per Queue | cn=q |
| qMgrName | Nome del gestore code a cui connettersi | "" (utilizzare il gestore code predefinito) |
| qName | Nome della coda da cui consumare | SYSTEM.DEFAULT.LOCAL.QUEUE |
| poolSize | Il numero di ServerSession creati nel ServerSessionPool utilizzato | 5 |
| batchSize | Il numero massimo di messaggi che è possibile assegnare a un ServerSession alla volta | 10 |

L'applicazione ottiene un QueueConnection dal QueueConnectionFactory.

Un ServerSessionPool, nella forma di un MyServerSessionPool, viene costruito utilizzando:

- il QueueConnection creato precedentemente
- il poolSize richiesto
- una modalità di riconoscimento, AUTO_ACKNOWLEDGE

- un'istanza di un CountingMessageListenerFactory, come descritto in "CountingMessageListenerFactory.java" a pagina 245

Viene quindi richiamato il metodo `createConnectionConsumer` della connessione, che passa:

- l'oggetto Queue ottenuto in precedenza
- un selettore dei messaggi null (che indica che dovrebbero essere accettati tutti i messaggi)
- il ServerSessionPool appena creato
- il batchSize richiesto

L'utilizzo dei messaggi viene quindi avviato tramite il richiamo del metodo `start()` della connessione.

L'applicazione client visualizza ogni 10 secondi statistiche sulla velocità di esecuzione per ogni listener dei messaggi utilizzato. Dopo un minuto la connessione viene chiusa, il pool di sessioni del server viene interrotto e l'applicazione termina.

## Load2.java

Questa classe è un'applicazione JMS che carica una determinata coda con un numero di messaggi, quindi termina, in modo simile a Load1.java. La sintassi della riga comandi è anche simile a quella relativa a Load1.java. Sostituire Load2 al posto di Load1 nella sintassi. Per ulteriori dettagli, consultare la sezione "Load1.java" a pagina 244.

La differenza sta nel fatto che ciascun messaggio contiene un valore richiamato dalla proprietà dell'utente che prende un valore intero selezionato in modo causale compreso tra 0 e 100. Questa proprietà significa che è possibile applicare ai messaggi dei selettori. Di conseguenza i messaggi possono essere condivisi tra i due consumer creati nell'applicazione client descritti in "ASFClient2.java".

## LoggingMessageListenerFactory.java

Questo file contiene definizioni per due classi:

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener è un'implementazione dell'interfaccia `javax.jms.MessageListener`. Prende i messaggi trasmessi e scrive una voce nel file di registrazione. Il file di registrazione predefinito è `./ASFClient2.log`. E' possibile esaminare questo file e verificare i messaggi inviati al consumer della connessione che sta utilizzando il listener del messaggio.

LoggingMessageListenerFactory è la classe predefinita per LoggingMessageListener e rappresenta un'implementazione dell'interfaccia `MessageListenerFactory` descritta in "MessageListenerFactory.java" a pagina 243.

## ASFClient2.java

ASFClient2.java è un'applicazione client leggermente più complicata di ASFClient1.java. Crea due ConnectionConsumer che popolano la stessa coda ma applicano diversi selettori dei messaggi. L'applicazione utilizza un CountingMessageListenerFactory per un consumer e un LoggingMessageListenerFactory per l'altro. L'utilizzo di due diverse impostazioni predefinite del listener dei messaggi significa che ciascun consumer deve disporre del proprio pool di sessioni del server.

L'applicazione visualizza statistiche relative a una ConnectionConsumer e scrive statistiche relative all'altra ConnectionConsumer in un file di registrazione.

La sintassi della riga comandi è simile a quella relativa a "ASFClient1.java" a pagina 246 (sostituire ASFClient2 per ASFClient1 nella sintassi). Ciascuno dei pool di sessioni del server contiene il numero di ServerSessions impostati dal parametro poolSize.

Dovrebbe esserci una distribuzione irregolare dei messaggi. I messaggi caricati nella coda di origine da Load2 contengono una proprietà dell'utente in cui il valore dovrebbe essere compreso tra 0 e 100, distribuiti in modo causale e irregolare. Il `value>75` del selettore del messaggio viene applicato a highConnectionConsumer e `value≤75` viene applicato a normalConnectionConsumer. I messaggi di highConnectionConsumer (approssimativamente il 25% del carico totale) vengono inviati a un LoggingMessageListener. I messaggi di normalConnectionConsumer (approssimativamente il 75% del carico totale) vengono inviati a un CountingMessageListener.

Durante l'esecuzione dell'applicazione client, le statistiche relative a normalConnectionConsumer, con i CountingMessageListenerFactory associati, vengono stampate sullo schermo ogni 10 secondi. Le statistiche relative al highConnectionConsumer, con i LoggingMessageListenerFactory associati, sono scritte nel file di registrazione.

Esaminando lo schermo e il file di registrazione, sarà possibile vedere la reale distribuzione dei messaggi. Aggiungere i totali di ciascun CountingMessageListener. Se l'applicazione client non termina prima dell'utilizzo di tutti i messaggi, questo ammonta a circa il 75% del carico. Il numero di voci del file di registrazione dovrebbe rappresentare il resto del carico. (Se l'applicazione client termina prima dell'utilizzo di tutti i messaggi, è possibile aumentare il timeout dell'applicazione.)

## TopicLoad.java

Questa classe è un'applicazione JMS che rappresenta una versione publish/subscribe del loader della coda Load2 descritto in "Load2.java" a pagina 248. Essa pubblica il numero richiesto di messaggi che ricadono in un determinato argomento, quindi termina. Ciascun messaggio contiene un valore richiamato dalla proprietà dell'utente che prende un valore intero selezionato in modo causale compreso tra 0 e 100.

Per utilizzare questa applicazione, assicurarsi che il broker sia in esecuzione e che la configurazione richiesta sia completa. Per ulteriori dettagli, consultare la sezione "Configurazione ulteriore per la modalità Publish/Subscribe" a pagina 24.

Questa applicazione dispone di due versioni della sintassi della riga comandi.

Per un utilizzo con JNDI la sintassi è:

```
java TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

Per un utilizzo senza JNDI la sintassi è:

```
java TopicLoad -nojndi [-qm qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

La Tabella 31 descrive i parametri e ne fornisce le impostazioni predefinite.

*Tabella 31. Parametri e impostazioni predefinite TopicLoad*

| Parametro | Significato | Impostazione predefinita |
|---|---|---|
| jndiICF | Classe del valore predefinito del contesto iniziale utilizzata per JNDI | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | URL del provider utilizzato per JNDI | ldap://localhost/o=ibm,c=us |
| tcfLookup | Chiave di ricerca JNDI utilizzata per TopicConnectionFactory | cn=tcf |
| tLookup | Chiave di ricerca JNDI utilizzata per Topic | cn=t |
| qMgrName | Nome del gestore code a cui connettersi e del gestore code del broker su cui pubblicare i messaggi | "" (utilizzare il gestore code predefinito) |
| tName | Nome dell'argomento in cui pubblicare | MQJMS/ASF/TopicLoad |
| sleepTime | Tempo (in millisecondi) di pausa tra gli inserimenti dei messaggi | 0 (nessuna pausa) |
| numMsgs | Numero di messaggi da inserire | 200 |

In caso di errori, viene visualizzato un messaggio di errore e l'applicazione viene terminata.

## ASFClient3.java

ASFClient3.java è un'applicazione client che rappresenta una versione publish/subscribe di "ASFClient1.java" a pagina 246. Imposta una sola ConnectionConsumer per consumare i messaggi pubblicati su un solo argomento. Visualizza statistiche sulla velocità di esecuzione per ogni listener dei messaggi utilizzato e termina dopo un minuto.

Questa applicazione dispone di due versioni della sintassi della riga comandi.

Per un utilizzo con JNDI la sintassi è:

```
java ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize dimensionepool] [-batchsize dimensionebatch]
```

Per un utilizzo senza JNDI la sintassi è:

```
java ASFClient3 -nojndi [-qm qMgrName] [-t tName]
                        [-poolsize dimensionepool] [-batchsize dimensionebatch]
```

La Tabella 32 descrive i parametri e ne fornisce le impostazioni predefinite.

*Tabella 32. Parametri e impostazioni predefinite ASFClient3*

| Parametro | Significato | Impostazione predefinita |
|---|---|---|
| jndiICF | Classe del valore predefinito del contesto iniziale utilizzata per JNDI | com.sun.jndi.ldap.LdapCtxFactory |

*Tabella 32. Parametri e impostazioni predefinite ASFClient3 (Continua)*

| Parametro | Significato | Impostazione predefinita |
|---|---|---|
| jndiURL | URL del provider utilizzato per JNDI | ldap://localhost/o=ibm,c=us |
| tcfLookup | Chiave di ricerca JNDI utilizzata per TopicConnectionFactory | cn=tcf |
| tLookup | Chiave di ricerca JNDI utilizzata per Topic | cn=t |
| qMgrName | Nome del gestore code a cui connettersi e del gestore code del broker su cui pubblicare i messaggi | "" (utilizzare il gestore code predefinito) |
| tName | Nome dell'argomento da cui consumare | MQJMS/ASF/TopicLoad |
| poolSize | Il numero di ServerSession creati nel ServerSessionPool utilizzato | 5 |
| batchSize | Il numero massimo di messaggi che è possibile assegnare a un ServerSession alla volta | 10 |

Così come ASFClient1, l'applicazione client visualizza ogni 10 secondi statistiche sulla velocità di esecuzione per ogni listener dei messaggi utilizzato. Dopo un minuto la connessione viene chiusa, il pool di sessioni del server viene interrotto e l'applicazione termina.

## ASFClient4.java

ASFClient4.java è un'applicazione client publish/subscribe più complessa. Crea tre ConnectionConsumer che riguardano lo stesso argomento, ma ciascuna di esse si applica a selettori dei messaggi diversi.

I primi due consumer utilizzano selettori di messaggi di tipo 'high' e 'normal', allo stesso modo dell'applicazione "ASFClient2.java" a pagina 248. Il terzo consumer non utilizza alcun selettore dei messaggi. L'applicazione utilizza due CountingMessageListenerFactory per i due consumer basati sui selettori e un LoggingMessageListenerFactory per il terzo consumer. Dal momento che l'applicazione utilizza diverse impostazioni predefinite del listener dei messaggi, ciascun consumer deve disporre del proprio pool di sessioni server.

L'applicazione visualizza statistiche relative ai due consumer basati sul selettore sullo schermo. Scrive inoltre statistiche relative al terzo ConnectionConsumer in un file di registrazione.

La sintassi della riga comandi è simile a quella relativa "ASFClient3.java" a pagina 250 (sostituire ASFClient4 al posto di ASFClient3 nella sintassi). Ciascuno dei tre pool di sessioni server contiene il numero di ServerSessions impostati dal parametro poolSize.

Durante l'esecuzione dell'applicazione client, le statistiche relative a normalConnectionConsumer e highConnectionConsumer, con i CountingMessageListenerFactory associati, vengono stampate sullo schermo ogni 10 secondi. Le statistiche relative alla terza ConnectionConsumer, con i LoggingMessageListenerFactory associati, vengono scritte nel file di registrazione.

## Esempi di utilizzo di ASF

Esaminando lo schermo e il file di registrazione, sarà possibile vedere la reale distribuzione dei messaggi. Aggiungere i totali di ciascun CountingMessageListener ed esaminare il numero di voci del file di registrazione.

La distribuzione dei messaggi dovrebbe essere diversa dalla distribuzione ottenuta da una versione point-to-point della stessa applicazione (ASFClient2.java). Ciò accade in quanto nel dominio publish/subscribe ciascun consumer di un argomento ottiene la propria copia di ogni messaggio pubblicato su quell'argomento. In questa applicazione per un determinato carico di argomenti, i consumer di tipo 'high' e 'normal' riceveranno rispettivamente circa il 25% e il 75% del carico. Il terzo consumer riceverà ancora il 100% del carico. Pertanto, il numero totale di messaggi ricevuti è superiore al 100% del carico inizialmente pubblicato sull'argomento.

# Capitolo 14. Interfacce e classi JMS

MQSeries classi per Java Message Service è composta da una serie di classi ed interfacce Java basate sul pacchetto Sun javax.jms di interfacce e classi. I client dovrebbero essere scritti utilizzando le interfacce e le classi indicate di seguito, descritte in dettaglio più avanti. I nomi degli oggetti MQSeries che implementano le interfacce e le classi Sun sono preceduti da un prefisso "MQ" (a meno che non venga specificato diversamente nella specifica descrizione dell'oggetto). Le descrizioni infatti comprendono informazioni dettagliate sulle eventuali deviazioni degli oggetti MQSeries dalle definizioni JMS standard. Tali deviazioni sono contrassegnate con un '*'.

## Clasi ed interfacce Sun Java Message Service

Nelle tabelle seguenti vengono elencati gli oggetti JMS contenuti nel pacchetto **javax.jms**. Le interfacce contrassegnate da '*' sono implementate dalle applicazioni. Le interfacce contrassegnate con '**' sono implementate dai server delle applicazioni.

*Tabella 33. Riepilogo delle interfacce*

| Interfaccia | Descrizione |
|---|---|
| **BytesMessage** | Una BytesMessage viene utilizzata per inviare un messaggio che contiene un flusso di byte non interpretati. |
| **Connection** | Una Connection JMS è una connessione attiva del client al proprio provider JMS. |
| **ConnectionConsumer** | Per i server applicazioni, le Connection forniscono una funzione speciale per la creazione di una ConnectionConsumer. |
| **ConnectionFactory** | Una ConnectionFactory incapsula un gruppo di parametri di configurazione delle connessione definiti da un amministratore. |
| **ConnectionMetaData** | ConnectionMetaData fornisce le informazioni che descrivono la Connection. |
| **DeliveryMode** | Modalità di consegna supportate da JMS. |
| **Destination** | L'interfaccia principale per Queue e Topic. |
| **ExceptionListener*** | Un listener delle eccezioni viene utilizzato per ricevere le eccezioni prodotte dai thread di consegna asincrona delle Connection. |
| **MapMessage** | Una MapMessage viene utilizzata per inviare una serie di coppie nome-valore in cui i nomi sono String e i valori sono tipi di valori primitivi Java. |
| **Message** | L'interfaccia Message è l'interfaccia principale di tutti i messaggi JMS. |
| **MessageConsumer** | L'interfaccia principale di tutti i consumer dei messaggi. |
| **MessageListener*** | Una MessageListener viene utilizzata per ricevere in modo asincrono i messaggi consegnati. |
| **MessageProducer** | Un client utilizza una MessageProducer per inviare i messaggi a una destinazione. |

*Tabella 33. Riepilogo delle interfacce (Continua)*

| Interfaccia | Descrizione |
|---|---|
| **ObjectMessage** | Una ObjectMessage viene utilizzata per inviare un messaggio contenente un oggetto Java. |
| **Queue** | Un'interfaccia Queue incapsula un nome di coda specifico del provider. |
| **QueueBrowser** | Un client utilizza una QueueBrowser per prendere visione dei messaggi su una coda senza rimuoverli. |
| **QueueConnection** | Una QueueConnection è una connessione attiva verso un provider JMS point-to-point. |
| **QueueConnectionFactory** | Un client utilizza una QueueConnectionFactory per creare QueueConnections con un provider JMS point-to-point. |
| **QueueReceiver** | Un client utilizza una QueueReceiver per la ricezione dei messaggi consegnati su una coda. |
| **QueueSender** | Un client utilizza una QueueSender per inviare messaggi a una coda. |
| **QueueSession** | Una QueueSession fornisce i metodi per creare QueueReceiver, QueueSender, QueueBrowser e TemporaryQueue. |
| **ServerSession** ** | ServerSession è un oggetto implementato da un server applicazioni. |
| **ServerSessionPool** ** | ServerSessionPool è un oggetto implementato da un server applicazioni per fornire un pool di ServerSession per l'elaborazione dei messaggi di una ConnectionConsumer. |
| **Session** | Una JMS Session è un contesto a singolo thread per la produzione e il consumo di messaggi. |
| **StreamMessage** | Una StreamMessage viene utilizzata per inviare un flusso di valori primitivi Java. |
| **TemporaryQueue** | Un TemporaryQueue è un oggetto Queue univoco creato per la durata di una QueueConnection. |
| **TemporaryTopic** | Un TemporaryTopic è un oggetto Topic univoco creato per la durata di una TopicConnection. |
| **TextMessage** | Una TextMessage viene utilizzata per inviare un messaggio contenente una `java.lang.String`. |
| **Topic** | Un oggetto Topic incapsula un nome di argomento specifico del provider. |
| **TopicConnection** | Una TopicConnection è una connessione attiva a un provider JMS Pub/Sub. |
| **TopicConnectionFactory** | Un client utilizza una TopicConnectionFactory per creare TopicConnection con un provider JMS Publish/Subscribe. |
| **TopicPublisher** | Un client utilizza un TopicPublisher per pubblicare dei messaggi su un argomento. |
| **TopicSession** | Una TopicSession fornisce metodi per la creazione di TopicPublisher, TopicSubscriber e TemporaryTopic. |
| **TopicSubscriber** | Un client utilizza una TopicSubscriber per la ricezione dei messaggi pubblicati in un argomento. |

*Tabella 33. Riepilogo delle interfacce (Continua)*

| Interfaccia | Descrizione |
|---|---|
| **XAConnection** | XAConnection estende la capacità di Connection fornendo una XASession. |
| **XAConnectionFactory** | Alcuni server applicazioni forniscono il supporto per integrare l'utilizzo di risorse con capacità JTS ( Java) in una transazione distribuita. |
| **XAQueueConnection** | XAQueueConnection fornisce le stesse opzioni creazione di QueueConnection. |
| **XAQueueConnectionFactory** | Una XAQueueConnectionFactory fornisce le stesse opzioni di creazione di una QueueConnectionFactory. |
| **XAQueueSession** | Una XAQueueSession fornisce una regolare QueueSession che può essere utilizzata per creare QueueReceiver, QueueSender e QueueBrowser. |
| **XASession** | XASession estende la capacità di Session aggiungendo l'accesso al supporto del provider JMS per JTA (Java Transaction API). |
| **XATopicConnection** | Una XATopicConnection fornisce le stesse opzioni di creazione di una TopicConnection. |
| **XATopicConnectionFactory** | Un'interfaccia XATopicConnectionFactory fornisce le stesse opzioni di creazione di un'interfaccia TopicConnectionFactory. |
| **XATopicSession** | Una XATopicSession fornisce una regolare TopicSession, che può essere utilizzata per creare TopicSubscriber e TopicPublisher. |

*Tabella 34. Riepilogo delle classi*

| Classe | Descrizione |
|---|---|
| **QueueRequestor** | JMS fornisce la classe QueueRequestor per semplificare l'esecuzione di richieste di servizio. |
| **TopicRequestor** | JMS fornisce la classe TopicRequestor per semplificare l'esecuzione di richieste di servizio. |

# Classi JMS MQSeries

Due pacchetti contengono MQSeries classi per Java Message Service che implementa le interfacce Sun. La Tabella 35 elenca il pacchetto **com.ibm.mq.jms** e la Tabella 36 a pagina 257 elenca il pacchetto **com.ibm.jms**.

*Tabella 35. Riepilogo delle classi del pacchetto 'com.ibm.mq.jms'*

| Classe | Implementa |
|---|---|
| MQConnection | Connection |
| MQConnectionConsumer | ConnectionConsumer |
| MQConnectionFactory | ConnectionFactory |
| MQConnectionMetaData | ConnectionMetaData |
| MQDestination | Destination |
| MQMessageConsumer | MessageConsumer |
| MQMessageProducer | MessageProducer |
| MQQueue | Queue |
| MQQueueBrowser | QueueBrowser |
| MQQueueConnection | QueueConnection |
| MQQueueConnectionFactory | QueueConnectionFactory |
| MQQueueEnumeration | java.util.Enumeration da QueueBrowser |
| MQQueueReceiver | QueueReceiver |
| MQQueueSender | QueueSender |
| MQQueueSession | QueueSession |
| MQSession | Session |
| MQTemporaryQueue | TemporaryQueue |
| MQTemporaryTopic | TemporaryTopic |
| MQTopic | Topic |
| MQTopicConnection | TopicConnection |
| MQTopicConnectionFactory | TopicConnectionFactory |
| MQTopicPublisher | TopicPublisher |
| MQTopicSession | TopicSession |
| MQTopicSubscriber | TopicSubscriber |
| MQXAConnection[1] | XAConnection |
| MQXAConnectionFactory[1] | XAConnectionFactory |
| MQXAQueueConnection[1] | XAQueueConnection |
| MQXAQueueConnectionFactory[1] | XAQueueConnectionFactory |
| MQXAQueueSession[1] | XAQueueSession |
| MQXASession[1] | XASession |
| MQXATopicConnection[1] | XATopicConnection |
| MQXATopicConnectionFactory[1] | XATopicConnectionFactory |
| MQXATopicSession[1] | XATopicSession |

**Note:**

1. Queste classi implementando la funzione XA non vengono supportate da MQ Java per iSeries & AS/400

*Tabella 36. Riepilogo delle classi del pacchetto 'com.ibm.jms'*

| **Classe** | **Implementa** |
|---|---|
| JMSBytesMessage | BytesMessage |
| JMSMapMessage | MapMessage |
| JMSMessage | Message |
| JMSObjectMessage | ObjectMessage |
| JMSStreamMessage | StreamMessage |
| JMSTextMessage | TextMessage |

Un'implementazione di esempio delle seguenti interfacce JMS viene fornita in questo rilascio di MQSeries classi per Java Message Service.

- ServerSession
- ServerSessionPool

Consultare il "Codice di esempio del server applicazioni" a pagina 240.

# BytesMessage

interfaccia pubblica **BytesMessage**
estende **Message**

Classe MQSeries: **JMSBytesMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

Una BytesMessage viene utilizzata per inviare un messaggio che contiene un flusso di byte non interpretati. Essa eredita **Message** ed aggiunge un corpo di messaggio di byte. Il destinatario del messaggio fornisce l'interpretazione dei byte.

**Nota:** Questo tipo di messaggio serve per la codifica client dei formati di messaggio esistenti. Se possibile, utilizzare invece uno degli altri tipi di messaggi a definizione automatica.

Vedere anche: **MapMessage**, **Message**, **ObjectMessage**, **StreamMessage** e **TextMessage**.

## Metodi

**readBoolean**

```
public boolean readBoolean() throws JMSException
```

Leggere un valore `boolean` dal messaggio di byte.

**Restituisce:**
il valore `boolean` letto.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se è la fine dei byte del messaggio.

**readByte**

```
public byte readByte() throws JMSException
```

Leggere un valore a otto bit dal messaggio di byte.

**Restituisce:**
il successivo byte dal messaggio di byte come un `byte` a otto bit con segno.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readUnsignedByte**

```
public int readUnsignedByte() throws JMSException
```

Leggere un numero a otto bit senza segno dal messaggio di byte.

**Restituisce:**
il successivo byte dal messaggio di byte, interpretato come un numero a 8 bit senza segno.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readShort**

```
public short readShort() throws JMSException
```

Leggere un numero a 16 bit dal messaggio di byte.

**Restituisce:**
i successivi due byte dal messaggio di byte, interpretati come un numero a 16 bit con segno.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readUnsignedShort**

```
public int readUnsignedShort() throws JMSException
```

Leggere un numero a sedici bit senza segno dal messaggio di byte.

**Restituisce:**
i successivi due byte dal messaggio di byte, interpretati come un valore integer a 16 bit senza segno.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readChar**

```
public char readChar() throws JMSException
```

Leggere un valore di carattere Unicode dal messaggio di byte.

**Restituisce:**
i successivi due byte dal messaggio di byte come un carattere Unicode.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readInt**

```
public int readInt() throws JMSException
```

Leggere un valore integer a 32 bit con segno dal messaggio di byte.

**Restituisce:**

i successivi quattro byte dal messaggio di byte, interpretati come un valore `int`.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readLong**

```
public long readLong() throws JMSException
```

Leggere un valore integer a 64 bit con segno dal messaggio di byte.

**Restituisce:**

i successivi otto byte dal messaggio di byte, interpretati come un valore `long`.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readFloat**

```
public float readFloat() throws JMSException
```

Leggere un valore `float` dal messaggio di byte.

**Restituisce:**

i successivi quattro byte dal messaggio di byte, interpretati come un valore `float`.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readDouble**

```
public double readDouble() throws JMSException
```

Leggere un valore `double` dal messaggio di byte.

**Restituisce:**
i successivi otto byte dal messaggio di byte, interpretati come un valore `double`.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readUTF**

```
public java.lang.String readUTF() throws JMSException
```

Leggere in una stringa che è stata codificata utilizzando un formato UTF-8 modificato dal messaggio di byte. I primi due byte sono interpretati come un campo con una lunghezza di 2 byte.

**Restituisce:**
Una stringa Unicode dal messaggio di byte.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- MessageEOFException - se è la fine dei byte del messaggio.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readBytes**

```
public int readBytes(byte[] value) throws JMSException
```

Leggere una matrice di byte dal messaggio di byte. Se nel flusso resta una quantità sufficiente di byte, viene riempito l'intero buffer. In caso contrario, il buffer viene riempito in modo parziale.

**Parametri:**
`value` - il buffer in cui vengono letti i dati.

**Restituisce:**
il numero totale di byte letto nel buffer oppure -1 se non ci sono più dati perché è stata raggiunta la fine dei byte.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**readBytes**

```
public int readBytes(byte[] value, int length)
                                        throws JMSException
```

Leggere una porzione del messaggio di byte.

**Parametri:**

- `value` - il buffer in cui vengono letti i dati.
- `length` - il numero di byte da leggere.

**Restituisce:**

il numero totale di byte letto nel buffer oppure -1 se non ci sono più dati perché è stata raggiunta la fine dei byte.

**Produce:**

- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.
- IndexOutOfBoundsException - se `lunghezza` è un valore negativo oppure se è inferiore alla lunghezza del `valore` della matrice
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**writeBoolean**

```
public void writeBoolean(boolean value) throws JMSException
```

Scrivere un valore `boolean` nel messaggio di byte come un valore a 1 byte. Il valore `true` viene scritto come il valore `(byte)1`; il valore `false` viene scritto come il valore `(byte)0`.

**Parametri:**

`value` - il valore `boolean` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeByte**

```
public void writeByte(byte value) throws JMSException
```

Scrivere un `byte` nel messaggio di byte come un valore a 1 byte.

**Parametri:**

`value` - il valore `byte` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeShort**

```
public void writeShort(short value) throws JMSException
```

Scrivere un valore `short` nel messaggio di byte come due byte.

**Parametri:**

`value` - il valore `short` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeChar**

```
public void writeChar(char value) throws JMSException
```

Scrivere un valore `char` nel messaggio di byte come un valore a due byte, inserendo prima il byte più significativo.

**Parametri:**

`value` - il valore `char` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeInt**

```
public void writeInt(int value) throws JMSException
```

Scrivere un valore `int` nel messaggio di byte come quattro byte.

**Parametri:**

`value` - il valore `int` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeLong**

```
public void writeLong(long value) throws JMSException
```

Scrivere un valore `long` nel messaggio di byte come otto byte.

**Parametri:**

`value` - il valore `long` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeFloat**

```
public void writeFloat(float value) throws JMSException
```

Convertire l'argomento float in un valore `int` utilizzando il metodo `floatToIntBits` nella classe `Float` e scrivere quindi questo valore `int` nel messaggio di byte come una quantità di 4 byte.

**Parametri:**

`value` - il valore `float` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeDouble**

```
public void writeDouble(double value) throws JMSException
```

Convertire l'argomento double argument in un valore `long` utilizzando il metodo `doubleToLongBits` nella classe `Double` e scrivere quindi questo valore `long` nel messaggio di byte come una quantità di 8 byte.

**Parametri:**

`value` - il valore `double` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeUTF**

```
public void writeUTF(java.lang.String value)
                             throws JMSException
```

Scrivere una stringa nel messaggio di byte utilizzando la codifica UTF-8 in un modo indipendente dalla macchina. La stringa UTF-8 scritta nel buffer inizia con un campo con una lunghezza di 2 byte.

**Parametri:**

`value` - il valore `String` da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeBytes**

```
public void writeBytes(byte[] value) throws JMSException
```

Scrivere una matrice di byte nel messaggio di byte.

**Parametri:**

`value` - la matrice di byte da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeBytes**

```
public void writeBytes(byte[] value,
                       int length) throws JMSException
```

Scrivere una porzione di una matrice di byte nel messaggio di byte.

**Parametri:**

- `value` - il valore di matrice di byte da scrivere.
- `offset` - l'offset iniziale nella matrice di byte.
- `length` - il numero di byte da utilizzare.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**writeObject**

```
public void writeObject(java.lang.Object value)
                                    throws JMSException
```

Scrivere un oggetto Java nel messaggio di byte.

**Nota:** Questo metodo funziona solo per i tipi di oggetto primitivi (come ad esempio Integer, Double e Long), per le stringhe e per le matrici di byte.

**Parametri:**

`valore` - l'oggetto Java da scrivere.

**Produce:**

- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- MessageFormatException - se l'oggetto è di tipo non valido.
- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.

**reset**

```
public void reset() throws JMSException
```

Inserire il corpo del messaggio in modalità di sola lettura e riposizionare i byte dei byte all'inizio.

**Produce:**

- JMSException - se JMS non riesce a reimpostare il messaggio a causa di un errore JMS interno.
- MessageFormatException - se il messaggio è in un formato non valido.

# Connection

interfaccia pubblica **Connection**
Interfacce secondarie: **QueueConnection**, **TopicConnection**, **XAQueueConnection** e**XATopicCo**

Classe MQSeries: **MQConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

Una Connection JMS è una connessione attiva del client al proprio provider JMS.

Vedere anche: **QueueConnection**, **TopicConnection**, **XAQueueConnection** e **XATopicConnection**

## Metodi

**getClientID**

```
public java.lang.String getClientID()
                                  throws JMSException
```

Richiamare l'id client per questa connessione. L'Id client può essere preconfigurato dall'amministratore in una ConnectionFactory oppure può essere assegnato dal setClientId che esegue la chiamata.

**Restituisce:**
L'ID client unico.

**Produce:**
JMSException - se l'implementazione JMS non riesce a restituire l'ID client per questa Connection a causa di un errore interno.

**setClientID**

```
public void setClientID(java.lang.String clientID)
                                            throws JMSException
```

Impostare l'id client per questa connessione.

**Nota:** L'id client viene ignorato per le connessioni point-to-point.

**Parametri:**
clientID - l'id client unico.

**Produce:**

- JMSException - se l'implementazione JMS non riesce ad impostare l'ID client per questa Connection a causa di un errore interno.
- InvalidClientIDException - se il client JMS specifica un ID client non valido o duplicato.
- IllegalStateException - se si sta tentando di impostare l'id client della connessione quando non appropriato oppure se esso è stato configurato da un amministratore.

**getMetaData**

```
public ConnectionMetaData getMetaData() throws JMSException
```

Richiamare i metadati per questa connessione.

**Restituisce:**
I metadati della connessione.

**Produce:**
JMSException - condizione di eccezione generica se l'implementazione JMS non riesce a richiamare i metadati per questa connessione.

**Vedere anche:**
**ConnectionMetaData**

**getExceptionListener**

```
public ExceptionListener getExceptionListener()
                                              throws JMSException
```

Richiamare l'ExceptionListener per questa connessione.

**Restituisce:**
L'ExceptionListener per questa connessione

**Produce:**
JMSException - condizione di eccezione generica se l'implementazione JMS non riesce a richiamare l'Exception Listener per questa connessione.

**setExceptionListener**

```
public void setExceptionListener(ExceptionListener listener)
                                                        throws JMSException
```

Impostare un Exception Listener per questa connessione.

**Parametri:**
handler - l'Exception Listener.

**Produce:**
JMSException - condizione di eccezione generica se l'implementazione JMS non riesce a impostare l'Exception Listener per questa connessione.

**start**

```
public void start() throws JMSException
```

Avviare (o riavviare il recapito dei messaggi in entrata della connessione. L'avvio di una sessione avviata viene ignorato.

**Produce:**
JMSException - se l'implementazione JMS non riesce ad avviare il recapito dei messaggi a causa di un errore interno.

**Vedere anche:**
stop

**stop**

```
public void stop() throws JMSException
```

Utilizzato per arrestare temporaneamente il recapito dei messaggi in entrata della connessione. Il recapito può essere riavviato utilizzando il metodo `start`. Quando è arrestato, il recapito a tutti i Message Consumer della connessione viene inibito. Le ricezioni sincrone vengono bloccate ed i messaggi non vengono recapitati ai Message Listener.

L'arresto di una sessione non influenza la possibilità di inviare messaggi. L'arresto di una sessione arrestata viene ignorato.

**Produce:**
JMSException - se l'implementazione JMS non riesce ad arrestare il recapito dei messaggi a causa di un errore interno.

**Vedere anche:**
start

**close**

```
public void close() throws JMSException
```

Poiché un provider potrebbe allocare delle risorse esterne alla JVM per una connessione, i client devono chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido. Non è necessario chiudere le sessioni, i produttori e i consumer di una connessione chiusa.

La chiusura di una connessione determina l'esecuzione del roll-back delle transazioni in elaborazione delle sue sessioni. Nel caso in cui il lavoro di una sessione sia coordinato da un programma di gestione delle transazioni esterno, quando si utilizza una XASession, i metodi di commit e di rollback di una sessione non vengono utilizzati e l'esito del lavoro di una sessione chiusa viene determinato successivamente da un programma di gestione delle transazioni. la chiusura di una connessione NON forza una conferma delle sessioni client riconosciute.

MQ JMS mantiene un pool di connessioni MQSeries disponibili per le sessioni. In determinate circostanze, Connection.close() elimina questo pool. Se un'applicazione utilizza delle connessioni multiple in modo sequenziale, è possibile forzare il pool a restare attivo tra le connessioni JMS. Per eseguire questa operazione, registrare un MQPoolToken con com.ibm.mq.MQEnvironment per la durata dell'applicazione JMS. Per ulteriori informazioni, consultare la sezione "Pool di connessioni" a pagina 71 e la sezione "MQEnvironment" a pagina 97.

**Produce:**
JMSException - se l'implementazione JMS non riesce a chiudere la connessione a causa di un errore interno. Degli esempi sono rappresentati dall'impossibilità di rilasciare risorse oppure di chiudere di una connessione socket.

# ConnectionConsumer

interfaccia pubblica **ConnectionConsumer**

classe MQSeries: **MQConnectionConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionConsumer
```

Per i server applicativi, le Connection forniscono una funzione speciale per creare un ConnectionConsumer. Un valore di destinazione (Destination) ed un valore di selezione delle proprietà (Property Selector) specificano i messaggi da elaborare. Per l'esecuzione di questa elaborazione bisogna inoltre specificare un valore di pool di sessioni server (ServerSessionPool) per il ConnectionConsumer.

Vedere anche: **QueueConnection** e **TopicConnection**.

## Metodi

**close()**

```
public void close() throws JMSException
```

Poiché un provider potrebbe allocare delle risorse esterne alla JVM per un ConnectionConsumer, i client devono chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Produce:**

JMSException - se un'implementazione JMS non riesce a rilasciare le risorse per un ConnectionConsumer o se non riesce a chiudere il Connection Consumer.

**getServerSessionPool()**

```
public ServerSessionPool getServerSessionPool()
                                        throws JMSException
```

Richiamare la sessione server associata con questo Connection Consumer.

**Restituisce:**

Il pool di sessioni server utilizzato da questo Connection Consumer.

**Produce:**

JMSException - se un'implementazione JMS non riesce a richiamare il pool di sessioni server associato a questo Connection Consumer a causa di un errore interno.

# ConnectionFactory

interfaccia pubblica **ConnectionFactory**
Interfacce secondarie: **QueueConnectionFactory**, **TopicConnectionFactory**, **XAQueueConnectionFactory** e **XATopicConnectionFactory**

Classe MQSeries: **MQConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

Una ConnectionFactory incapsula un gruppo di parametri di configurazione delle connessione definiti da un amministratore. Un client la utilizza per creare una connessione con un provider JMS.

Vedere anche: **QueueConnectionFactory**, **TopicConnectionFactory**, **XAQueueConnectionFactory** e **XATopicConnectionFactory**

## Costruttore di MQSeries

**MQConnectionFactory**

```
public MQConnectionFactory()
```

## Metodi

**setDescription ***

```
public void setDescription(String x)
```

Una breve descrizione dell'oggetto.

**getDescription ***

```
public String getDescription()
```

Richiamare la descrizione dell'oggetto.

**setTransportType ***

```
public void setTransportType(int x) throws JMSException
```

Impostare il tipo di trasporto da utilizzare. Può essere JMSC.MQJMS_TP_BINDINGS_MQ o JMSC.MQJMS_TP_CLIENT_MQ_TCPIP.

**getTransportType ***

```
public int getTransportType()
```

Richiamare il tipo di trasporto.

**setClientId ***

```
public void setClientId(String x)
```

Impostare l'id client da utilizzare per tutte le connessioni create utilizzando questa connessione.

**getClientId ***

```
public String getClientId()
```

Richiamare 'id client utilizzato per tutte le connessioni create utilizzando questa ConnectionFactory.

**setQueueManager ***

```
public void setQueueManager(String x) throws JMSException
```

Impostare il nome del Queue Manager con cui stabilire la connessione.

**getQueueManager ***

```
public String getQueueManager()
```

Richiamare il nome del Queue Manager.

**setHostName ***

```
public void setHostName(String hostname)
```

Solo per i client, il nome dell'host con cui stabilire la connessione.

**getHostName ***

```
public String getHostName()
```

Richiamare il nome dell'host.

**setPort ***

```
public void setPort(int port) throws JMSException
```

Impostare la porta per una connessione client.

**Parametri:**
port - il nuovo valore da utilizzare.

**Produce:**
JMSException se la porta è negativa.

**getPort ***

```
public int getPort()
```

Solo per le connessioni client, richiamare il numero della porta.

**setChannel ***

```
public void setChannel(String x) throws JMSException
```

Solo per i client, impostare il canale da utilizzare.

**getChannel ***

```
public String getChannel()
```

Solo per i client, richiamare il canale che è stato utilizzato.

**setCCSID ***

```
public void setCCSID(int x) throws JMSException
```

Impostare il set di caratteri da utilizzare quando si stabilisce una connessione con il Gestore code. Consultare la Tabella 13 a pagina 116 per un elenco dei valori consentiti. Si consiglia di utilizzare il valore predefinito (819) nella maggior parte dei casi.

**getCCSID ***

```
public int getCCSID()
```

Richiamare il CCSID del Queue Manager.

**setReceiveExit ***

```
public void setReceiveExit(String receiveExit)
```

Il nome di una classe che implementa un'uscita di ricezione.

**getReceiveExit ***

```
public String getReceiveExit()
```

Richiamare il nome della classe di uscita di ricezione.

**setReceiveExitInit ***

```
public void setReceiveExitInit(String x)
```

Stringa di inizializzazione passata al costruttore della classe di uscita di ricezione.

**getReceiveExitInit ***

```
public String getReceiveExitInit()
```

Richiamare la stringa di inizializzazione che è stata passata alla classe di uscita di ricezione.

**setSecurityExit ***

```
public void setSecurityExit(String securityExit)
```

Il nome di una classe che implementa un'uscita di sicurezza.

**getSecurityExit ***

```
public String getSecurityExit()
```

Richiamare ilnome della classe di uscita di sicurezza.

**setSecurityExitInit ***

```
public void setSecurityExitInit(String x)
```

Stringa di inizializzazione che viene passata al costruttore dell'uscita di sicurezza.

**getSecurityExitInit ***

```
public String getSecurityExitInit()
```

Richiamare la stringa di inizializzazione dell'uscita di sicurezza.

**setSendExit ***

```
public void setSendExit(String sendExit)
```

Il nome di una classe che implementa un'uscita di invio.

**getSendExit ***

```
public String getSendExit()
```

Richiamare il nome della classe di uscita di invio.

**setSendExitInit ***

```
public void setSendExitInit(String x)
```

Stringa di inizializzazione che viene passata al costruttore dell'uscita di invio.

**getSendExitInit ***

```
public String getSendExitInit()
```

Richiamare la stringa di inizializzazione dell'uscita di invio.

# ConnectionMetaData

interfaccia pubblica **ConnectionMetaData**

Classe MQSeries: **MQConnectionMetaData**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

ConnectionMetaData fornisce le informazioni che descrivono la connessione.

## Costruttore di MQSeries

**MQConnectionMetaData**

```
public MQConnectionMetaData()
```

## Metodi

**getJMSVersion**

```
public java.lang.String getJMSVersion() throws JMSException
```

Richiamare la versione JMS.

**Restituisce:**
la versione JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**getJMSMajorVersion**

```
public int getJMSMajorVersion() throws JMSException
```

Richiamare il numero di versione principale di JMS.

**Restituisce:**
Il numero di versione principale di JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**getJMSMinorVersion**

```
public int getJMSMinorVersion() throws JMSException
```

Richiamare il numero di versione minore di JMS.

**Restituisce:**
Il numero di versione minore di JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**getJMSXPropertyNames**

```
public java.util.Enumeration getJMSXPropertyNames()
                                                  throws JMSException
```

Richiamare un'enumerazione dei nomi delle proprietà JMSX supportate da questa connessione.

**Restituisce:**
Un'enumerazione dei nomi di proprietà JMSX.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei nomi di proprietà.

**getJMSProviderName**

```
public java.lang.String getJMSProviderName()
                                        throws JMSException
```

Richiamare il nome del provider JMS.

**Restituisce:**
il nome del provider JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**getProviderVersion**

```
public java.lang.String getProviderVersion()
                                         throws JMSException
```

Richiamare la versione del provider JMS.

**Restituisce:**
la versione del provider JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**getProviderMajorVersion**

```
public int getProviderMajorVersion() throws JMSException
```

Richiamare il numero di versione principale del provider JMS.

**Restituisce:**
il numero di versione principale del provider JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**getProviderMinorVersion**

```
public int getProviderMinorVersion() throws JMSException
```

Richiamare il numero di versione minore del provider JMS.

**Restituisce:**
il numero di versione minore del provider JMS.

**Produce:**
JMSException - se si verifica un errore interno nell'implementazione JMS durante il richiamo dei metadati.

**toString ***

```
public String toString()
```

**Sostituisce:**
toString nella classe Object.

# DeliveryMode

interfaccia pubblica **DeliveryMode**

Modalità di consegna supportate da JMS.

## Campi

**NON_PERSISTENT**

```
public static final int NON_PERSISTENT
```

Questa è la modalità di consegna più bassa in quanto non richiede che il messaggio sia registrato nella memoria stabile.

**PERSISTENT**

```
public static final int PERSISTENT
```

Questa modalità indica al provider JMS di registrare il messaggio nella memoria stabile come parte dell'operazione di invio del client.

# Destination

interfaccia pubblica **Destination**
Interfacce secondarie: **Queue**, **TemporaryQueue**, **TemporaryTopic** e **Topic**

Classe MQSeries: **MQDestination**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

L'oggetto Destination incapsula indirizzi specifici del fornitore.

Vedere anche: **Queue**, **TemporaryQueue**, **TemporaryTopic** e **Topic**

## Costruttori di MQSeries

**MQDestination**

```
public MQDestination()
```

## Metodi

**setDescription ***

```
public void setDescription(String x)
```

Una breve descrizione dell'oggetto.

**getDescription ***

```
public String getDescription()
```

Richiamare la descrizione dell'oggetto.

**setPriority ***

```
public void setPriority(int priority) throws JMSException
```

Utilizzato per ignorare la priorità di tutti i messaggi inviati a questa destinazione.

**getPriority ***

```
public int getPriority()
```

Richiamare il valore della proprietà da ignorare.

**setExpiry ***

```
public void setExpiry(int expiry) throws JMSException
```

Utilizzato per ignorare la scadenza di tutti i messaggi inviati a questa destinazione.

**getExpiry ***

```
public int getExpiry()
```

Richiamare il valore della scadenza relativa a questa destinazione.

**setPersistence ***

```
public void setPersistence(int persistence)
                                    throws JMSException
```

Utilizzato per ignorare la persistenza di tutti i messaggi inviati a questa destinazione.

**getPersistence ***

```
public int getPersistence()
```

Richiamare il valore della persistenza relativa a questa destinazione.

**setTargetClient ***

```
public void setTargetClient(int targetClient)
                                  throws JMSException
```

Flag per indicare se l'applicazione remota è conforme a JMS.

**getTargetClient ***

```
public int getTargetClient()
```

Richiamare il flag che indica la conformità a JMS.

**setCCSID ***

```
public void setCCSID(int x) throws JMSException
```

Set di caratteri da utilizzare per codificare stringhe di testo in messaggi inviati a questa destinazione. Consultare la Tabella 13 a pagina 116 per un elenco dei valori consentiti. Il valore predefinito è 1208 (UTF8).

**getCCSID ***

```
public int getCCSID()
```

Richiamare il nome del set di caratteri utilizzato da questa destinazione.

**setEncoding ***

```
public void setEncoding(int x) throws JMSException
```

Specifica la codifica da utilizzare per i campi numerici nei messaggi inviati a questa destinazione. Consultare la Tabella 13 a pagina 116 per un elenco dei valori consentiti.

**getEncoding ***

```
public int getEncoding()
```

Richiamare la codifica utilizzata per questa destinazione.

# ExceptionListener

interfaccia pubblica **ExceptionListener**

Se un provider JMS rileva un grave problema con una Connection, informerà l'ExceptionListener di Connection se ne è stato registrato uno. A tale scopo viene richiamato il metodo onException() del listener, passandolo a una JMSException che descrive il problema.

In questo modo un client viene notificato in modo asincrono di un problema. Alcune Connection consumano solo i messaggi affinché non ci sia altro modo per apprendere che le proprie Connection non sono riuscite.

Le eccezioni vengono recapitate quando:
- C'è un problema nella ricezione di un messaggio asincrono
- Un messaggio produce un'eccezione in fase di runtime

## Metodi

**onException**

```
public void onException(eccezione JMSException)
```

Segnalare all'utente un'eccezione JMS.

**Parametri:**

exception - l'eccezione JMS. Si tratta delle eccezioni prodotte dalla consegna del messaggio asincrono. Generalmente indicano un problema nella ricezione di un messaggio dal gestore code o possibilmente un errore interno nell'implementazione JMS.

# MapMessage

interfaccia pubblica **MapMessage**
estende **Message**

Classe MQSeries: **JMSMapMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

Una MapMessage viene utilizzata per inviare una serie di coppie nome-valore in cui i nomi sono String e i valori sono tipi di valori primitivi Java. E' possibile accedere a questi elementi in modo sequenziale oppure in modo casuale per nome. L'ordine degli elementi non è definito.

Vedere anche: **BytesMessage**, **Message**, **ObjectMessage**, **StreamMessage** e **TextMessage**

## Metodi

**getBoolean**

```
public boolean getBoolean(java.lang.String name)
                                                throws JMSException
```

Restituire il valore boolean con il nome assegnato.

**Parametri:**
name - il nome del valore boolean.

**Restituisce:**
il valore boolean con il nome assegnato.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getByte**

```
public byte getByte(java.lang.String name)
                                        throws JMSException
```

Restituire il valore byte con il nome indicato.

**Parametri:**
name - il nome del valore byte.

**Restituisce:**
il valore byte con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getShort**

```
public short getShort(java.lang.String name) throws JMSException
```

Restituire il valore short con il nome indicato.

**Parametri:**
name - il nome del valore short.

**Restituisce:**
il valore short con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getChar**

```
public char getChar(java.lang.String name)
                                        throws JMSException
```

Restituire il valore del carattere Unicode con il nome indicato.

**Parametri:**
name - il nome del carattere Unicode.

**Restituisce:**
il valore del carattere Unicode con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getInt**

```
public int getInt(java.lang.String name)
                                             throws JMSException
```

Restituire il valore integer con il nome indicato.

**Parametri:**
name - il nome del valore integer.

**Restituisce:**
il valore integer con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getLong**

```
public long getLong(java.lang.String name)
                                           throws JMSException
```

Restituire il valore long con il nome indicato.

**Parametri:**
name - il nome del valore long.

**Restituisce:**
il valore long con il nome indicato.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getFloat**

```
public float getFloat(java.lang.String name) throws JMSException
```

Restituire il valore float con il nome indicato.

**Parametri:**
name - il nome del valore float.

**Restituisce:**
il valore float con il nome indicato.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getDouble**

```
public double getDouble(java.lang.String name) throws JMSException
```

Restituire il valore double con il nome indicato.

**Parametri:**
name - il nome del valore double.

**Restituisce:**
il valore double con il nome indicato.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getString**

```
public java.lang.String getString(java.lang.String name)
                                                  throws JMSException
```

Restituire il valore String con il nome indicato.

**Parametri:**
name - il nome del valore String.

**Restituisce:**
il valore String con il nome indicato. Se non esiste alcun elemento con questo nome, viene restituito un valore null.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

- MessageFormatException - se questo tipo di conversione non è valido.

**getBytes**

```
public byte[] getBytes(java.lang.String name) throws JMSException
```

Restituire il valore della matrice di byte con il nome indicato.

**Parametri:**
name - il nome della matrice di byte.

**Restituisce:**
una copia del valore della matrice di byte con il nome indicato. Se non esiste alcun elemento con questo nome, viene restituito un valore null.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getObject**

```
public java.lang.Object getObject(java.lang.String name)
                                  throws JMSException
```

Restituisce il valore dell'oggetto Java con il nome indicato. Questo metodo restituisce un formato dell'oggetto, un valore memorizzato nel Map mediante la chiamata del metodo setObject o il metodo set del valore primitivo corrispondente.

**Parametri:**
nme - il nome dell'oggetto Java.

**Restituisce:**
una copia del valore dell'oggetto Java con il nome assegnato, in formato oggetto (se è impostato come int, viene restituito un Integer). Se non esiste alcun elemento con questo nome, viene restituito un valore null.

**Produce:**
JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**getMapNames**

```
public java.util.Enumeration getMapNames() throws JMSException
```

Restituire una enumerazione di tutti i nomi del messaggio Map.

**Restituisce:**
una enumerazione di tutti i nomi in questo messaggio Map.

**Produce:**
JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

**setBoolean**

```
public void setBoolean(java.lang.String name,
                       boolean value) throws JMSException
```

Impostare un valore boolean con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore boolean.
- value - il valore boolean da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

## MapMessage

**setByte**

```
public void setByte(java.lang.String name,
                    byte value) throws JMSException
```

Impostare un valore byte con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore byte.
- value - il valore byte da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setShort**

```
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

Impostare un valore short con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore short.
- value - il valore short da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setChar**

```
public void setChar(java.lang.String name,
                    char value) throws JMSException
```

Impostare un valore del carattere Unicode con il nome indicato nel Map.

**Parametri:**

- name - il nome del carattere Unicode.
- value - il valore del carattere Unicode da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setInt**

```
public void setInt(java.lang.String name,
                   int value) throws JMSException
```

Impostare un valore integer con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore integer.
- value - il valore integer da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setLong**

```
public void setLong(java.lang.String name,
                    long value) throws JMSException
```

Impostare un valore long con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore long.
- value - il valore long da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setFloat**

```
public void setFloat(java.lang.String name,
                     float value) throws JMSException
```

Impostare un valore float con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore float.
- value - il valore float da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setDouble**

```
public void setDouble(java.lang.String name,
                      double value) throws JMSException
```

Impostare un valore double con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore double.
- value - il valore double da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

## MapMessage

### setString

```
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

Impostare un valore String con il nome indicato nel Map.

**Parametri:**

- name - il nome del valore String.
- value - il valore String da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

### setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value) throws JMSException
```

Impostare un valore della matrice di byte con il nome indicato nel Map.

**Parametri:**

- name - il nome della matrice di byte.
- value - il valore della matrice di byte da impostare nel Map.

  La matrice viene copiata, in modo che il valore nel map non sia alterato da modifiche successive sulla matrice.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

### setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value,
                     int offset,
                     int length) throws JMSException
```

Impostare una porzione della matrice di byte con il nome indicato nel Map.

La matrice viene chiamata, in modo che il valore nel map non sia alterato da modifiche successive sulla matrice.

**Parametri:**

- name - il nome della matrice di byte.
- value - il valore della matrice di byte da impostare nel Map.
- offset - l'offset iniziale nella matrice di byte.
- length - il numero di byte da copiare.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**setObject**

```
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

Impostare un valore dell'oggetto Java con il nome indicato nel Map. Questo metodo funziona solo per i tipi di valori primitivi dell'oggetto (Integer, Double, Long, ad esempio), String e matrici di byte.

**Parametri:**

- nme - il nome dell'oggetto Java.
- valore - il valore dell'oggetto Java da impostare nel Map.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di qualche errore JMS interno.
- MessageFormatException - se l'oggetto non è valido.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**itemExists**

```
public boolean itemExists(java.lang.String name)
                                      throws JMSException
```

Verificare se un elemento esiste in questo MapMessage.

**Parametri:**

name - il nome dell'elemento da verificare.

**Restituisce:**

true se l'elemento esiste.

**Produce:**

JMSException - se si verifica un errore JMS.

# Message

public interface **Message**
Interfacce secondarie: **BytesMessage**, **MapMessage**, **ObjectMessage**, **StreamMessage** e **TextMessage**

Classe MQSeries: **JMSMessage**

```
java.lang.Object
  |
  +----com.ibm.jms.MQJMSMessage
```

L'interfaccia Message è l'interfaccia principale di tutti i messaggi JMS. Essa definisce l'intestazione JMS ed il metodo di comunicazione di avvenuta ricezione per tutti i messaggi.

## Campi

**DEFAULT_DELIVERY_MODE**

```
public static final int DEFAULT_DELIVERY_MODE
```

Il valore della modalità di consegna predefinita.

**DEFAULT_PRIORITY**

```
public static final int DEFAULT_PRIORITY
```

Il valore della priorità predefinita.

**DEFAULT_TIME_TO_LIVE**

```
public static final long DEFAULT_TIME_TO_LIVE
```

Il valore della durata predefinita.

## Metodi

**getJMSMessageID**

```
public java.lang.String getJMSMessageID()
                                        throws JMSException
```

Richiamare l'ID del messaggio.

**Restituisce:**
l'ID del messaggio.

**Produce:**
JMSException - se JMS non riesce ad ottenere l'ID del messaggio a causa di un errore JMS interno.

**Vedere anche:**
setJMSMessageID()

**setJMSMessageID**

```
public void setJMSMessageID(java.lang.String id)
                                               throws JMSException
```

Impostare l'ID di messaggio.

Un valore impostato utilizzando questo metodo viene ignorato quando il messaggio viene inviato, ma questo metodo può essere utilizzato per modificare il valore in un messaggio ricevuto.

**Parametri:**
id - l'ID del messaggio.

**Produce:**
JMSException - se JMS non riesce ad impostare l'ID di messaggio a causa di un errore JMS interno.

**Vedere anche:**
getJMSMessageID()

### getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException
```

Ottenere la data/ora del messaggio.

**Restituisce:**
La data/ora del messaggio.

**Produce:**
JMSException - se JMS non riesce ad ottenere la data/ora del messaggio a causa di un errore JMS interno.

**Vedere anche:**
setJMSTimestamp()

### setJMSTimestamp

```
public void setJMSTimestamp(long timestamp)
                                        throws JMSException
```

Impostare la data/ora del messaggio.

Un valore impostato utilizzando questo metodo viene ignorato quando il messaggio viene inviato, ma questo metodo può essere utilizzato per modificare il valore in un messaggio ricevuto.

**Parametri:**
timestamp - la data/ora per questo messaggio.

**Produce:**
JMSException - se JMS non riesce ad impostare la data/ora del messaggio a causa di un errore JMS interno.

**Vedere anche:**
getJMSTimestamp()

### getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes()
                                                    throws JMSException
```

Ottenere il correlationID come una matrice di byte per il messaggio.

**Restituisce:**
Il correlationID di un messaggio come una matrice di byte.

**Produce:**
JMSException - se JMS non riesce ad ottenere il correlationID a causa di un errore JMS interno.

**Vedere anche:**
setJMSCorrelationID(), getJMSCorrelationID(),
setJMSCorrelationIDAsBytes()

**setJMSCorrelationIDAsBytes**

```
public void setJMSCorrelationIDAsBytes(byte[]
                                       correlationID)
                                              throws JMSException
```

Impostare il correlationID come una matrice di byte per il messaggio. Un client può utilizzare questa chiamata per impostare il correlationID in modo che sia uguale ad un messageID di un messaggio precedente oppure ad una stringa specifica per le applicazioni. Le stringhe specifiche dell'applicazione non devono iniziare con gli ID dei caratteri:

**Parametri:**
correlationID - l'ID di correlazione come stringa o l'ID messaggio di un messaggio a cui viene fatto riferimento.

**Produce:**
JMSException - se JMS non riesce ad impostare l'ID di correlazione a causa di un errore JMS interno.

**Vedere anche:**
setJMSCorrelationID(), getJMSCorrelationID(), getJMSCorrelationIDAsBytes()

**getJMSCorrelationID**

```
public java.lang.String getJMSCorrelationID()
                                        throws JMSException
```

Richiamare l'ID di correlazione relativo al messaggio.

**Restituisce:**
Il correlationID di un messaggio come String.

**Produce:**
JMSException - se JMS non riesce a richiamare l'ID di correlazione a causa di un errore JMS interno.

**Vedere anche:**
setJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()

**setJMSCorrelationID**

```
public void setJMSCorrelationID
                  (java.lang.String correlationID)
                                       throws JMSException
```

Impostare l'ID di correlazione relativo al messaggio.

Un client può utilizzare il campo dell'intestazione JMSCorrelationID per collegare un messaggio all'altro. Un utilizzo tipico è il collegamento di un messaggio di risposta al relativo messaggio di richiesta.

**Nota:** L'utilizzo di un valore byte [] per JMSCorrelationID non è portabile.

**Parametri:**
correlationID - l'ID messaggio di un messaggio a cui viene fatto riferimento.

**Produce:**
JMSException - se JMS non riesce ad impostare l'ID di correlazione a causa di un errore JMS interno.

**Vedere anche:**
getJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()

**getJMSReplyTo**

```
public Destination getJMSReplyTo() throws JMSException
```

Ottenere la destinazione di una risposta al messaggio in questione.

**Restituisce:**
la destinazione di una risposta al messaggio in questione

**Produce:**
JMSException - se JMS non riesce a richiamare ReplyTo Destination a causa di un errore JMS interno.

**Vedere anche:**
setJMSReplyTo()

**setJMSReplyTo**

```
public void setJMSReplyTo(Destination replyTo)
                                          throws JMSException
```

Impostare la destinazione di una risposta al messaggio in questione.

**Parametri:**
replyTo - la destinazione di una risposta al messaggio in questione. Un valore null indica che non è prevista alcuna risposta.

**Produce:**
JMSException - se JMS non riesce a impostare ReplyTo Destination a causa di un errore JMS interno.

**Vedere anche:**
getJMSReplyTo()

**getJMSDestination**

```
public Destination getJMSDestination() throws JMSException
```

Richiamare la destinazione del messaggio.

**Restituisce:**
la destinazione del messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare JMS Destination a causa di un errore JMS interno.

**Vedere anche:**
setJMSDestination()

**setJMSDestination**

```
public void setJMSDestination(Destination destination)
                                                 throws JMSexception
```

Impostare la destinazione del messaggio.

Un valore impostato utilizzando questo metodo viene ignorato quando il messaggio viene inviato, ma questo metodo può essere utilizzato per modificare il valore in un messaggio ricevuto.

**Parametri:**
destination - la destinazione del messaggio.

**Produce:**
JMSException - se JMS non riesce a impostare JMS Destination a causa di un errore JMS interno.

**Vedere anche:**
getJMSDestination()

**getJMSDeliveryMode**

```
public int getJMSDeliveryMode() throws JMSException
```

Richiamare la modalità di consegna del messaggio.

**Restituisce:**
la modalità di consegna del messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare JMS DeliveryMode a causa di un errore JMS interno.

**Vedere anche:**
setJMSDeliveryMode(), DeliveryMode

**setJMSDeliveryMode**

```
public void setJMSDeliveryMode(int deliveryMode)
                                          throws JMSException
```

Impostare la modalità di consegna del messaggio.

Un valore impostato utilizzando questo metodo viene ignorato quando il messaggio viene inviato, ma questo metodo può essere utilizzato per modificare il valore in un messaggio ricevuto.

Per modificare la modalità di consegna quando viene inviato un messaggio, utilizzare il metodo setDeliveryMode su QueueSender o su TopicPublisher (questo metodo viene ereditato da MessageProducer).

**Parametri:**
deliveryMode - la modalità di consegna del messaggio.

**Produce:**
JMSException - se JMS non riesce a impostare JMS DeliveryMode a causa di un errore JMS interno.

**Vedere anche:**
getJMSDeliveryMode(), DeliveryMode

**getJMSRedelivered**

```
public boolean getJMSRedelivered() throws JMSException
```

Ottenere un'indicazione dell'eventuale ripetuta consegna del messaggio.

Se un client riceve un messaggio in cui è impostato l'indicatore della ripetuta consegna,è probabile, ma non garantito, che il messaggio sia stato consegnato al client in precedenza ma che il client non ne abbia riconosciuto la ricevuta in quel momento.

**Restituisce:**
impostato su true se è stata ripetuta la consegna del messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare il flag di JMS Redelivered a causa di un errore JMS interno.

**Vedere anche:**
setJMSRedelivered()

**setJMSRedelivered**

```
public void setJMSRedelivered(boolean redelivered)
                                                throws JMSException
```

Impostare per indicare se è stata ripetuta la consegna del messaggio.

Un valore impostato utilizzando questo metodo viene ignorato quando il messaggio viene inviato, ma questo metodo può essere utilizzato per modificare il valore in un messaggio ricevuto.

**Parametri:**
redelivered - un'indicazione dell'eventuale ripetuta consegna del messaggio.

**Produce:**
JMSException - se JMS non riesce a impostare il flag JMSRedelivered a causa di un errore JMS interno.

**Vedere anche:**
getJMSRedelivered()

**getJMSType**

```
public java.lang.String getJMSType() throws JMSException
```

Richiamare il tipo di messaggio.

**Restituisce:**
il tipo di messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare il tipo di messaggio JMS a causa di un errore JMS interno.

**Vedere anche:**
setJMSType()

**setJMSType**

```
public void setJMSType(java.lang.String type)
                                          throws JMSException
```

Impostare il tipo di messaggio.

I client JMS dovrebbero assegnare un valore al tipo indipendentemente dal fatto che l'applicazione ne faccia uso o meno. Questa operazione garantisce che sia impostato correttamente per i fornitori che lo richiedono.

**Parametri:**
type - la classe del messaggio.

**Produce:**
JMSException - se JMS non riesce a impostare il tipo di messaggio JMS a causa di un errore JMS interno.

**Vedere anche:**
getJMSType()

**getJMSExpiration**

```
public long getJMSExpiration() throws JMSException
```

Richiamare il valore di scadenza del messaggio.

**Restituisce:**
l'ora di scadenza del messaggio. Si tratta della somma del valore della durata residua specificata dal client e il GMT all'ora dell'invio.

**Produce:**
JMSException - se JMS non riesce a richiamare la scadenza del messaggio JMS a causa di un errore JMS interno.

**Vedere anche:**
setJMSExpiration()

**setJMSExpiration**

```
public void setJMSExpiration(long expiration)
                                        throws JMSException
```

Impostare il valore di scadenza del messaggio.

Un valore impostato utilizzando questo metodo viene ignorato quando il messaggio viene inviato, ma questo metodo può essere utilizzato per modificare il valore in un messaggio ricevuto.

**Parametri:**
expiration - l'ora di scadenza del messaggio.

**Produce:**
JMSException - se JMS non riesce a impostare la scadenza del messaggio JMS a causa di un errore JMS interno.

**Vedere anche:**
getJMSExpiration()

**getJMSPriority**

```
public int getJMSPriority() throws JMSException
```

Richiamare la priorità del messaggio.

**Restituisce:**
la priorità del messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare la priorità del messaggio JMS a causa di un errore JMS interno.

**Vedere anche:**
setJMSPriority() per i livelli di priorità

**setJMSPriority**

```
public void setJMSPriority(int priority)
                                     throws JMSException
```

Impostare la priorità del messaggio.

JMS definisce un valore della priorità a dieci livelli, con 0 come priorità più bassa e 9 come priorità più alta. Inoltre i client dovrebbero considerare i valori da 0 a 4 come gradazioni della priorità normale e i valori da 5 a 9 come gradazioni della priorità estesa.

**Parametri:**
priority - la priorità del messaggio.

**Produce:**
JMSException - se JMS non riesce a impostare la priorità del messaggio JMS a causa di un errore JMS interno.

**Vedere anche:**
getJMSPriority()

**clearProperties**

```
public void clearProperties() throws JMSException
```

Cancellare le proprietà di un messaggio. I campi dell'intestazione e il corpo del messaggio non vengono cancellati.

**Produce:**
JMSException - se JMS non riesce a cancellare le proprietà del messaggio JMS a causa di un errore JMS interno.

**propertyExists**

```
public boolean propertyExists(java.lang.String name)
                                                throws JMSException
```

Verificare se esiste un valore della proprietà.

**Parametri:**
name - il nome della proprietà da verificare.

**Restituisce:**
true se la proprietà esiste.

**Produce:**
JMSException - se JMS non riesce a verificare l'esistenza di una proprietà a causa di un errore JMS interno.

**getBooleanProperty**

```
public boolean getBooleanProperty(java.lang.String name)
                                                    throws JMSException
```

Restituire il valore della proprietà boolean con quel determinato nome.

**Parametri:**
name - il nome della proprietà boolean.

**Restituisce:**
il valore della proprietà boolean con quel determinato nome.

**Produce:**

- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido

**getByteProperty**

```
public byte getByteProperty(java.lang.String name)
                                                throws JMSException
```

Restituisce il valore di proprietà byte con il nome indicato.

**Parametri:**
name - il nome della proprietà byte.

**Restituisce:**
il valore della proprietà byte con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getShortProperty**

```
public short getShortProperty(java.lang.String name)
                                                    throws JMSException
```

Restituire il valore della proprietà short con il nome indicato.

**Parametri:**
name - il nome della proprietà short.

**Restituisce:**
il valore della proprietà short con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getIntProperty**

```
public int getIntProperty(java.lang.String name)
                                                  throws JMSException
```

Restituire il valore della proprietà integer con il nome indicato.

**Parametri:**
name - il nome della proprietà integer.

**Restituisce:**
il valore della proprietà integer con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getLongProperty**

```
public long getLongProperty(java.lang.String name)
                                                   throws JMSException
```

Restituire il valore della proprietà long con il nome indicato.

**Parametri:**
name - il nome della proprietà long.

**Restituisce:**
il valore della proprietà long con il nome indicato.

**Produce:**

- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getFloatProperty**

```
public float getFloatProperty(java.lang.String name)
                                                    throws JMSException
```

Restituire il valore della proprietà float con il nome indicato.

**Parametri:**
name - il nome della proprietà float.

**Restituisce:**
il valore della proprietà float con il nome indicato.

**Produce:**
- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getDoubleProperty**

```
public double getDoubleProperty(java.lang.String name)
                                                    throws JMSException
```

Restituire il valore della proprietà double con il nome indicato.

**Parametri:**
name - il nome della proprietà double.

**Restituisce:**
il valore della proprietà double con il nome indicato.

**Produce:**
- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getStringProperty**

```
public java.lang.String getStringProperty (java.lang.String name)
                                                              throws JMSException
```

Restituire il valore della proprietà String con il nome indicato.

**Parametri:**
name - il nome della proprietà String.

**Restituisce:**
il valore della proprietà String con il nome indicato. Se non esiste alcuna proprietà in base a questo nome, viene restituito un valore null.

**Produce:**
- JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.
- MessageFormatException - se questo tipo di conversione non è valido.

**getObjectProperty**

```
public java.lang.Object getObjectProperty (java.lang.String name)
                                                              throws JMSException
```

Restituisce il valore della proprietà dell'oggetto Java con il nome indicato.

**Parametri:**
nome - il nome della proprietà dell'oggetto Java.

**Restituisce:**
il valore della proprietà object Java con il nome indicato in formato

object. Se, ad esempio è stato impostato come int., verrà restituito un Integer. Se non esiste alcuna proprietà in base a questo nome, viene restituito un valore null.

**Produce:**
JMSException - se JMS non riesce ad ottenere la proprietà a causa di un errore JMS interno.

**getPropertyNames**

```
public java.util.Enumeration getPropertyNames()
                                              throws JMSException
```

Restituire una enumerazione di tutti i nomi di proprietà.

**Restituisce:**
una enumerazione di tutti i nomi dei valori delle proprietà.

**Produce:**
JMSException - se JMS non riesce ad ottenere i nomi delle proprietà a causa di un errore JMS interno.

**setBooleanProperty**

```
public void setBooleanProperty(java.lang.String name,
                               boolean value) throws JMSException
```

Impostare un valore della proprietà boolean con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà boolean.
- value - il valore della proprietà boolean da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce a impostare Property a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setByteProperty**

```
public void setByteProperty(java.lang.String name,
                            byte value) throws JMSException
```

Impostare un valore della proprietà byte con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà byte.
- value - il valore della proprietà byte da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce a impostare Property a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setShortProperty**

```
public void setShortProperty(java.lang.String name,
                             short value) throws JMSException
```

Impostare un valore della proprietà short con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà short.
- value - il valore della proprietà short da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce a impostare Property a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setIntProperty**

```
public void setIntProperty(java.lang.String name,
                           int value) throws JMSException
```

Impostare un valore della proprietà integer con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà integer.
- value - il valore della proprietà integer da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce a impostare Property a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setLongProperty**

```
public void setLongProperty(java.lang.String name,
                            long value) throws JMSException
```

Impostare un valore della proprietà long con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà long.
- value - il valore della proprietà long da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce a impostare Property a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setFloatProperty**

```
public void setFloatProperty(java.lang.String name,
                             float value) throws JMSException
```

Impostare un valore della proprietà float con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà float.
- value - il valore della proprietà float da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce ad impostare la proprietà a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setDoubleProperty**

```
public void setDoubleProperty(java.lang.String name,
                              double value) throws JMSException
```

Impostare un valore della proprietà double con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà double.
- value - il valore della proprietà double da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce ad impostare la proprietà a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setStringProperty**

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value) throws JMSException
```

Impostare un valore della proprietà String con il nome indicato nel Message.

**Parametri:**

- name - il nome della proprietà String.
- value - il valore della proprietà String da impostare nel Message.

**Produce:**

- JMSException - se JMS non riesce ad impostare la proprietà a causa di un errore JMS interno.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**setObjectProperty**

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value) throws JMSException
```

Impostare un valore della proprietà con il nome indicato nel Message.

**Parametri:**

- nome - il nome della proprietà dell'oggetto Java.
- valore - il valore della proprietà dell'oggetto Java da impostare nel Messaggio.

**Produce:**

- JMSException - se JMS non riesce a impostare Property a causa di un errore JMS interno.
- MessageFormatException - se l'oggetto non è valido.
- MessageNotWriteableException - se le proprietà sono di sola lettura.

**acknowledge**

```
public void acknowledge() throws JMSException
```

Riconoscere questo e tutti i messaggi precedenti ricevuti dalla sessione.

**Produce:**

JMSException - se JMS non riesce a effettuare il riconoscimento a causa di un errore JMS interno.

**clearBody**

```
public void clearBody() throws JMSException
```

Cancellare il corpo del messaggio. Tutte le altre parti del messaggio non vengono toccate.

**Produce:**

JMSException - se JMS non riesce a cancellare a causa di un errore JMS interno.

# MessageConsumer

interfaccia pubblica **MessageConsumer**
Interfacce secondarie: **QueueReceiver** e **TopicSubscriber**

Classe MQSeries: **MQMessageConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

L'interfaccia principale di tutti i consumer dei messaggi. Un client utilizza un consumer di messaggi per ricevere messaggi da una destinazione.

## Metodi

**getMessageSelector**

```
public java.lang.String getMessageSelector()
                                  throws JMSException
```

Richiamare l'espressione del selettore dei messaggi del consumer di questo messaggio.

**Restituisce:**
il selettore dei messaggi del consumer di questo messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare il selettore dei messaggi a causa di un errore JMS.

**getMessageListener**

```
public MessageListener getMessageListener()
                                      throws JMSException
```

Richiamare il MessageListener del consumer del messaggio.

**Restituisce:**
il listener del consumer del messaggio o null se non è impostato un listener.

**Produce:**
JMSException - se JMS non riesce a richiamare il listener dei messaggi a causa di un errore JMS.

**Vedere anche:**
setMessageListener

**setMessageListener**

```
public void setMessageListener(MessageListener listener)
                                             throws JMSException
```

Impostare il MessageListener del consumer del messaggio.

**Parametri:**
messageListener - i messaggi vengono consegnati al listener.

**Produce:**
JMSException - se JMS non riesce a impostare il listener dei messaggi a causa di un errore JMS.

**Vedere anche:**
getMessageListener

**receive**

```
public Message receive() throws JMSException
```

Ricevere il messaggio successivo prodotto per questo consumer del messaggio.

**Restituisce:**
il messaggio successivo prodotto per questo consumer del messaggio.

**Produce:**
JMSException - se JMS non riesce a ricevere il messaggio successivo a causa di un errore.

**receive**

```
public Message receive(long timeOut) throws JMSException
```

Ricevere il messaggio successivo che arriva entro l'intervallo del timeout specificato. Un valore del timeout pari a zero determina un'attesa indefinita da parte della chiamata fino all'arrivo di un messaggio.

**Parametri:**
timeout - il valore del timeout (in millisecondi).

**Restituisce:**
il messaggio successivo prodotto per questo consumer del messaggio o null se non ne è disponibile uno.

**Produce:**
JMSException - se JMS non riesce a ricevere il messaggio successivo a causa di un errore.

**receiveNoWait**

```
public Message receiveNoWait() throws JMSException
```

Ricevere il messaggio successivo se ne disponibile uno immediatamente.

**Restituisce:**
il messaggio successivo prodotto per questo consumer del messaggio o null se non ne è disponibile uno.

**Produce:**
JMSException - se JMS non riesce a ricevere il messaggio successivo a causa di un errore.

**close**

```
public void close() throws JMSException
```

Dal momento che un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una MessageConsumer, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

Questa chiamata si blocca fino al completamento di un listener di ricezione o del messaggio in corso.

**Produce:**
JMSException - se JMS non riesce a chiudere il consumer a causa di un errore.

# MessageListener

interfaccia pubblica **MessageListener**

Una MessageListener viene utilizzata per ricevere in modo asincrono i messaggi consegnati.

## Metodi

**onMessage**

```
public void onMessage(Message message)
```

Passare un messaggio al Listener.

**Parametri:**
message - il messaggio viene passato al listener.

**Vedere anche**
Session.setMessageListener

# MessageProducer

interfaccia pubblica **MessageProducer**
Interfacce secondarie:
**QueueSender** e **TopicPublisher**

Classe MQSeries: **MQMessageProducer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

Un client utilizza un produttore di messaggi per inviare messaggi a una destinazione.

## Costruttori di MQSeries

**MQMessageProducer**

```
public MQMessageProducer()
```

## Metodi

**setDisableMessageID**

```
public void setDisableMessageID(boolean value)
                                          throws JMSException
```

Impostare l'eventuale disabilitazione degli ID dei messaggi.

Gli ID dei messaggi sono abilitati in base all'impostazione predefinita.

**Nota:** Questo metodo viene ignorato nell'implementazione di MQSeries classi per Java Message Service.

**Parametri:**
value - indica se gli ID dei messaggi sono disabilitati.

**Produce:**
JMSException - se JMS non riesce ad impostare l'ID di messaggio disabilitato a causa di un errore interno.

**getDisableMessageID**

```
public boolean getDisableMessageID() throws JMSException
```

Ottenere un'indicazione dell'eventuale disabilitazione degli ID dei messaggi.

**Restituisce:**
un'indicazione dell'eventuale disabilitazione degli ID dei messaggi.

**Produce:**
JMSException - se JMS non riesce a ottenere l'ID di messaggio disabilitato a causa di un errore interno.

**setDisableMessageTimestamp**

```
public void setDisableMessageTimestamp(boolean value)
                                                 throws JMSException
```

Impostare l'eventuale disabilitazione delle indicazioni data/ora dei messaggi.

Le indicazioni data/ora dei messaggi sono impostate in base all'impostazione predefinita.

**Nota:** Questo metodo viene ignorato nell'implementazione di MQSeries classi per Java Message Service.

**Parametri:**
value - indica se le indicazioni data/ora dei messaggi sono disabilitate.

**Produce:**
JMSException - se JMS non riesce ad impostare l'indicazione data/ora dei messaggi disabilitata a causa di un errore interno.

**getDisableMessageTimestamp**

```
public boolean getDisableMessageTimestamp()
                                        throws JMSException
```

Ottenere un'indicazione dell'eventuale disabilitazione dell'indicazione data/ora dei messaggi.

**Restituisce:**
un'indicazione dell'eventuale disabilitazione degli ID dei messaggi.

**Produce:**
JMSException - se JMS non riesce a ottenere l'indicazione data/ora dei messaggi disabilitata a causa di un errore interno.

**setDeliveryMode**

```
public void setDeliveryMode(int deliveryMode)
                                         throws JMSException
```

Impostare la modalità di consegna predefinita del produttore.

La modalità di consegna è impostata su DeliveryMode.PERSISTENT in base all'impostazione predefinita.

**Parametri:**
deliveryMode - la modalità di consegna dei messaggi relativa al produttore del messaggio.

**Produce:**
JMSException - se JMS non riesce ad impostare la modalità di consegna a causa di un errore interno.

**Vedere anche:**
getDeliveryMode, DeliveryMode.NON_PERSISTENT, DeliveryMode.PERSISTENT

**getDeliveryMode**

```
public int getDeliveryMode() throws JMSException
```

Ottenere la modalità di consegna predefinita del produttore.

**Restituisce:**
la modalità di consegna dei messaggi relativa al produttore del messaggio.

**Produce:**
JMSException - se JMS non riesce a ottenere la modalità di consegna a causa di un errore interno.

**Vedere anche:**
setDeliveryMode

**setPriority**

```
public void setPriority(int priority) throws JMSException
```

Impostare la priorità predefinita del produttore.

La priorità è impostata su 4 in base all'impostazione predefinita.

**Parametri:**
priority - la priorità dei messaggi relativa al produttore del messaggio.

**Produce:**
JMSException - se JMS non riesce ad impostare la priorità a causa di un errore interno.

**Vedere anche:**
getPriority

**getPriority**

```
public int getPriority() throws JMSException
```

Ottenere la priorità predefinita del produttore.

**Restituisce:**
la priorità dei messaggi relativa al produttore del messaggio.

**Produce:**
JMSException - se JMS non riesce ad ottenere la priorità a causa di un errore interno.

**Vedere anche:**
setPriority

**setTimeToLive**

```
public void setTimeToLive(long timeToLive)
                                    throws JMSException
```

Impostare il lasso di tempo predefinito, espresso in millisecondi dall'ora di invio, nel quale un messaggio prodotto dovrebbe essere conservato dal sistema dei messaggi.

Il tempo è impostato su zero in base all'impostazione predefinita.

**Parametri:**
timeToLive - la durata del messaggio in millisecondi; zero rappresenta un tempo illimitato.

**Produce:**
JMSException - se JMS non riesce ad impostare la durata a causa di un errore interno.

**Vedere anche:**
getTimeToLive

**getTimeToLive**

```
public long getTimeToLive() throws JMSException
```

## MessageProducer

Ottenere il lasso di tempo predefinito, espresso in millisecondi dall'ora di invio, nel quale un messaggio prodotto dovrebbe essere conservato dal sistema dei messaggi.

**Restituisce:**
la durata del messaggio in millisecondi; zero rappresenta un tempo illimitato.

**Produce:**
JMSException - se JMS non riesce ad ottenere la durata a causa di un errore interno.

**Vedere anche:**
setTimeToLive

**close**

```
public void close() throws JMSException
```

Dal momento che un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una MessageProducer, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Produce:**

JMSException - se JMS non riesce a chiudere il produttore a causa di un errore.

# MQQueueEnumeration *

classe pubblica **MQQueueEnumeration**
estende **Object** implementa **Enumeration**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

Enumerazione di messaggi su una coda. Questa classe non è definita nella specifica JMS, viene creata chiamando il metodo getEnumeration di MQQueueBrowser. La classe contiene un'istanza MQQueue di base per conservare il cursore di esplorazione. La coda viene chiusa dopo che il cursore ha raggiunto la fine della coda.

Non c'è modo di reimpostare un'istanza di questa classe in quanto si tratta di un meccanismo non ripetibile.

Vedere anche: **MQQueueBrowser**

## Metodi

**hasMoreElements**

```
public boolean hasMoreElements()
```

Indica se è possibile restituire un altro messaggio.

**nextElement**

```
public Object nextElement() throws NoSuchElementException
```

Restituire il messaggio corrente.

Se `hasMoreElements()` restituisce 'true', `nextElement()` restituisce sempre un messaggio. E' possibile per il messaggio restituito superare la data di scadenza tra le chiamate `hasMoreElements()` e `nextElement`.

# ObjectMessage

interfaccia pubblica **ObjectMessage**
estende **Message**

Classe MQSeries: **JMSObjectMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

Una ObjectMessage viene utilizzata per inviare un messaggio contenente un oggetto Java serializzabile. Eredita da Message e aggiunge un corpo contenente un solo riferimento Java. Soltanto oggetti Serializable Java possono essere utilizzati.

Vedere anche: **BytesMessage**, **MapMessage**, **Message**, **StreamMessage** e **TextMessage**

## Metodi

**setObject**

```
public void setObject(java.io.Serializable object)
                                        throws JMSException
```

Impostare l'oggetto serializzabile contenente i dati di questo messaggio. ObjectMessage contiene un'istantanea dell'oggetto al momento della chiamata del metodo setObject(). Le modifiche successive dell'oggetto non hanno effetto sul corpo ObjectMessage.

**Parametri:**
object - i dati del messaggio.

**Produce:**

- JMSException - se JMS non riesce ad impostare l'oggetto a causa di un errore JMS interno.
- MessageFormatException - se la serializzazione dell'oggetto non riesce.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**getObject**

```
public java.io.Serializable getObject()
                              throws JMSException
```

Richiamare l'oggetto serializzabile contenente i dati di questo messaggio. Il valore predefinito è null.

**Restituisce:**
l'oggetto serializzabile contenente i dati di questo messaggio.

**Produce:**

- JMSException - se JMS non riesce a richiamare l'oggetto a causa di un errore JMS interno.
- MessageFormatException - se la deserializzazione dell'oggetto non riesce.

# Queue

interfaccia pubblica **Queue**
estende **Destination**
Interfacce secondarie: **TemporaryQueue**

Classe MQSeries: **MQQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

Un oggetto Queue incapsula un nome di coda specifico del provider. Si tratta del modo in cui un client specifica l'identità di una coda sui metodi JMS.

## Costruttori di MQSeries

**MQQueue ***

```
public MQQueue()
```

Construttore predefinito destinato all'utilizzo da parte dello strumento di amministrazione.

**MQQueue ***

```
public MQQueue(String URIqueue)
```

Creare una nuova istanza di MQQueue. La stringa prende un formato URI, come viene descritto a pagina 194.

**MQQueue ***

```
public MQQueue(String queueManagerName,
               String queueName)
```

## Metodi

**getQueueName**

```
public java.lang.String getQueueName()
                                  throws JMSException
```

Richiamare il nome di questa coda.

I client che dipendono dal nome non sono portabili.

**Restituisce:**
il nome della coda

**Produce:**
JMSException - se l'implementazione JMS per Queue non riesce a restituire il nome della coda a causa di un errore interno.

**toString**

```
public java.lang.String toString()
```

Restituire una versione stampata del nome della coda.

**Restituisce:**
i valori dell'identità specifica del provider per questa coda.

**Sostituisce:**
toString nella classe java.lang.Object

**getReference ***

```
public Reference getReference() throws NamingException
```

Creare un riferimento per questa coda.

**Restituisce:**
un riferimento per questo oggetto

**Produce:**
NamingException

**setBaseQueueName ***

```
public void setBaseQueueName(String x) throws JMSException
```

Impostare il valore del nome della coda di MQSeries.

**Nota:** Questo metodo dovrebbe essere utilizzato solo dallo strumento di amministrazione. Non effettua alcun tentativo di codificare le stringhe di formato queue:qmgr:queue.

**getBaseQueueName ***

```
public String getBaseQueueName()
```

**Restituisce:**
il valore del nome di Queue di MQSeries.

**setBaseQueueManagerName ***

```
public void setBaseQueueManagerName(String x) throws JMSException
```

Impostare il valore del nome del nome del gestore code di MQSeries.

**Nota:** Questo metodo dovrebbe essere utilizzato solo dallo strumento di amministrazione.

**getBaseQueueManagerName ***

```
public String getBaseQueueManagerName()
```

**Restituisce:**
il valore del nome del gestore code di MQSeries.

# QueueBrowser

interfaccia pubblica **QueueBrowser**

Classe MQSeries: **MQQueueBrowser**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

Un client utilizza una QueueBrowser per prendere visione dei messaggi su una coda senza rimuoverli.

**Nota:** La classe MQSeries **MQQueueEnumeration**viene utilizzata per mantenere il cursore di esplorazione.

Vedere anche: **QueueReceiver**

## Metodi

**getQueue**

```
public Queue getQueue() throws JMSException
```

Richiamare la coda associata a questo browser della coda.

**Restituisce:**
la coda.

**Produce:**
JMSException - se JMS non riesce a richiamare la coda relativa a questo browser a causa di un errore interno.

**getMessageSelector**

```
public java.lang.String getMessageSelector() throws JMSException
```

Richiamare l'espressione del selettore messaggi del browser di questa coda.

**Restituisce:**
il selettore messaggi del browser di questa coda.

**Produce:**
JMSException - se JMS non riesce a richiamare il selettore messaggi di questo browser a causa di un errore JMS interno.

**getEnumeration**

```
public java.util.Enumeration getEnumeration() throws JMSException
```

Richiamare un'enumerazione per esplorare i messaggi della coda correnti nell'ordine in cui dovrebbero essere ricevuti.

**Restituisce:**
un'enumerazione per l'esplorazione dei messaggi.

**Produce:**
JMSException - se JMS non riesce a richiamare l'enumerazione relativa a questo browser a causa di un errore JMS.

**Nota:** Se il browser viene creato per una coda inesistente, non viene rilevato fino alla prima chiamata a getEnumeration.

**close**

```
public void close() throws JMSException
```

Dal momento che un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una QueueBrowser, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Produce:**

JMSException - se JMS non riesce a chiudere questo Browser a causa di un errore JMS.

# QueueConnection

interfaccia pubblica **QueueConnection**
estende **Connection**
Interfacce secondarie: **XAQueueConnection**

Classe MQSeries: **MQQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection           |
          +----com.ibm.mq.jms.MQQueueConnection
```

Una QueueConnection è una connessione attiva verso un provider JMS point-to-point. Un client utilizza una QueueConnection per creare una o più QueueSession per la produzione e il consumo di messaggi.

Vedere anche: **Connection**, **QueueConnectionFactory** e **XAQueueConnection**

## Metodi

**createQueueSession**

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

Creare una QueueSession.

**Parametri:**

- transacted - se ″true″, la sessione viene negoziata.
- acknowledgeMode - indica se il consumer o il client riconosceranno i messaggi ricevuti. I valori possibili sono:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Questo parametro verrà ignorato se la sessione è negoziata.

**Restituisce:**
una nuova sessione della coda.

**Produce:**
JMSException - se una connessione JMS non riesce a creare una sessione a causa di un errore interno o di una mancanza di supporto per la transazione e la modalità di riconoscimento specifica.

**createConnectionConsumer**

```
public ConnectionConsumer createConnectionConsumer
                                    (Queue queue,
                                     java.lang.String messageSelector,
                                     ServerSessionPool sessionPool,
                                     int maxMessages)
                                                throws JMSException
```

Creare un consumer per questa connessione. Si tratta di una funzione esperta che non viene utilizzata dai client JMS regolari.

**Parametri:**

- queue - la coda a cui accedere.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi.
- sessionPool - il pool di sessioni del server da associare a questo consumer della connessione.
- maxMessages - il numero massimo di messaggi che è possibile assegnare a un server alla volta.

**Restituisce:**

il consumer della connessione.

**Produce:**

- JMSException - se una connessione JMS non riesce a creare un consumer della connessione a causa di un errore interno o di argomenti non validi sessionPool e messageSelector.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

**Vedere anche:**

ConnectionConsumer

**close ***

```
public void close() throws JMSException
```

**Sostituisce:**

close nella classe MQConnection.

# QueueConnectionFactory

interfaccia pubblica **QueueConnectionFactory**
estende **ConnectionFactory**
Interfacce secondarie: **XAQueueConnectionFactory**

Classe MQSeries: **MQQueueConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory          |
          +----com.ibm.mq.jms.MQQueueConnectionFactory
```

Un client utilizza una QueueConnectionFactory per creare QueueConnections con un provider JMS point-to-point.

Vedere anche: **ConnectionFactory** e **XAQueueConnectionFactory**

## Costruttori di MQSeries

**MQQueueConnectionFactory**

```
public MQQueueConnectionFactory()
```

## Metodi

**createQueueConnection**

```
public QueueConnection createQueueConnection()
                                          throws JMSException
```

Creare una connessione della coda con l'identità utente predefinita. La connessione viene creata in modalità arrestata. Nessun messaggio verrà consegnato fino all'esplicito richiamo del metodo Connection.start.

**Restituisce:**
una nuova connessione della coda.

**Produce:**

- JMSException - se la connessione JMS non riesce a creare una Queue Connection a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

**createQueueConnection**

```
public QueueConnection createQueueConnection
                                (java.lang.String userName,
                                 java.lang.String password)
                                                  throws JMSException
```

Creare una connessione della coda con l'identità utente specificata.

**Nota:** Questo metodo può essere utilizzato solo con il tipo di trasporto JMSC.MQJMS_TP_CLIENT_MQ_TCPIP (vedere ConnectionFactory). La connessione viene creata in modalità arrestata. Nessun messaggio verrà consegnato fino all'esplicito richiamo del metodo Connection.start.

**Parametri:**

- userName - il nome utente di chi esegue la chiamata.
- password - la password di chi esegue la chiamata.

**Restituisce:**
una nuova connessione della coda.

**Produce:**

- JMSException - se la connessione JMS non riesce a creare una Queue Connection a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

**setTemporaryModel ***

```
public void setTemporaryModel(String x) throws JMSException
```

**getTemporaryModel ***

```
public String getTemporaryModel()
```

**getReference ***

```
public Reference getReference() throws NamingException
```

Creare un riferimento per il valore predefinito di questa connessione della coda.

**Restituisce:**
un riferimento per questo oggetto.

**Produce:**
NamingException.

**setMessageRetention***

```
public void setMessageRetention(int x) throws JMSException
```

Impostare il metodo per l'attributo messageRetention.

**Parametri:**
I valori validi sono:

- JMSC.MQJMS_MRET_YES - i messaggi non desiderati restano nella coda di input.
- JMSC.MQJMS_MRET_NO - i messaggi non desiderati vengono gestiti in base alle rispettive opzioni di disposizione.

**getMessageRetention***

```
public int getMessageRetention()
```

Richiamare il metodo per l'attributo messageRetention.

**Restituisce:**

- JMSC.MQJMS_MRET_YES - i messaggi non desiderati restano nella coda di input.
- JMSC.MQJMS_MRET_NO - i messaggi non desiderati vengono gestiti in base alle rispettive opzioni di disposizione.

# QueueReceiver

interfaccia pubblica **QueueReceiver**
estende **MessageConsumer**

Classe MQSeries: **MQQueueReceiver**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

Un client utilizza una QueueReceiver per la ricezione dei messaggi consegnati su una coda.

Vedere anche: **MessageConsumer**

Questa classe eredita i seguenti metodi da **MQMessageConsumer**.
- receive
- receiveNoWait
- close
- getMessageListener
- setMessageListener

## Metodi

**getQueue**

```
public Queue getQueue() throws JMSException
```

Richiamare la coda associata a questo destinatario della coda.

**Restituisce:**
la coda.

**Produce:**
JMSException - se JMS non riesce a richiamare la coda relativa a questo destinatario a causa di un errore interno.

# QueueRequestor

classe pubblica **QueueRequestor**
estende **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.QueueRequestor
```

JMS fornisce la classe QueueRequestor per semplificare l'esecuzione di richieste di servizio. Al costruttore QueueRequestor viene fornita una QueueSession non negoziata e una Queue di destinazione. Crea una TemporaryQueue per le risposte e fornisce un metodo request() che invia il messaggio di richiesta e attende la risposta. Gli utenti possono comunque creare risposte più sofisticate.

Vedere anche: **TopicRequestor**

## Costruttori

**QueueRequestor**

```
public QueueRequestor(QueueSession session,
                      Queue queue)
                              throws JMSException
```

Questa implementazione presuppone che il parametro della sessione non sia negoziato e che sia AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE.

**Parametri:**

- session - la sessione della coda a cui appartiene la coda.
- queue - la coda su cui eseguire la chiamata di richiesta/risposta.

**Produce:**

JMSException - se si verifica un errore JMS.

## Metodi

**request**

```
public Message request(Message message)
                                    throws JMSException
```

Inviare una richiesta e attendere una risposta. La coda temporanea viene utilizzata per replyTo e si prevede solo una risposta a richiesta.

**Parametri:**

message - il messaggio da inviare.

**Restituisce:**

il messaggio di risposta.

**Produce:**

JMSException - se si verifica un errore JMS.

## QueueRequestor

### close

```
public void close() throws JMSException
```

Dal momento che un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una QueueRequestor, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Nota:** Questo metodo chiude l'oggetto Session passato al costruttore QueueRequestor.

**Produce:**

JMSException - se si verifica un errore JMS.

# QueueSender

interfaccia pubblica **QueueSender**
estende **MessageProducer**

Classe MQSeries: **MQQueueSender**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

Un client utilizza una QueueSender per inviare messaggi a una coda.

Una QueueSender è normalmente associata a una determinata Queue. Tuttavia è possibile creare una QueueSender non identificata che non è associata a una determinata coda.

Vedere anche: **MessageProducer**

## Metodi

**getQueue**

```
public Queue getQueue() throws JMSException
```

Richiamare la coda associata a questo mittente della coda.

**Restituisce:**
la coda.

**Produce:**
JMSException - se JMS non riesce a richiamare la coda relativa a questo mittente a causa di un errore interno.

**send**

```
public void send(Message message) throws JMSException
```

Inviare un messaggio alla coda. Utilizzare la modalità di consegna, la durata e la priorità predefinita della QueueSender.

**Parametri:**
message - il messaggio da inviare.

**Produce:**

- JMSException - se JMS non riesce a inviare il messaggio a causa di un errore.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con un mittente della coda con una coda non valida.

**send**

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

## QueueSender

Inviare un messaggio specificando la modalità di consegna, la priorità e la durata della coda.

**Parametri:**

- message - il messaggio da inviare.
- deliveryMode - la modalità di consegna da utilizzare.
- priority - la priorità per questo messaggio.
- timeToLive - la durata del messaggio (in millisecondi).

**Produce:**

- JMSException - se JMS non riesce a inviare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con un mittente della coda con una coda non valida.

**send**

```
public void send(Queue queue,
                 Message message) throws JMSException
```

Inviare un messaggio alla coda specificata con la modalità di consegna, la durata e la priorità predefinita della QueueSender.

**Nota:** Questo metodo può essere utilizzato solo con QueueSender non identificate.

**Parametri:**

- queue - la coda a cui dovrebbe essere inviato il messaggio.
- message - il messaggio da inviare.

**Produce:**

- JMSException - se JMS non riesce a inviare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con una coda non valida.

**send**

```
public void send(Queue queue,
                 Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

Inviare un messaggio alla coda specificata con la modalità di consegna, la durata e la priorità.

**Nota:** Questo metodo può essere utilizzato solo con QueueSender non identificate.

**Parametri:**

- queue - la coda a cui dovrebbe essere inviato il messaggio.
- message - il messaggio da inviare.
- deliveryMode - la modalità di consegna da utilizzare.

- priority - la priorità per questo messaggio.
- timeToLive - la durata del messaggio (in millisecondi).

**Produce:**

- JMSException - se JMS non riesce a inviare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con una coda non valida.

**close ***

```
public void close() throws JMSException
```

Poiché un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una QueueSender, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Produce:**

JMSException se JMS non riesce a chiudere il produttore a causa di qualche errore.

**Sostituisce:**

close nella classe MQMessageProducer.

# QueueSession

interfaccia pubblica **QueueSession**
estende **Session**

Classe MQSeries: **MQQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

Una QueueSession fornisce i metodi per creare QueueReceiver, QueueSender, QueueBrowser e TemporaryQueue.

Vedere anche: **Session**

I metodi che seguono vengono ereditati da **MQSession**:
- close
- commit
- rollback
- recover

## Metodi

**createQueue**

```
public Queue createQueue(java.lang.String queueName)
                                              throws JMSException
```

Creare una Queue a cui è stato dato un nome Queue. In questo modo è consentita la creazione di una coda con un nome specifico di provider. La stringa prende un formato URI, come viene descritto a pagina 194.

**Nota:** I client che dipendono da questa capacità non sono portabili.

**Parametri:**
queueName - il nome di questa coda.

**Restituisce:**
una Queue con quel determinato nome.

**Produce:**
JMSException - se una sessione non riesce a creare una coda a causa di un errore JMS.

**createReceiver**

```
public QueueReceiver createReceiver(Queue queue)
                                              throws JMSException
```

Creare una QueueReceiver per ricevere messaggi dalla coda specificata.

**Parametri:**
queue - la coda a cui accedere.

**Produce:**
- JMSException - se una sessione non riesce a creare un destinatario a causa di un errore JMS.

- InvalidDestinationException - se viene specificato un Queue non valido.

**createReceiver**

```
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
                                    throws JMSException
```

Creare una QueueReceiver per ricevere messaggi dalla coda specificata.

**Parametri:**

- queue - la coda a cui accedere.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi.

**Produce:**

- JMSException - se una sessione non riesce a creare un destinatario a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Queue non valido.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

**createSender**

```
public QueueSender createSender(Queue queue)
                                        throws JMSException
```

Creare una QueueSender per inviare messaggi alla coda specificata.

**Parametri:**

queue - la coda a cui accedere o null se deve essere un produttore non identificato.

**Produce:**

- JMSException - se una sessione non riesce a creare un mittente a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Queue non valido.

**createBrowser**

```
public QueueBrowser createBrowser(Queue queue)
                                      throws JMSException
```

Creare una QueueBrowser per dare una rapida occhiata ai messaggi sulla coda specificata.

**Parametri:**

queue - la coda a cui accedere.

**Produce:**

- JMSException - se una sessione non riesce a creare un browser a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Queue non valido.

**createBrowser**

```
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
                                  throws JMSException
```

Creare una QueueBrowser per dare una rapida occhiata ai messaggi sulla coda specificata.

**Parametri:**

- queue - la coda a cui accedere.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi.

**Produce:**

- JMSException - se una sessione non riesce a creare un browser a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Queue non valido.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

### createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                       throws JMSException
```

Creare una coda temporanea. La durata corrisponderà a quella di QueueConnection a meno che non venga eliminato prima.

**Restituisce:**

una coda temporanea.

**Produce:**

JMSException - se una sessione non riesce a creare una coda temporanea a causa di un errore JMS.

# Session

interfaccia pubblica **Session**
estende **java.lang.Runnable**
Interfacce secondarie: **QueueSession**, **TopicSession**, **XAQueueSession**, **XASession** e **XATopicSession**

Classe MQSeries: **MQSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

Una JMS Session è un contesto a singolo thread per la produzione e il consumo di messaggi.

Vedere anche: **QueueSession**, **TopicSession**, **XAQueueSession**, **XASession** e **XATopicSession**

## Campi

**AUTO_ACKNOWLEDGE**

```
public static final int AUTO_ACKNOWLEDGE
```

Con questa modalità di riconoscimento, la sessione riconosce automaticamente un messaggio quando è stato correttamente restituito da una chiamata per ricevere o il listener del messaggio richiamato per elaborare il messaggio viene correttamente restituito.

**CLIENT_ACKNOWLEDGE**

```
public static final int CLIENT_ACKNOWLEDGE
```

Con questa modalità di riconoscimento il client riconosce un messaggio richiamando il metodo di riconoscimento di un messaggio.

**DUPS_OK_ACKNOWLEDGE**

```
public static final int DUPS_OK_ACKNOWLEDGE
```

Questa modalità di riconoscimento indica alla sessione di riconoscere la consegna dei messaggi.

## Metodi

**createBytesMessage**

```
public BytesMessage createBytesMessage()
                                throws JMSException
```

Crea una BytesMessage. Una BytesMessage viene utilizzata per inviare un messaggio che contiene un flusso di byte non interpretati.

**Produce:**
JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**createMapMessage**

```
public MapMessage createMapMessage() throws JMSException
```

Creare una MapMessage. Una MapMessage viene utilizzata per inviare una serie autodefinita di coppie nome-valore in cui i nomi sono String e i valori sono tipi di valori primitivi Java.

**Produce:**

JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**createMessage**

```
public Message createMessage() throws JMSException
```

Crea una Message. L'interfaccia Message è l'interfaccia principale di tutti i messaggi JMS. Contiene tutte le informazioni sull'intestazione del messaggio standard. Può essere inviata quando un messaggio contenente solo le informazioni sull'intestazione è insufficiente.

**Produce:**

JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**createObjectMessage**

```
public ObjectMessage createObjectMessage()
                              throws JMSException
```

Creare una ObjectMessage. Una ObjectMessage viene utilizzata per inviare un messaggio contenente un oggetto Java serializzabile.

**Produce:**

JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**createObjectMessage**

```
public ObjectMessage createObjectMessage
                             (java.io.Serializable object)
                              throws JMSException
```

Creare una ObjectMessage inizializzata. Una ObjectMessage viene utilizzata per inviare un messaggio contenente un oggetto Java.

**Parametri:**

object - l'oggetto da utilizzare per inizializzare questo messaggio.

**Produce:**

JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**createStreamMessage**

```
public StreamMessage createStreamMessage()
                              throws JMSException
```

Creare una StreamMessage. Una StreamMessage viene utilizzata per inviare un flusso autodefinito di valori primitivi Java.

**Produce:**

JMSException se JMS non riesce a creare il messaggio a causa di un errore interno.

## Session

**createTextMessage**

```
public TextMessage createTextMessage() throws JMSException
```

Creare una TextMessage. Una TextMessage viene utilizzata per inviare un messaggio contenente una String.

**Produce:**

JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**createTextMessage**

```
public TextMessage createTextMessage
                      (java.lang.String string)
                      throws JMSException
```

Creare una TextMessage inizializzata. Una TextMessage viene utilizzata per inviare un messaggio contenente una String.

**Parametri:**

string - la stringa utilizzata per inizializzare il messaggio.

**Produce:**

JMSException - se JMS non riesce a creare il messaggio a causa di un errore interno.

**getTransacted**

```
public boolean getTransacted() throws JMSException
```

La sessione è in modalità negoziata?

**Restituisce:**

true se la sessione è in modalità negoziata.

**Produce:**

JMSException - se JMS non riesce a restituire la modalità di transazione a causa di un errore interno nel JMS Provider.

**commit**

```
public void commit() throws JMSException
```

Esegue il commit di tutti i messaggi effettuati in questa transazione e rilasciare tutti i blocchi al momento presenti.

**Produce:**

- JMSException - se l'implementazione JMS non riesce a eseguire il commit della transazione a causa di un errore interno.
- TransactionRolledBackException - se viene eseguito il rollback della transazione a causa di un errore interno durante il commit.

**rollback**

```
public void rollback() throws JMSException
```

Esegue il rollback di tutti i messaggi effettuati in questa transazione e rilasciare tutti i blocchi al momento presenti.

**Produce:**

JMSException - se l'implementazione JMS non riesce a eseguire il rollback della transazione a causa di un errore interno.

**close**

```
public void close() throws JMSException
```

Dal momento che un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una Session, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

La chiusura di una sessione negoziata determina il rollback di tutte le transazioni in corso. La chiusura di una sessione determina la chiusura automatica dei relativi produttori e consumer, in modo che non sia necessario chiuderle singolarmente.

**Produce:**
JMSException - se l'implementazione JMS non riesce a chiudere una Session a causa di un errore interno.

**recover**

```
public void recover() throws JMSException
```

Interrompere la consegna dei messaggi in questa sessione e riavviare l'invio dei messaggi con il messaggio riconosciuto meno recente.

**Produce:**
JMSException - se l'implementazione JMS non riesce a interrompere la consegna dei messaggi e riavviare l'invio a causa di un errore interno.

**getMessageListener**

```
public MessageListener getMessageListener()
                                  throws JMSException
```

Restituisce il listner del messaggio distinto della sessione.

**Restituisce:**
il listener del messaggio associato a questa sessione.

**Produce:**
JMSException - se JMS non riesce a richiamare il listener del messaggio a causa di un errore interno nel JMS Provider.

**Vedere anche:**
setMessageListener

**setMessageListener**

```
public void setMessageListener(MessageListener listener)
                                            throws JMSException
```

Impostare il listener del messaggio distinto della sessione. Quando viene impostato, non è possibile utilizzare alcuna altra forma di ricevuta di messaggio nella sessione. Tuttavia tutte le altre forme di invio dei messaggi sono ancora supportate.

Si tratta di una funzione esperta che non viene utilizzata dai client JMS regolari.

**Parametri:**
listener - il listener del messaggio da associare a questa sessione.

**Produce:**
JMSException - se JMS non riesce a impostare il listener del messaggio a causa di un errore interno nel JMS Provider.

**Vedere anche:**
getMessageListener, ServerSessionPool, ServerSession

**run**

```
public void run()
```

Questo metodo è destinato all'utilizzo esclusivo da parte dei server applicazioni.

**Specificato da:**
`run` nell'interfaccia java.lang.Runnable

**Vedere anche:**
ServerSession

# StreamMessage

interfaccia pubblica **StreamMessage**
estende **Message**

Classe MQSeries: **JMSStreamMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

Un oggetto StreamMessage viene utilizzato per inviare un flusso di valori primitivi Java.

Vedere anche: **BytesMessage**, **MapMessage**, **Message**, **ObjectMessage** e **TextMessage**

## Metodi

**readBoolean**

```
public boolean readBoolean() throws JMSException
```

Leggere un valore booleano dal messaggio del flusso.

**Restituisce:**
il valore boolean read.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException - se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.

**readByte**

```
public byte readByte() throws JMSException
```

Leggere un valore byte dal messaggio del flusso.

**Restituisce:**
il byte successivo dal messaggio del flusso come byte a 8 bit.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException - se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.

**readShort**

```
public short readShort() throws JMSException
```

Leggere un valore a 16 bit dal messaggio del flusso.

**Restituisce:**
un numero a 16 bit dal messaggio del flusso.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException - se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.

**readChar**

```
public char readChar() throws JMSException
```

Leggere un valore di carattere Unicode dal messaggio del flusso.

**Restituisce:**
un carattere Unicode dal messaggio del flusso.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException se questo tipo di conversione non è valido.
- MessageNotReadableException se il messaggio è in modalità di sola scrittura.

**readInt**

```
public int readInt() throws JMSException
```

Leggere un valore a 32 bit dal messaggio del flusso.

**Restituisce:**
un valore intero a 32 bit dal messaggio del flusso, interpretato come un int.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException se questo tipo di conversione non è valido.
- MessageNotReadableException se il messaggio è in modalità di sola scrittura.

**readLong**

```
public long readLong() throws JMSException
```

Leggere un numero intero a 64 bit dal messaggio del flusso.

**Restituisce:**
un valore intero a 64 bit dal messaggio del flusso, interpretato come un long.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - nel caso di una fine del flusso di messaggi
- MessageFormatException se questo tipo di conversione non è valido.
- MessageNotReadableException se il messaggio è in modalità di sola scrittura.

**readFloat**

```
public float readFloat() throws JMSException
```

Leggere un valore mobile dal messaggio del flusso.

**Restituisce:**
un valore mobile dal messaggio del flusso.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - nel caso di una fine del flusso di messaggi
- MessageFormatException se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.

**readDouble**

```
public double readDouble() throws JMSException
```

Leggere un valore doppio dal messaggio del flusso.

**Restituisce:**
un valore doppio dal messaggio del flusso.

**Produce:**
- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException - se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.

**readString**

```
public java.lang.String readString() throws JMSException
```

Leggere in una stringa dal messaggio del flusso.

**Restituisce:**
una stringa Unicode dal messaggio del flusso.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException - se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura

**readBytes**

```
public int readBytes(byte[] value)
            throws JMSExceptioneam message.
```

Leggere un campo di una gamma di byte dal messaggio del flusso nell'oggetto byte[] specificato (il buffer di lettura). Se la dimensione del buffer è inferiore o pari alla dimensione dei dati nel campo del messaggio, è necessario che l'applicazione effettui ulteriori chiamate a questo metodo per richiamare il resto dei dati. Dopo che è stata effettuata la prima chiamata readBytes su un valore del campo byte[], il valore completo del campo deve essere letto prima che sia valido leggere il campo successivo. Un tentativo di leggere il campo successivo prima di questa operazione produrrà una MessageFormatException.

**Parametri:**
value - il buffer in cui sono letti i dati.

**Restituisce:**
il numero totale di byte letto nel buffer oppure -1 se non ci sono più dati in quanto è stato raggiunto il campo della fine dei byte.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- MessageFormatException - se questo tipo di conversione non è valido.
- MessageNotReadableException - se il messaggio è in modalità di sola scrittura.

**readObject**

```
public java.lang.Object readObject() throws JMSException
```

Leggere un oggetto Java dal messaggio del flusso.

**Restituisce:**
un oggetto Java dal messaggio del flusso in formato oggetto. Se, ad esempio è stato impostato come int., verrà restituito un Integer.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.

- MessageEOFException - se viene ricevuta una fine del flusso di messaggi.
- NotReadableException - se il messaggio è in modalità di sola scrittura.

**writeBoolean**

```
public void writeBoolean(boolean value) throws JMSException
```

Scrivere un valore booleano sul messaggio del flusso.

**Parametri:**

value - il valore booleano da scrivere.

**Produce:**

- JMSException - se JMS non riesce a leggere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

## StreamMessage

**writeByte**

```
public void writeByte(byte value) throws JMSException
```

Scrivere un byte sul messaggio del flusso.

**Parametri:**

value - il valore byte da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeShort**

```
public void writeShort(short value) throws JMSException
```

Scrivere un valore short sul messaggio del flusso.

**Parametri:**

value - il valore short da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeChar**

```
public void writeChar(char value) throws JMSException
```

Scrivere un valore char sul messaggio del flusso.

**Parametri:**

value - il valore char da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeInt**

```
public void writeInt(int value) throws JMSException
```

Scrivere un valore int sul messaggio del flusso.

**Parametri:**

value - il valore int da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeLong**

```
public void writeLong(long value) throws JMSException
```

Scrivere un valore long sul messaggio del flusso.

**Parametri:**
value - il valore long da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeFloat**

```
public void writeFloat(float value) throws JMSException
```

Scrivere un valore float sul messaggio del flusso.

**Parametri:**
value - il valore float da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeDouble**

```
public void writeDouble(double value) throws JMSException
```

Scrivere un valore double sul messaggio del flusso.

**Parametri:**
value - il valore double da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeString**

```
public void writeString(java.lang.String value)
                                          throws JMSException
```

Scrivere un valore string sul messaggio del flusso.

**Parametri:**
value - il valore String da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeBytes**

```
public void writeBytes(byte[] value) throws JMSException
```

Scrivere una matrice di byte nel messaggio del flusso.

**Parametri:**

value - la matrice di byte da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeBytes**

```
public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException
```

Scrivere una porzione di una matrice di byte nel messaggio del flusso.

**Parametri:**

- value - il valore della matrice di byte da scrivere.
- offset - l'offset iniziale nella matrice di byte.
- length - il numero di byte da utilizzare.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**writeObject**

```
public void writeObject(java.lang.Object value)
                                          throws JMSException
```

Scrivere un oggetto Java nel messaggio del flusso. Questo metodo funziona solo per i tipi di oggetti primitivi (Integer, Double e Long), per le stringhe e per le matrici di byte.

**Parametri:**

value - l'oggetto Java da scrivere.

**Produce:**

- JMSException - se JMS non riesce a scrivere il messaggio a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.
- MessageFormatException - se l'oggetto non è valido.

**reset**

```
public void reset() throws JMSException
```

Inserire il messaggio in modalità di sola lettura e riposizionare il flusso all'inizio.

**Produce:**

- JMSException - se JMS non riesce a reimpostare il messaggio a causa di un errore JMS interno.
- MessageFormatException - se il messaggio è in un formato non valido.

# TemporaryQueue

interfaccia pubblica **TemporaryQueue**
estende **Queue**

Classe MQSeries: **MQTemporaryQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

Un TemporaryQueue è un oggetto Queue univoco creato per la durata di una QueueConnection.

## Metodi

**delete**

```
public void delete() throws JMSException
```

Eliminare questa coda temporanea. Se ci sono ancora mittenti o destinatari che la utilizzano, verrà prodotta una JMSException.

**Produce:**

JMSException - se un'implementazione JMS non riesce a eliminare un TemporaryQueue a causa di un errore interno.

# TemporaryTopic

interfaccia pubblica **TemporaryTopic**
estende **Topic**

Classe MQSeries: **MQTemporaryTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

Un TemporaryTopic è un oggetto Topic univoco creato per la durata di una TopicConnection è può essere consumato solo dai consumer di quella connessione.

## Costruttori di MQSeries

**MQTemporaryTopic**

```
MQTemporaryTopic() throws JMSException
```

## Metodi

**delete**

```
public void delete() throws JMSException
```

Eliminare questo argomento temporaneo. Se ci sono ancora editori o sottoscrittori che lo utilizzano, verrà prodotta una JMSException.

**Produce:**

JMSException - se un'implementazione JMS non riesce a eliminare un TemporaryTopic a causa di un errore interno.

# TextMessage

interfaccia pubblica **TextMessage**
estende **Message**

Classe MQSeries: **JMSTextMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

TextMessage viene utilizzata per inviare un messaggio contenente una java.lang.String. Eredita da Message e aggiunge un corpo del messaggio di testo.

Vedere anche: **BytesMessage**, **MapMessage**, **Message**, **ObjectMessage** e **StreamMessage**

## Metodi

**setText**

```
public void setText(java.lang.String string)
                                        throws JMSException
```

Impostare la stringa contenente i dati di questo messaggio.

**Parametri:**
string - la String contenente i dati del messaggio.

**Produce:**

- JMSException - se JMS non riesce a impostare il testo a causa di un errore JMS interno.
- MessageNotWriteableException - se il messaggio è in modalità di sola lettura.

**getText**

```
public java.lang.String getText() throws JMSException
```

Richiamare la stringa contenente i dati di questo messaggio. Il valore predefinito è null.

**Restituisce:**
la String contenente i dati del messaggio.

**Produce:**
JMSException - se JMS non riesce a richiamare il testo a causa di un errore JMS interno.

# Topic

interfaccia pubblica **Topic**
estende **Destination**
Interfacce secondarie: **TemporaryTopic**

Classe MQSeries: **MQTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

Un oggetto Topic incapsula un nome di argomento specifico del provider. Si tratta del modo in cui un client specifica l'identità di un argomento sui metodi JMS.

Vedere anche: **Destination**

## Costruttori di MQSeries

**MQTopic**

```
public MQTopic()
public MQTopic(string URItopic)
```

Vedere **TopicSession.createTopic**.

## Metodi

**getTopicName**

```
public java.lang.String getTopicName() throws JMSException
```

Richiamare il nome di questo argomento in formato URI. (Il formato URI è descritto in "Creazione di argomenti in fase di runtime" a pagina 204.)

**Nota:** I client che dipendono dal nome non sono portabili.

**Restituisce:**
il nome dell'argomento.

**Produce:**
JMSException - se l'implementazione JMS per Topic non riesce a restituire il nome dell'argomento a causa di un errore interno.

**toString**

```
public String toString()
```

Restituire una versione stampata del nome del Topic.

**Restituisce:**
i valori dell'identità specifica del provider per questo Topic.

**Sostituisce:**
toString nella classe Object.

**getReference ***

```
public Reference getReference()
```

Creare un riferimento per l'argomento specificato.

**Restituisce:**
un riferimento per questo oggetto.

**Produce:**
NamingException.

**setBaseTopicName ***

```
public void setBaseTopicName(String x)
```

impostare il metodo per il nome dell'argomento MQSeries sottostante.

**getBaseTopicName ***

```
public String getBaseTopicName()
```

richiamare il metodo per il nome dell'argomento MQSeries sottostante.

**setBrokerDurSubQueue ***

```
public void setBrokerDurSubQueue(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerDurSubQueue.

**Parametri:**
brokerDurSubQueue - il nome della coda di sottoscrizione durevole da utilizzare.

**getBrokerDurSubQueue ***

```
public String getBrokerDurSubQueue()
```

Richiamare il metodo per l'attributo brokerDurSubQueue.

**Restituisce:**
il nome della coda di sottoscrizione durevole (brokerDurSubQueue) da utilizzare.

**setBrokerCCDurSubQueue ***

```
public void setBrokerCCDurSubQueue(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerCCDurSubQueue.

**Parametri:**
brokerCCDurSubQueue - il nome della coda di sottoscrizione durevole da utilizzare per una ConnectionConsumer.

**getBrokerCCDurSubQueue ***

```
public String getBrokerCCDurSubQueue()
```

Richiamare il metodo per l'attributo brokerCCDurSubQueue.

**Restituisce:**
il nome della coda di sottoscrizione durevole (brokerCCDurSubQueue) da utilizzare per una ConnectionConsumer.

# TopicConnection

interfaccia pubblica **TopicConnection**
estende **Connection**
Interfacce secondarie: **XATopicConnection**

Classe MQSeries: **MQTopicConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection           |
           +----com.ibm.mq.jms.MQTopicConnection
```

Una TopicConnection è una connessione attiva a un provider JMS Publish/Subscribe.

Vedere anche: **Connection**, **TopicConnectionFactory** e **XATopicConnection**

## Metodi

**createTopicSession**

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                       throws JMSException
```

Creare una TopicSession.

**Parametri:**

- transacted - se ″true″, la sessione viene negoziata.
- acknowledgeMode - uno dei seguenti valori:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Indica se il consumer o il client genereranno dei messaggi di avvenuta ricezione quando ricevono dei messaggi. Questo parametro verrà ignorato se la sessione è negoziata.

**Restituisce:**
una nuova sessione dell'argomento.

**Produce:**
JMSException - se una connessione JMS non riesce a creare una sessione a causa di un errore interno o di una mancanza di supporto per la transazione e la modalità di riconoscimento specifica.

**createConnectionConsumer**

```
public ConnectionConsumer createConnectionConsumer
                            (Topic topic,
                             java.lang.String messageSelector,
                             ServerSessionPool sessionPool,
                             int maxMessages)
                                      throws JMSException
```

Creare un consumer per questa connessione. Si tratta di una funzione esperta che non viene utilizzata dai client JMS regolari.

**Parametri:**

- topic - l'argomento a cui accedere.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi.
- sessionPool - il pool di sessioni del server da associare a questo consumer della connessione.
- maxMessages - il numero massimo di messaggi che è possibile assegnare a un server alla volta.

**Restituisce:**
il consumer della connessione.

**Produce:**

- JMSException - se una connessione JMS non riesce a creare un consumer della connessione a causa di un errore interno o di argomenti non validi per sessionPool.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

**Vedere anche:**
ConnectionConsumer

### createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer
                                (Topic topic,
                                 java.lang.String subscriptionName
                                 java.lang.String messageSelector,
                                 ServerSessionPool sessionPool,
                                 int maxMessages)
                                          throws JMSException
```

Creare un consumer di connessione durevole per questa connessione. Si tratta di una funzione esperta che non viene utilizzata dai client JMS regolari.

**Parametri:**

- topic - l'argomento a cui accedere.
- subscriptionName - nome della sottoscrizione durevole.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi.
- sessionPool - il pool di sessioni del server da associare a questo consumer della connessione durevole.
- maxMessages - il numero massimo di messaggi che è possibile assegnare a un server alla volta.

**Restituisce:**
il consumer della connessione durevole.

**Produce:**

- JMSException - se una connessione JMS non riesce a creare un consumer della connessione a causa di un errore interno o di argomenti non validi per sessionPool e messageSelector.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

**Vedere anche:**
ConnectionConsumer

# TopicConnectionFactory

interfaccia pubblica **TopicConnectionFactory**
estende **ConnectionFactory**
Interfacce secondarie: **XATopicConnectionFactory**

Classe MQSeries: **MQTopicConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory          |
          +----com.ibm.mq.jms.MQTopicConnectionFactory
```

Un client utilizza un TopicConnectionFactory per creare TopicConnections con un provider JMS Publish/Subscribe.

Vedere anche: **ConnectionFactory** e **XATopicConnectionFactory**

## Costruttori di MQSeries

**MQTopicConnectionFactory**

```
public MQTopicConnectionFactory()
```

## Metodi

**createTopicConnection**

```
public TopicConnection createTopicConnection()
                                        throws JMSException
```

Creare una connessione dell'argomento con l'identità utente predefinita. La connessione viene creata in modalità arrestata. Nessun messaggio verrà consegnato fino all'esplicito richiamo del metodo Connection.start.

**Restituisce:**
una nuova connessione all'argomento.

**Produce:**

- JMSException - se la connessione JMS non riesce a creare un Topic Connection a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

**createTopicConnection**

```
public TopicConnection createTopicConnection
                                        (java.lang.String userName,
                                        java.lang.String password)
                                        throws JMSException
```

Creare una connessione dell'argomento con l'identità utente specificata. La connessione viene creata in modalità arrestata. Nessun messaggio verrà consegnato fino all'esplicito richiamo del metodo Connection.start.

**Nota:** Questo metodo è valido solo per il tipo di trasporto IBM_JMS_TP_CLIENT_MQ_TCPIP. Vedere ConnectionFactory.

**Parametri:**

- userName - il nome utente di chi esegue la chiamata.
- password - la password di chi esegue la chiamata.

**Restituisce:**
una nuova connessione all'argomento.

**Produce:**

- JMSException - se la connessione JMS non riesce a creare un Topic Connection a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

**setBrokerControlQueue ***

```
public void setBrokerControlQueue(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerControlQueue.

**Parametri:**
brokerControlQueue - il nome della coda di controllo del broker.

**getBrokerControlQueue ***

```
public String getBrokerControlQueue()
```

Richiamare il metodo per l'attributo brokerControlQueue.

**Restituisce:**
il nome della coda di controllo del broker

**setBrokerQueueManager ***

```
public void setBrokerQueueManager(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerQueueManager.

**Parametri:**
brokerQueueManager - il nome del gestore code del broker.

**getBrokerQueueManager ***

```
public String getBrokerQueueManager()
```

Richiamare il metodo per l'attributo brokerQueueManager.

**Restituisce:**
il nome del gestore code del broker

**setBrokerPubQueue ***

```
public void setBrokerPubQueue(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerPubQueue.

**Parametri:**
brokerPubQueue - il nome della coda di pubblicazione del broker.

**getBrokerPubQueue ***

```
public String getBrokerPubQueue()
```

Richiamare il metodo per l'attributo brokerPubQueue.

**Restituisce:**
il nome della coda di pubblicazione del broker

**setBrokerSubQueue ***

```
public void setBrokerSubQueue(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerSubQueue.

**Parametri:**
brokerSubQueue - il nome della coda di sottoscrizione non durevole da utilizzare.

**getBrokerSubQueue ***

```
public String getBrokerSubQueue()
```

Richiamare il metodo per l'attributo brokerSubQueue.

**Restituisce:**
il nome della coda di sottoscrizione non durevole da utilizzare.

**setBrokerCCSubQueue ***

```
public void setBrokerCCSubQueue(String x) throws JMSException
```

Impostare il metodo per l'attributo brokerCCSubQueue.

**Parametri:**
brokerSubQueue - il nome della coda di sottoscrizione non durevole da utilizzare per una ConnectionConsumer.

**getBrokerCCSubQueue ***

```
public String getBrokerCCSubQueue()
```

Richiamare il metodo per l'attributo brokerCCSubQueue.

**Restituisce:**
il nome della coda di sottoscrizione non durevole da utilizzare per una ConnectionConsumer.

**setBrokerVersion ***

```
public void setBrokerVersion(int x) throws JMSException
```

Impostare il metodo per l'attributo brokerVersion.

**Parametri:**
brokerVersion - il numero di versione del broker.

**getBrokerVersion ***

```
public int getBrokerVersion()
```

Richiamare il metodo per l'attributo brokerVersion.

**Restituisce:**
il numero di versione del broker.

**getReference ***

```
public Reference getReference()
```

Restituire un riferimento per questo valore predefinito della connessione all'argomento.

**Restituisce:**
un riferimento per questo valore predefinito della connessione all'argomento.

**Produce:**
NamingException.

# TopicPublisher

interfaccia pubblica **TopicPublisher**
estende **MessageProducer**

Classe MQSeries: **MQTopicPublisher**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

Un client utilizza un TopicPublisher per pubblicare dei messaggi su un argomento. TopicPublisher è la variante Pub/Sub di un produttore di messaggi JMS.

## Metodi

**getTopic**

```
public Topic getTopic() throws JMSException
```

Richiamare l'argomento associato a questo editore.

**Restituisce:**
l'argomento di questo editore

**Produce:**
JMSException - se JMS non riesce a richiamare l'argomento per questo editore dell'argomento a causa di un errore interno.

**publish**

```
public void publish(Message message) throws JMSException
```

Pubblicare un messaggio sull'argomento. Utilizzare la modalità di consegna predefinita, la durata e la priorità dell'argomento.

**Parametri:**
message - il messaggio da pubblicare

**Produce:**

- JMSException - se JMS non riesce a pubblicare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con un Topic Publisher con un argomento non valido.

**publish**

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

Pubblicare un messaggio sull'argomento specificando la modalità di consegna, la priorità e la durata dell'argomento.

**Parametri:**

- message - il messaggio da pubblicare.
- deliveryMode - la modalità di consegna da utilizzare.
- priority - la priorità per questo messaggio.
- timeToLive - la durata del messaggio (in millisecondi).

**Produce:**

- JMSException - se JMS non riesce a pubblicare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con un Topic Publisher con un argomento non valido.

**publish**

```
public void publish(Topic topic,
                    Message message) throws JMSException
```

Pubblicare un messaggio su un argomento per un produttore di messaggi non identificato. Utilizzare la modalità di consegna predefinita, la durata e la priorità dell'argomento.

**Parametri:**

- topic - l'argomento su cui pubblicare questo messaggio.
- message - il messaggio da inviare.

**Produce:**

- JMSException - se JMS non riesce a pubblicare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con un argomento non valido.

**publish**

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

Pubblicare un messaggio su un argomento per un produttore di messaggi non identificati, specificando la modalità di consegna, la priorità e la durata.

**Parametri:**

- topic - l'argomento su cui pubblicare questo messaggio.
- message - il messaggio da inviare.
- deliveryMode - la modalità di consegna da utilizzare.
- priority - la priorità per questo messaggio.
- timeToLive - la durata del messaggio (in millisecondi).

**Produce:**

- JMSException - se JMS non riesce a pubblicare il messaggio a causa di un errore interno.
- MessageFormatException - se viene specificato un messaggio non valido.
- InvalidDestinationException - se un client utilizza questo metodo con un argomento non valido.

**close ***

```
public void close() throws JMSException
```

Poiché un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una TopicPublisher, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Produce:**

JMSException se JMS non riesce a chiudere il produttore a causa di un errore.

**Sostituisce:**

close nella classe MQMessageProducer.

# TopicRequestor

classe pubblica **TopicRequestor**
estende **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.TopicRequestor
```

JMS fornisce questa classe TopicRequestor che consente di effettuare richieste di servizi.

Al costruttore TopicRequestor viene fornita una TopicSession non negoziata e un Topic di destinazione. Crea una TemporaryTopic per le risposte e fornisce un metodo request() che invia il messaggio di richiesta e attende la risposta. Gli utenti possono comunque creare risposte più sofisticate.

## Costruttori

**TopicRequestor**

```
public TopicRequestor(TopicSession session,
                      Topic topic) throws JMSException
```

Costruttore per la classe TopicRequestor. Questa implementazione presuppone che il parametro della sessione non sia negoziato e che sia AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE.

**Parametri:**

- session - la sessione dell'argomento a cui appartiene l'argomento.
- topic - l'argomento su cui eseguire la chiamata di richiesta/risposta.

**Produce:**
JMSException - se si verifica un errore JMS.

## Metodi

**request**

```
public Message request(Message message) throws JMSException
```

Inviare una richiesta e attendere una risposta.

**Parametri:**
message - il messaggio da inviare.

**Restituisce:**
il messaggio di risposta.

**Produce:**
JMSException - se si verifica un errore JMS.

**close**

```
public void close() throws JMSException
```

## TopicRequestor

Dal momento che un provider potrebbe assegnare alcune risorse al di fuori di JVM da parte di una TopicRequestor, i client dovrebbero chiuderle quando non sono necessarie. La funzione di Garbage Collection potrebbe infatti non eseguire questa operazione in modo abbastanza rapido.

**Nota:** Questo metodo chiude l'oggetto Session passato al costruttore TopicRequestor.

**Produce:**

JMSException - se si verifica un errore JMS.

# TopicSession

interfaccia pubblica **TopicSession**
estende **Session**

Classe MQSeries: **MQTopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

Una TopicSession fornisce metodi per la creazione di TopicPublisher, TopicSubscriber e TemporaryTopic.

Vedere anche: **Session**

## Costruttori di MQSeries

**MQTopicSession**

```
public MQTopicSession(boolean transacted,
                      int acknowledgeMode) throws JMSException
```

Vedere **TopicConnection.createTopicSession**.

## Metodi

**createTopic**

```
public Topic createTopic(java.lang.String topicName)
                                                  throws JMSException
```

Creare un Topic in base a un Nome argomento nel formato URI. (Il formato URI è descritto in "Creazione di argomenti in fase di runtime" a pagina 204.) In questo modo è consentita la creazione di un argomento con un nome specifico di provider.

**Nota:** I client che dipendono da questa capacità non sono portabili.

**Parametri:**
topicName - il nome di questo argomento.

**Restituisce:**
un Topic con il nome assegnato.

**Produce:**
JMSException - se una sessione non riesce a creare un argomento a causa di un errore JMS.

**createSubscriber**

```
public TopicSubscriber createSubscriber(Topic topic)
                                        throws JMSException
```

Creare un sottoscrittore non durevole all'argomento specificato.

**Parametri:**
topic - l'argomento a cui effettuare la sottoscrizione.

**Produce:**

- JMSException - se una sessione non riesce a creare un sottoscrittore a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Topic non valido.

**createSubscriber**

```
public TopicSubscriber createSubscriber
                            (Topic topic,
                             java.lang.String messageSelector,
                             boolean noLocal) throws JMSException
```

Creare un sottoscrittore non durevole all'argomento specificato.

**Parametri:**

- topic - l'argomento a cui effettuare la sottoscrizione.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi. Questo valore potrebbe essere null.
- noLocal - se impostato, blocca la consegna dei messaggi pubblicati dalla propria connessione.

**Produce:**

- JMSException - se una sessione non riesce a creare un sottoscrittore a causa di un errore JMS di un selettore non valido.
- InvalidDestinationException - se viene specificato un Topic non valido.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

**createDurableSubscriber**

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name) throws JMSException
```

Creare un sottoscrittore durevole all'argomento specificato. Un client può modificare una sottoscrizione durevole esistente creando un sottoscrittore durevole con lo stesso nome e un nuovo argomento e/o selettore dei messaggi.

**Parametri:**

- topic - l'argomento a cui effettuare la sottoscrizione.
- name - il nome utilizzato per identificare questa sottoscrizione.

**Produce:**

- JMSException - se una sessione non riesce a creare un sottoscrittore a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Topic non valido.

Vedere **TopicSession.unsubscribe**

**createDurableSubscriber**

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name,
                                     java.lang.String messageSelector,
                                     boolean noLocal)  throws JMSException
```

Creare un sottoscrittore durevole all'argomento specificato.

**Parametri:**

- topic - l'argomento a cui effettuare la sottoscrizione.
- name - il nome utilizzato per identificare questa sottoscrizione.
- messageSelector - vengono recapitati solo i messaggi con proprietà che corrispondono all'espressione del selettore dei messaggi. Questo valore potrebbe essere null.
- noLocal - se impostato, blocca la consegna dei messaggi pubblicati dalla propria connessione.

**Produce:**

- JMSException - se una sessione non riesce a creare un sottoscrittore a causa di un errore JMS di un selettore non valido.
- InvalidDestinationException - se viene specificato un Topic non valido.
- InvalidSelectorException - se il selettore dei messaggi non è valido.

**createPublisher**

```
public TopicPublisher createPublisher(Topic topic)
                                      throws JMSException
```

Creare un editore per l'argomento specificato.

**Parametri:**

topic - l'argomento in cui effettuare la pubblicazione o null se si tratta di un produttore non identificato.

**Produce:**

- JMSException - se una sessione non riesce a creare un editore a causa di un errore JMS.
- InvalidDestinationException - se viene specificato un Topic non valido.

**createTemporaryTopic**

```
public TemporaryTopic createTemporaryTopic()
                                       throws JMSException
```

Creare un argomento temporaneo. La durata corrisponderà a quella di TopicConnection a meno che non venga eliminato prima.

**Restituisce:**

un argomento temporaneo.

**Produce:**

JMSException - se una sessione non riesce a creare un argomento temporaneo a causa di un errore JMS.

**unsubscribe**

```
public void unsubscribe(java.lang.String name)
                                        throws JMSException
```

Annulla una sottoscrizione durevole creata da un client.

**Nota:** Non utilizzare questo metodo mentre esiste una sottoscrizione attiva. E' necessario prima chiudere (close()) il sottoscrittore.

**Parametri:**

name - il nome utilizzato per identificare questa sottoscrizione.

**Produce:**

- JMSException - se JMS non riesce ad annullare la sottoscrizione durevole a causa di un errore JMS interno.
- InvalidDestinationException - se viene specificato un Topic non valido.

# TopicSubscriber

interfaccia pubblica **TopicSubscriber**
estende **MessageConsumer**

Classe MQSeries: **MQTopicSubscriber**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

Un client utilizza una TopicSubscriber per la ricezione dei messaggi pubblicati in un argomento. TopicSubscriber è la variante Pub/Sub di un consumer di messaggi JMS.

Vedere anche: **MessageConsumer** e **TopicSession.createSubscriber**

MQTopicSubscriber eredita i seguenti metodi da MQMessageConsumer:
close
getMessageListener
receive
receiveNoWait
setMessageListener

## Metodi

**getTopic**

```
public Topic getTopic() throws JMSException
```

Richiamare l'argomento associato a questo sottoscrittore.

**Restituisce:**
l'argomento del sottoscrittore.

**Produce:**
JMSException - se JMS non riesce a richiamare l'argomento per questo sottoscrittore dell'argomento a causa di un errore interno.

**getNoLocal**

```
public boolean getNoLocal() throws JMSException
```

Richiamare l'attributo NoLocal per questo TopicSubscriber. Il valore predefinito per questo attributo è false.

**Restituisce:**
impostato su true se i messaggi pubblicati localmente sono bloccati.

**Produce:**
JMSException - se JMS non riesce a richiamare l'attributo NoLocal per questo sottoscrittore dell'argomento a causa di un errore interno.

# XAConnection

interfaccia pubblica **XAConnection**
Interfacce secondarie:
**XAQueueConnection** e **XATopicConnection**

Classe MQSeries: **MQXAConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnection
```

XAConnection estende la capacità di Connection fornendo una XASession. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **XAQueueConnection** e **XATopicConnection**

# XAConnectionFactory

interfaccia pubblica **XAConnectionFactory**
Interfacce secondarie: **XAQueueConnectionFactory** e **XATopicConnectionFactory**

Classe MQSeries: **MQXAConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnectionFactory
```

Alcuni server applicazioni forniscono il supporto per integrare l'utilizzo di risorse con capacità JTS in una transazione distribuita. Per includere transazioni JMS in una transazione JTS, un server applicazioni richiede un provider JMS che supporti JTS. Un provider JMS espone il proprio supporto JTS utilizzando una JMS XAConnectionFactory, che un server applicazioni utilizza per creare XASession. XAConnectionFactory sono oggetti gestiti da JMS proprio come ConnectionFactory. Si presuppone che i server applicazioni utilizzino JNDI per trovarli.

Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **XAQueueConnectionFactory** e **XATopicConnectionFactory**

# XAQueueConnection

interfaccia pubblica **XAQueueConnection**
estende **QueueConnection** e **XAConnection**

Classe MQSeries: **MQXAQueueConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection            |
            +----com.ibm.mq.jms.MQQueueConnection
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnection
```

XAQueueConnection fornisce le stesse opzioni creazione di QueueConnection. La sola differenza consiste nel fatto che, per definizione, una XAConnection è negoziata. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **XAConnection** e **QueueConnection**

## Metodi

**createXAQueueSession**

```
public XAQueueSession createXAQueueSession()
```

Creare una XAQueueSession.

**Produce:**
JMSException - se la connessione JMS non riesce a creare una sessione della coda XA a causa di un errore interno.

**createQueueSession**

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

Creare una QueueSession.

**Parametri:**

- transacted - se ″true″, la sessione viene negoziata.
- acknowledgeMode - indica se il consumer o il client riconosceranno i messaggi ricevuti. I valori possibili sono:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Questo parametro verrà ignorato se la sessione è negoziata.

**Restituisce:**
una nuova sessione della coda (notare che questa non è una coda della sessione XA).

**Produce:**
JMSException - se la connessione JMS non riesce a creare una sessione della coda a causa di un errore interno.

# XAQueueConnectionFactory

interfaccia pubblica **XAQueueConnectionFactory**
estende **QueueConnectionFactory** e **XAConnectionFactory**

Classe MQSeries: **MQXAQueueConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory            |
            +----com.ibm.mq.jms.MQQueueConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

Una XAQueueConnectionFactory fornisce le stesse opzioni di creazione di una QueueConnectionFactory. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **QueueConnectionFactory** e **XAConnectionFactory**

## Metodi

**createXAQueueConnection**

```
public XAQueueConnection createXAQueueConnection()
                                              throws JMSException
```

Creare una XAQueueConnection utilizzando l'identità utente predefinita. La connessione viene creata in modalità arrestata. Il recapito dei messaggi inizia solo dopo che è stata eseguita una chiamata esplicita del metodo Connection.start.

**Restituisce:**
Una nuova connessione della coda XA.

**Produce:**

- JMSException - se il provider JMS non riesce a creare una connessione della coda XA a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

**createXAQueueConnection**

```
public XAQueueConnection createXAQueueConnection
                                    (java.lang.String userName,
                                     java.lang.String password)
                                                    throws JMSException
```

Creare una connessione della coda XA utilizzando un'identità utente specifica. La connessione viene creata in modalità arrestata. Il recapito dei messaggi inizia solo dopo che è stata eseguita una chiamata esplicita del metodo Connection.start.

**Parametri:**

- userName - il nome utente di chi esegue la chiamata.
- password - la password di chi esegue la chiamata.

**Restituisce:**
una nuova connessione della coda XA.

**Produce:**

- JMSException - se il provider JMS non riesce a creare una connessione della coda XA a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

# XAQueueSession

interfaccia pubblica **XAQueueSession**
estende **XASession**

Classe MQSeries: **MQXAQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXAQueueSession
```

Una XAQueueSession fornisce una regolare QueueSession che può essere utilizzata per creare QueueReceivers, QueueSenders e QueueBrowsers. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

La XAResource che corrisponde alla QueueSession che può essere ottenuta richiamando il metodo getXAResource, che viene ereditato da XASession.

Vedere anche: **XASession**

## Metodi

**getQueueSession**

```
public QueueSession  getQueueSession()
                                throws JMSException
```

Richiamare la sessione della coda associata a questa XAQueueSession.

**Restituisce:**
l'oggetto della sessione della coda.

**Produce:**
JMSException - se si verifica un errore JMS.

# XASession

interfaccia pubblica **XASession**
estende **Session**
Interfacce secondarie: **XAQueueSession** e **XATopicSession**

Classe MQSeries: **MQXASession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
```

XASession estende le funzioni di Session aggiungendo l'accesso ad un supporto del provider JMS per JTA. Questo supporto si presenta come un oggetto javax.transaction.xa.XAResource. La funzionalità di quest'oggetto è molto simile a quella definita dall'interfaccia standard X/Open XA Resource.

Un server delle applicazioni controlla l'assegnazione transazionale di una XASession richiamandone la XAResource. Il server utilizza la XAResource per assegnare la sessione ad una transazione, preparare le attività da eseguire sulla transazione, eseguirne il commit e così via.

Una XAResource fornisce alcune funzioni abbastanza sofisticate come ad esempio la possibilità di eseguire operazioni di interleaf su più transazioni e di recuperare un elenco delle transazioni in corso.

Un provider JMS con supporto JTA deve implementare completamente questa funzionalità. A tale scopo può utilizzare i servizi di un database che supporta XA oppure implementare questa funzionalità partendo da zero.

A un client del server applicazioni viene fornito quella che sembra essere una regolare JMS Session. Il server applicazioni controlla quindi la gestione della transazione della sottostante XASession.

Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **XAQueueSession** e **XATopicSession**

## Metodi

**getXAResource**

```
public javax.transaction.xa.XAResource getXAResource()
```

Restituisce una risorsa XA al chiamante.

**Restituisce:**
una risorsa XA al chiamante.

**getTransacted**

```
public boolean getTransacted()
                       throws JMSException
```

Restituisce sempre true.

**Specificato da:**
getTransacted nell'interfaccia Session

**Restituisce:**
true - se la sessione è in modalità negoziata.

**Produce:**
JMSException - se JMS non riesce a restituire la modalità di transazione a causa di un errore interno nel JMS Provider.

**commit**

```
public void commit()
            throws JMSException
```

Questo metodo non dovrebbe essere richiamato per un oggetto XASession. Se invece viene richiamato, produce una TransactionInProgressException.

**Specificato da:**
commit nell'interfaccia Session.

**Produce:**
TransactionInProgressException - se questo metodo viene richiamato su una XASession.

**rollback**

```
public void rollback()
              throws JMSException
```

Questo metodo non dovrebbe essere richiamato per un oggetto XASession. Se invece viene richiamato, produce una TransactionInProgressException.

**Specificato da:**
rollback nell'interfaccia Session.

**Produce:**
TransactionInProgressException - se questo metodo viene richiamato su una XASession.

# XATopicConnection

interfaccia pubblica **XATopicConnection**
estende **TopicConnection** e **XAConnection**

Classe MQSeries: **MQXATopicConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection            |
            +----com.ibm.mq.jms.MQTopicConnection
                    |
                    +----com.ibm.mq.jms.MQXATopicConnection
```

Una XATopicConnection fornisce le stesse opzioni di creazione di una TopicConnection. La sola differenza consiste nel fatto che, per definizione, una XAConnection è negoziata. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **TopicConnection** e **XAConnection**

## Metodi

**createXATopicSession**

```
public XATopicSession createXATopicSession()
                                      throws JMSException
```

Creare una XATopicSession.

**Produce:**
JMSException - se la connessione JMS non riesce a creare una XATopicSession a causa di un errore interno.

**createTopicSession**

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                 throws JMSException
```

Creare una TopicSession.

**Specificato da:**
createTopicSession in interface TopicConnection.

**Parametri:**

- transacted - se ″true″, la sessione viene negoziata.
- acknowledgeMode - uno dei seguenti valori:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Indica se il consumer o il client genereranno dei messaggi di avvenuta ricezione quando ricevono dei messaggi. Questo parametro verrà ignorato se la sessione è negoziata.

**Restituisce:**
una nuova TopicSession (notare che questa non è una XATopicSession).

**Produce:**

JMSException - se una connessione JMS non riesce a creare una TopicSession a causa di un errore interno.

# XATopicConnectionFactory

interfaccia pubblica **XATopicConnectionFactory**
estende **TopicConnectionFactory** e **XAConnectionFactory**

Classe MQSeries: **MQXATopicConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory           |
            +----com.ibm.mq.jms.MQTopicConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

Un'interfaccia XATopicConnectionFactory fornisce le stesse opzioni di creazione di un'interfaccia TopicConnectionFactory. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

Vedere anche: **TopicConnectionFactory** e **XAConnectionFactory**

## Metodi

**createXATopicConnection**

```
public XATopicConnection createXATopicConnection()
                                        throws JMSException
```

Creare una XATopicConnection utilizzando l'id utente predefinito. La connessione viene creata in modalità arrestata. Il recapito dei messaggi inizia solo dopo che è stata eseguita una chiamata esplicita del metodo Connection.start.

**Restituisce:**
Una nuova XATopicSession.

**Produce:**

- JMSException - se il provider JMS non riesce a creare una XATopicConnection a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

**createXATopicConnection**

```
public XATopicConnection createXATopicConnection(java.lang.String userName,
                                                 java.lang.String password)
                                        throws JMSException
```

Creare una XATopicConnection utilizzando l'id utente specificato. La connessione viene creata in modalità arrestata. Il recapito dei messaggi inizia solo dopo che è stata eseguita una chiamata esplicita del metodo Connection.start.

**Parametri:**

- userName - il nome utente di chi esegue la chiamata
- password - la password di chi esegue la chiamata

**Restituisce:**
Una nuova XATopicSession.

**Produce:**

- JMSException - se il provider JMS non riesce a creare una XATopicConnection a causa di un errore interno.
- JMSSecurityException - se l'autenticazione client non riesce a causa di un nome utente o di una password non validi.

# XATopicSession

interfaccia pubblica **XATopicSession**
estende **XASession**

Classe MQSeries: **MQXATopicSession**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQXASession
            |
            +----com.ibm.mq.jms.MQXATopicSession
```

Una XATopicSession fornisce una TopicSession, che può essere utilizzata per creare TopicSubscriber e TopicPublisher. Consultare l'"Appendice E. Interfaccia JMS JTA/XA con WebSphere" a pagina 401 per ulteriori informazioni sul modo in cui MQ JMS utilizza le classi XA.

La XAResource che corrisponde alla TopicSession può essere ottenuta richiamando il metodo getXAResource, ereditato dalla XASession.

Vedere anche **TopicSession** e **XASession**

## Metodi

### getTopicSession

```
public TopicSession getTopicSession()
                            throws JMSException
```

Richiamare la TopicSession associata a questa XATopicSession.

**Restituisce:**
L'oggetto TopicSession.

**Produce:**

- JMSException - se si verifica un errore JMS.

# Parte 4. Appendici

# Appendice A. Associazione tra le proprietà dello strumento di amministrazione e le proprietà programmabili

MQSeries Classes for Java Message Service fornisce delle funzioni per l'impostazione e l'interrogazione delle proprietà degli oggetti amministrati utilizzando lo strumento di amministrazione MQ JMS o in un programma applicativo. Nella Tabella 37 viene illustrata la corrispondenza tra il nome di ogni proprietà utilizzata con lo strumento di amministrazione e la corrispondente variabile del membro a cui fa riferimento. Viene mostrata inoltre l'associazione tra i valori delle proprietà simbolici utilizzati nello strumento e i rispettivi equivalenti programmabili.

*Tabella 37. Confronto delle rappresentazioni dei valori della proprietà all'interno dello strumento di amministrazione e dei programmi.*

| **Proprietà** | **Nome della variabile del membro** | **Valori della proprietà dello strumento** | **Valori della proprietà del programma** |
|---|---|---|---|
| DESCRIZIONE | descrizione | | |
| TRANSPORT | transportType | • BIND<br>• CLIENT | JMSC.MQJMS_TP_BINDINGS_MQ<br>JMSC.MQJMS_TP_CLIENT_MQ_TCPIP |
| CLIENTID | clientId | | |
| QMANAGER | queueManager* | | |
| HOSTNAME | hostName | | |
| PORT | port | | |
| CHANNEL | channel | | |
| CCSID | CCSID | | |
| RECEXIT | receiveExit | | |
| RECEXITINIT | receiveExitInit | | |
| SECEXIT | securityExit | | |
| SECEXITINIT | securityExitInit | | |
| SENDEXIT | sendExit | | |
| SENDEXITINIT | sendExitInit | | |
| TEMPMODEL | temporaryModel | | |
| MSGRETENTION | messageRetention | • YES<br>• NO | JMSC.MQJMS_MRET_YES<br>JMSC.MQJMS_MRET_NO |
| BROKERVER | brokerVersion | • V1 | JMSC.MQJMS_BROKER_V1 |
| BROKERPUBQ | brokerPubQueue | | |
| BROKERSUBQ | brokerSubQueue | | |
| BROKERDURSUBQ | brokerDurSubQueue | | |
| BROKERCCSUBQ | brokerCCSubQueue | | |
| BROKERCCDSUBQ | brokerCCDurSubQueue | | |
| BROKERQMGR | brokerQueueManager | | |
| BROKERCONQ | brokerControlQueue | | |

## Proprietà

*Tabella 37. Confronto delle rappresentazioni dei valori della proprietà all'interno dello strumento di amministrazione e dei programmi. (Continua)*

| **Proprietà** | **Nome della variabile del membro** | **Valori della proprietà dello strumento** | **Valori della proprietà del programma** |
|---|---|---|---|
| EXPIRY | expiry | • APP<br>• UNLIM | JMSC.MQJMS_EXP_APP<br>JMSC.MQJMS_EXP_UNLIMITED |
| PRIORITY | priority | • APP<br>• QDEF | JMSC.MQJMS_PRI_APP<br>JMSC.MQJMS_PRI_QDEF |
| PERSISTENCE | persistence | • APP<br>• QDEF<br>• PERS<br>• NON | JMSC.MQJMS_PER_APP<br>JMSC.MQJMS_PER_QDEF<br>JMSC.MQJMS_PER_PER<br>JMSC.MQJMS_PER_NON |
| TARGCLIENT | targetClient | • JMS<br>• MQ | JMSC.MQJMS_CLIENT_JMS_COMPLIANT<br>JMSC.MQJMS_CLIENT_NONJMS_MQ |
| ENCODING | encoding | | |
| QUEUE | baseQueueName | | |
| TOPIC | baseTopicName | | |
| **Nota:** * per un oggetto MQQueue, il nome della variabile del membro è baseQueueManagerName | | | |

# Appendice B. Script forniti con MQSeries classi per Java Message Service

I file che seguono sono forniti nella directory `bin` dell'installazione di MQ JMS. Questi script consentono di eseguire con maggiore facilità le attività più comuni relative all'installazione o all'utilizzo di MQ JMS. Nella Tabella 38 sono riportati gli script e il rispettivo utilizzo.

*Tabella 38. Programmi di utilità forniti con MQSeries classi per Java Message Service*

| **Programma di utilità** | **Uso** |
|---|---|
| IVTRun.bat<br>IVTTidy.bat<br>IVTSetup.bat | Utilizzato per eseguire il programma di verifica dell'installazione point-to-point, descritto nella sezione "Esecuzione del programma IVT point-to-point" a pagina 26. |
| PSIVTRun.bat | Utilizzato per eseguire il programma di verifica dell'installazione Pub/Sub descritto nella sezione "Il programma ITV (Installation Verification Test) Publish/Subscribe" a pagina 30. |
| formatLog.bat | Utilizzato per convertire i file di registrazione binari in solo testo, come viene descritto nella sezione "Registrazione" a pagina 34. |
| JMSAdmin.bat | Utilizzato per eseguire lo strumento di amministrazione, descritto nella sezione "Capitolo 5. Utilizzo dello strumento di amministrazione di MQ JMS" a pagina 35. |
| JMSAdmin.config | File di configurazione per lo strumento di amministrazione, descritto nella sezione "Configurazione" a pagina 36. |
| runjms.bat | Uno script del programma di utilità che consente di eseguire con più facilità le applicazioni JMS, come viene descritto nella sezione "Esecuzione dei programmi MQ JMS" a pagina 33. |
| PSReportDump.class | Utilizzato per visualizzare i messaggi di prospetto del broker, come viene descritto nella sezione "Gestione dei prospetti del broker" a pagina 212. |
| **Nota:** Sui sistemi UNIX, l'estensione ".bat" viene omessa dai nomi dei file. | |

## Script

# Appendice C. Definizione dello schema LDAP per la memorizzazione degli oggetti Java

Questa appendice fornisce dettagli relativi alle definizioni dello schema (definizioni dell'objectClass e dell'attributo) necessari in una directory LDAP per memorizzare gli oggetti Java. Tale appendice è rivolta agli utenti che desiderano utilizzare un server LDAP come provider di servizi JNDI, in cui memorizzare gli oggetti amministrati da MQ JMS. E' necessario assicurarsi che lo schema del server LDAP contenga le seguenti definizioni; la procedura esatta per ottenerle varia da server a server. La modalità per apportare modifiche ad alcuni specifici server LDAP è trattata successivamente in questa sezione.

La maggior parte dei dati contenuti in questa appendice sono stati presi da RFC 2713 *Schema per la rappresentazione degli oggetti Java in una directory LDAP*, che è possibile trovare all'indirizzo http://www.faqs.org/rfcs/rfc2713.html. Informazioni specifiche sul server LDAP sono state prese dal provider di servizi LDAP JNDI 1.2.1 di Sun Microsystems, disponibile all'indirizzo http://java.sun.com/products/jndi.

## Verifica della configurazione del server LDAP

Per verificare se il server LDAP è già configurato per accettare gli oggetti Java, eseguire lo MQ JMS Strumento di amministrazione JMSAdmin sul server LDAP (consultare "Richiamo dello strumento di amministrazione" a pagina 35).

Tentare di creare e visualizzare un oggetto di prova utilizzando i seguenti comandi:

```
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

Se non si verifica alcun errore, il server è correttamente configurato per memorizzare gli oggetti Java ed è possibile procedere alla memorizzazione degli oggetti JMS. Tuttavia, se il server LDAP contiene definizioni dello schema precedenti (per esempio, da una precedente bozza di RFC 2713 come le specificazioni, divenute adesso obsolete, di "draft-ryan-java-schema-00" e "draft-ryan-java-schema-01"), è possibile aggiornarle con quelle qui presentate.

Se si presenta uno SchemaViolationException o viene restituito il messaggio "Impossibile associare all'oggetto", il server non è configurato correttamente. Il server non è configurato per memorizzare gli oggettiJava, le autorizzazioni sugli oggetti non sono corrette, il suffisso o il contesto fornito non è stato impostato. Le seguenti informazioni dovrebbero essere di aiuto nella parte di configurazione dello schema di installazione del server.

# Definizioni dell’attributo

*Tabella 39. Impostazioni dell’attributo per javaCodebase*

| Attributo | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.1.7 |
| Sintassi | Stringa IA5 (1.3.6.1.4.1.1466.115.121.1.26) |
| Massima lunghezza | 2048 |
| Valore singolo/multivalore | Multivalore |
| Utente modificabile? | Sì |
| Regole corrispondenti | caseExactIA5Match |
| Classe di accesso | normale |
| Utilizzo | Applicazioniutente |
| Descrizione | URL che specifica la posizione della definizione della classe |

*Tabella 40. Impostazioni dell’attributo per javaClassName*

| Attributo | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.1.6 |
| Sintassi | Stringa della directory (1.3.6.1.4.1.1466.115.121.1.15) |
| Massima lunghezza | 2048 |
| Valore singolo/multivalore | Valore singolo |
| Utente modificabile? | Sì |
| Regole corrispondenti | caseExactMatch |
| Classe di accesso | normale |
| Utilizzo | Applicazioniutente |
| Descrizione | Nome completo della classe o dell’interfaccia Java distinta |

*Tabella 41. Impostazioni dell’attributo per javaClassNames*

| Attributo | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.1.13 |
| Sintassi | Stringa della directory (1.3.6.1.4.1.1466.115.121.1.15) |
| Massima lunghezza | 2048 |
| Valore singolo/multivalore | Multivalore |
| Utente modificabile? | Sì |
| Regole corrispondenti | caseExactMatch |
| Classe di accesso | normale |
| Utilizzo | Applicazioniutente |
| Descrizione | Nome interfaccia o classe completo di Java |

*Tabella 42. Impostazioni dell'attributo per javaFactory*

| Attributo | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.1.10 |
| Sintassi | Stringa della directory (1.3.6.1.4.1.1466.115.121.1.15) |
| Massima lunghezza | 2048 |
| Valore singolo/multivalore | Valore singolo |
| Utente modificabile? | Sì |
| Regole corrispondenti | caseExactMatch |
| Classe di accesso | normale |
| Utilizzo | Applicazioniutente |
| Descrizione | Nome classe Java completo di un oggetto JNDI |

*Tabella 43. Impostazioni dell'attributo per javaReferenceAddress*

| Attributo | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.1.11 |
| Sintassi | Stringa della directory (1.3.6.1.4.1.1466.115.121.1.15) |
| Massima lunghezza | 2048 |
| Valore singolo/multivalore | Multivalore |
| Utente modificabile? | Sì |
| Regole corrispondenti | caseExactMatch |
| Classe di accesso | normale |
| Utilizzo | Applicazioniutente |
| Descrizione | Indirizzi associati ad un riferimento JNDI |

*Tabella 44. Impostazioni dell'attributo per javaSerializedData*

| Attributo | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.1.8 |
| Sintassi | Stringa di ottetti (1.3.6.1.4.1.1466.115.121.1.40) |
| Valore singolo/multivalore | Valore singolo |
| Utente modificabile? | Sì |
| Classe di accesso | normale |
| Utilizzo | Applicazioniutente |
| Descrizione | Serializzato da un oggetto Java |

## Definizioni dell'objectClass

*Tabella 45. definizione dell'objectClass per javaSerializedObject*

| Definizione | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.2.5 |
| Estensione/superiore | javaObject |
| Tipo | AUXILIARY |
| Attributi richiesti (necessari) | javaSerializedData |

*Tabella 46. definizione dell'objectClass per javaObject*

| Definizione | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.2.4 |
| Estensione/superiore | top |
| Tipo | ABSTRACT |
| Attributi richiesti (necessari) | javaClassName |
| Attributi facoltativi (facoltativi) | javaClassNames javaCodebase javaDoc descrizione |

*Tabella 47. definizione dell'objectClass per javaContainer*

| Definizione | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.2.1 |
| Estensione/superiore | top |
| Tipo | STRUCTURAL |
| Attributi richiesti (necessari) | cn |

*Tabella 48. definizione dell'objectClass per javaNamingReference*

| Definizione | Valore |
|---|---|
| OID (Object Identifier) | 1.3.6.1.4.1.42.2.27.4.2.7 |
| Estensione/superiore | javaObject |
| Tipo | AUXILIARY |
| Attributi facoltativi (facoltativi) | javaReferenceAddress javaFactory |

# Dettagli sulla configurazione di uno specifico server

## Netscape Directory (4.1 e precedente)

Questo livello di Netscape Directory non supporta la sintassi della Stringa di ottetti, così è necessario utilizzare la sintassi Binaria (1.3.6.1.4.1.1466.115.121.1.5) al suo posto. Inoltre Netscape Directory 4.1 ha problemi ad analizzare una definizione della classe dell'oggetto che contiene una clausola MUST senza le parentesi. La soluzione alternativa è quella di aggiungere un valore superfluo (objectClass) a ciascuna clausola MUST.

Ancora, è possibile utilizzare le applicazioni di modifica dello schema fornite da Sun descritte in "Applicazioni di modifica dello schema di Sun Microsystems" a pagina 397.

## Microsoft Active Directory

All'interno della Active Directory, soltanto i nomi delle classi strutturali (e non delle classi ausiliarie) possono apparire nell'attributo della classe dell'oggetto di una voce. Perciò, le classi astratte ed ausiliarie nella definizione dello schema Java devono essere definite nuovamente come strutturali. Ne consegue che:

- la classe javaObject può essere ereditata da javaContainer

- le classi javaNamingReference e javaSerializedObject possono essere ereditate da javaObject

Invece di apportare tali modifiche manualmente, è possibile utilizzare le applicazioni di modifica dello schema fornite da Sun descritte in "Applicazioni di modifica dello schema di Sun Microsystems".

## Applicazioni di modifica dello schema di Sun Microsystems

E' possibile utilizzare lo strumento di amministrazione del server LDAP(per esempio, lo Strumento di gestione della Directory per la directory SecureWay dell'IBM) per verificare o aggiungere le definizioni descritte precedentemente. In alternativa, il provider dei servizi LDAP JNDI 1.2.1 di Sun Microsystems (disponibile all'indirizzo http://java.sun.com/products/jndi) contiene applicazioni Java (CreateJavaSchema.java e UpdateJavaSchema.java) che aggiungono o aggiornano le definizioni dello schema richieste automaticamente. Queste applicazioni contengono soluzioni alternative per errori dello schema e comportamento di server specifico sia in Netscape Directory Server (pre-4.1 e 4.1) che in Microsoft Windows 2000 Active Directory.

Queste applicazioni non sono fornite con le classi di MQSeries per Java Message Service. Dettagli sulla loro esecuzione possono essere trovati nel README e nell'origine dell'applicazione contenuta nel provider dei servizi LDAP JNDI 1.2.1 di Sun scaricato.

## Modifica dello schema iSeries OS/400 V4R5

E' possibile utilizzare lo strumento di amministrazione del server LDAP(lo Strumento di gestione della Directory per la directory SecureWay dell'IBM) per verificare o aggiungere le definizioni descritte precedentemente.

Il server LDAP di OS/400 V4R5 è inviato con una versione non aggiornata dello schema RFC 2713 per gli oggetti java. Tale schema deve essere aggiornato allo schema come descritto precedentemente, per una corretta operazione con JMSAdmin. La modifica dello schema richiede definizioni non aggiornate e qualsiasi utilizzo di queste definizioni deve essere eliminato prima di aggiungere le definizioni corrette.

OS/400 V5R1 è inviato con la versione corrente di RFC 2713 e non richiede tali modifiche.

# Appendice D. Connessione a MQSeries Integrator V2

E' possibile utilizzare MQSeries Integrator V2:

- come broker publish/subscribe per MQ JMS
- per instradare o trasformare i messaggi creati da un'applicazione client JMS e per inviare o pubblicare messaggi in un client JMS

## Publish/subscribe

E' possibile utilizzare MQSeries Integrator V2 come broker publish/subscribe per MQ JMS. Tale operazione richiede le seguenti attività di configurazione:

- Base MQSeries

  Innanzitutto è necessario creare una coda di pubblicazione del broker. Tale coda è una coda MQSeries sul gestore code del broker e viene utilizzata per sottoporre le pubblicazioni al broker. E' possibile scegliere il proprio nome per questa coda ma deve corrispondere al nome della coda nella proprietà BROKERPUBQ di TopicConnectionFactory. In base all'impostazione predefinita, una proprietà BROKERPUBQ di TopicConnectionFactory è impostata sul valore SYSTEM.BROKER.DEFAULT.STREAM. Pertanto, a meno che non si desideri configurare un diverso nome in TopicConnectionFactory, è necessario assegnare alla coda il nome SYSTEM.BROKER.DEFAULT.STREAM.

- MQSeries Integrator V2

  Il passaggio successivo consiste nell'impostare un *flusso di messaggi* all'interno di un gruppo di esecuzione per il broker. La finalità di questo flusso di messaggi è la lettura dei messaggi dalla coda di pubblicazione del broker. Se lo si desidera, è possibile impostare più code di pubblicazione. Ciascuna di esse necessiterà della propria TopicConnectionFactory e del proprio flusso di messaggi.

  Il flusso di messaggi di base è composto da un nodo MQInput (configurato per leggere dalla coda SYSTEM.BROKER.DEFAULT.STREAM) il cui output è connesso all'input di un nodo Publication (o MQOutput).

  Il diagramma del flusso dei messaggi avrà quindi un aspetto simile al seguente:


*Figura 7. Flusso di messaggi di MQSeries Integrator*

Quando il flusso dei messaggi viene distribuito ed il broker viene avviato, dalla prospettiva ' dell'applicazione JMS il broker MQSeries Integrator V2 si comporta come un broker MQSeries Publish/Subscribe. Lo stato della sottoscrizione corrente può essere visualizzato utilizzando il Control Center MQSeries Integrator.

**Note:**

1. Nessuna modifica viene richiesta alle classi MQSeries per Java Message Service.
2. I broker MQSeries Publish/Subscribe e MQSeries Integrator V2 non possono coesistere sullo stesso gestore code.

3. La procedura di installazione e configurazione di MQSeries Integrator V2 viene descritta in tutti i dettagli nella Guida all'installazione *MQSeries Integrator per Windows NT Versione 2.0*.

## Trasformazione e routing

E' possibile utilizzare MQSeries Integrator V2 per effettuare il routing o trasformare i messaggi creati da un'applicazione client JMS e per inviare o pubblicare messaggi in un client JMS.

L'implementazione JMS di MQSeries utilizza la cartella mcd della MQRFH2 per veicolare le informazioni sul messaggio, come descritto in "L'intestazione MQRFH2" a pagina 219. In base all'impostazione predefinita, la proprietà Message Domain (Msd) viene utilizzata per identificare se il messaggio è di testo, byte, flusso, map o oggetto.L'impostazione di questo valore dipende dal tipo di messaggio JMS.

Se l'applicazione chiama setJMSType, può impostare il campo tipo mcd ad un valore di sua scelta. Questo campo tipo può essere letto dal flusso dei messaggi Integrator MQSeries ed un'applicazione di ricezione JMS può utilizzare il metodo getJMSType per richiamare il valore. Tale operazione si riferisce a tutti i tipi di messaggio JMS.

Quando un'applicazione JMS crea un messaggio di testo o di byte, può impostare esplicitamente i campi della cartella mcd richiamando il metodo setJMSType e passando in un argomento della stringa in un formato speciale URI come segue:

`mcd://`*`dominio`*`/[`*`imposta`*`]/[`*`tipo`*`][?format=`*`fmt`*`]`

Questo modulo URI consente ad un'applicazione di impostare mcd in un dominio, vale a dire non uno dei valori standard "jms_xxxx"; per esempio, al sominio "mrm". Inoltre, consente all'applicazione di impostare alcune impostazioni mcd e se si desidera campi di tipo e di formato.

L'argomento stringa setJMSType viene interpretato come segue:

1. Se la stringa non appare nel formato URI speciale (i.e. non si avvia con "mcd://"), viene aggiunta nella cartella mcd come il campo `tipo`.
2. Se la stringa non si avvia con "mcd://" e corrisponde al formato URI *ed* il messaggio è un messaggio di Testo o di Byte, la stringa URI viene divisa nelle sue parti costituenti. La parte del dominio sovrappone il valore jms_text o il valore jms_bytes che è stato generato diversamente e le parti rimanenti (se presenti) vengono utilizzate per impostare i campi impostazione, tipo e formato in mcd. I campi impostazione, tipo e formato sono tutti facoltativi.
3. Se la stringa si avvia con "mcd://" ed il messaggio è un Map, un Flusso o un messaggio Oggetto, la chiamata setJMSType lancia un'eccezione. Coscché non è possibile sovrapporre il dominio o fornire un'impostazione o uin formato per queste classi di messaggio, ma se si desidera è possibile fornirne un tipo.

Quando un messaggio MQ viene ricevuto con un dominio Msd diverso da un valore standard "jms_xxxx", viene istanziato come un testo JMS o come un messaggio byte e gli viene assegnato URI-style JMSType. L'applicazione di ricezione può leggere questo messaggio utilizzando il metodo getJMSType.

# Appendice E. Interfaccia JMS JTA/XA con WebSphere

MQSeries classi per Java Message Service include le interfacce JMS XA che consentono a MQ JMS di partecipare a un commit a due fasi coordinato da un gestore delle transazioni conformi al JTA (Java Transaction API).

In questa sezione verrà descritto come utilizzare queste funzioni con il WebSphere Application Server, Advanced Edition, in modo che WebSphere possa coordinare le operazioni di invio e ricezione JMS e gli aggiornamenti del database in una transazione globale.

**Note:**

1. Prima di utilizzare MQ JMS e le classi XA con WebSphere, potrebbe essere necessario eseguire ulteriori operazioni di installazione o configurazione. Fare riferimento al file `Readme.txt` nella pagina Web MQSeries Using Java SupportPac per ottenere informazioni più aggiornate (`www.ibm.com/software/ts/mqseries/txppacs/ma88.html`).
2. Le funzioni descritte in questa pagina non sono supportate da MQ Java per iSeries & AS/400.

## Utilizzo dell'interfaccia JMS con WebSphere

In questa sezione verranno fornite delle indicazioni sull'utilizzo dell'interfaccia JMS con il WebSphere Application Server, Advanced Edition.

Si presuppone una conoscenza delle funzioni principali dei programmi JMS, MQSeries e dei bean EJB. Questi dettagli sono nella specifica JMS e in quella EJB V2 (entrambe disponibili presso la Sun), nella presente pubblicazione, negli esempi forniti con MQ JMS e in altre pubblicazioni per MQSeries e WebSphere.

### Oggetti amministrati

JMS utilizza gli oggetti amministrati per incapsulare informazioni specifiche del fornitore. In questo modo l'impatto dei dettagli specifici del fornitore viene ridotto al minimo sulle applicazioni dell'utente finale. Gli oggetti amministrati vengono memorizzati in uno spazio dei nomi JNDI e possono essere richiamati e utilizzati in modo portabile senza una conoscenza del contenuto specifico del fornitore.

Per un utilizzo standalone, MQ JMS fornisce le seguenti classi:

- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere fornisce un'ulteriore coppia di oggetti amministrati in modo che MQ JMS possa integrarsi con WebSphere:

- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

Questi oggetti vanno utilizzati esattamente come MQQueueConnectionFactory e MQTopicConnectionFactory. Tuttavia essi utilizzano implicitamente le versioni XA delle classi JMS e includono MQ XAResource nella transazione WebSphere.

## Confronto tra le transazioni gestite dai container e le transazioni gestite dai bean

Le transazioni gestite dai container sono transazioni in bean EJB demarcati automaticamente dal container EJB. Le transazioni gestite dai bean sono transazioni in bean EJB demarcati dal programma tramite l'interfaccia UserTransaction.

## Confronto tra il commit a due fasi e l'ottimizzazione a una fase

Solo il coordinatore WebSphere richiama un commit a due fasi se in una determinata transazione viene utilizzata più di una XAResource. Delle transazioni che coinvolgono una sola risorsa viene eseguito un commit con un'ottimizzazione a una fase. In questo modo la necessità di utilizzare diversi ConnectionFactory per transazioni distribuite e non distribuite si riduce notevolmente.

## Definizione di oggetti amministrati

E' possibile utilizzare lo strumento di amministrazione MQ JMS per definire le impostazioni predefinite delle connessioni specifiche di WebSphere e memorizzarle in uno spazio dei nomi JNDI. Il file admin.config nel percorso *MQ_install_dir*/bin dovrebbe contenere le seguenti righe:

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
PROVIDER_URL=iiop://nomehost/
```

*MQ_install_dir* è la directory di installazione per MQ JMS e *nomehost* è il nome o l'indirizzo IP del computer su cui è in esecuzione WebSphere.

Per accedere a com.ibm.ejs.ns.jndi.CNInitialContextFactory, è necessario aggiungere il file ejs.jar dalla directory lib WebSphere a CLASSPATH.

Per creare le nuove impostazioni predefinite, utilizzare il verb di definizione con i seguenti due nuovi tipi:

```
def WSQCF(nome) [properties]
def WSTCF(nome) [properties]
```

Questi nuovi tipi utilizzano le stesse proprietà dei tipi equivalenti QCF o TCF, ad eccezione del fatto che è consentito solo il tipo di trasporto BIND. Pertanto le proprietà del client non possono essere configurate. Per ulteriori dettagli, consultare la sezione "Amministrazione degli oggetti JMS" a pagina 40.

## Richiamo degli oggetti di amministrazione

In un bean EJB, gli oggetti amministrati da JMS vengono richiamati utilizzando il metodo InitialContext.lookup(), ad esempio:

```
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

Degli oggetti può essere eseguito il casting e possono essere utilizzati come interfacce JMS generiche. Generalmente non è necessario programmare su classi specifiche MQSeries nl codice dell'applicazione.

# Esempi

Esistono tre esempi che illustrano gli elementi fondamentali dell'utilizzo di MQ JMS con WebSphere Application Server Advanced Edition. Tali esempi si trovano nelle sottodirectory di *MQ_install_dir*/samples/ws, dove *MQ_install_dir* è la directory di installazione per MQ JMS.

- L'esempio 1 (Sample1) dimostra una semplice operazione di put e get per un messaggio in una coda utilizzando le transazioni gestite dai container.
- L'esempio 2 (Sample2) dimostra una semplice operazione di put e get per un messaggio in una coda utilizzando le transazioni gestite dai bean.
- L'esempio 3 (Sample3) illustra l'utilizzo dell'API di tipo publish/subscribe.

Per informazioni dettagliate su come creare e distribuire i bean EJB, fare riferimento alla documentazione WebSphere Application Server.

I file readme.txt in ogni directory di esempio include un output di esempio da ciascun bean EJB. Gli script forniti presuppongono che un gestore code predefinito sia disponibile sulla macchina locale. Se l'installazione è diversa rispetto all'impostazione predefinita, è possibile modificare gli script in base alle esigenze specifiche.

## Esempio 1

Sample1EJB.java, nella directory sample1, definisce due metodi che utilizzano JMS:

- putMessage() invia un TextMessage a una coda e restituisce il MessageID del messaggio inviato
- getMessage() legge il messaggio con il MessageID specificato nuovamente nella coda

Prima di eseguire l'esempio, è necessario memorizzare due oggetti amministrati nello spazio dei nomi JNDI WebSphere:

**QCF1** un'impostazione predefinita della connessione della coda specifica di WebSphere

**Q1** una coda

Entrambi gli oggetti devono essere collegati nel subcontesto jms/Samples.

Per impostare gli oggetti amministrati, è possibile utilizzare lo strumento di amministrazione MQ JMS e impostarli manualmente oppure è possibile utilizzare lo script fornito.

Lo strumento di amministrazione MQ JMS deve essere configurato per accedere allo spazio dei nomi WebSphere. Per informazioni dettagliate su come configurare lo strumento di amministrazione, fare riferimento alla sezione "Configurazione per WebSphere" a pagina 37.

Per impostare gli oggetti amministrati con le impostazioni predefinite tipiche, è possibile immettere il seguente comando per eseguire lo script `admin.scp`:

```
JMSAdmin < admin.scp
```

Il bean deve essere distribuito con i metodi getMessage e putMessage contrassegnati come TX_REQUIRED. Questa operazione garantisce che il container avvii una transazione prima di immettere ciascun metodo ed esegue il commit della transazione quando il metodo viene completato. All'interno dei metodi non è

necessario utilizzare alcun codice dell'applicazione relativo allo stato della transazione. Tuttavia è importante ricordare che il messaggio inviato da putMessage si verifica durante il syncpoint e non sarà disponibile fino al commit della transazione.

Nella directory sample1, è disponibile il semplice programma client, Sample1Client.java, per chiamare il bean EJB. Viene fornito anche uno script, runClient, che semplifica l'esecuzione di questo programma.

Il programma client (o script) assume un singolo parametro, che viene utilizzato come corpo di un TextMessage che verrà inviato dal metodo putMessage del bean EJB. Quindi viene richiamato getMessage per rileggere il messaggio al di fuori della coda e restituire il corpo al client per la visualizzazione. Il bean EJB invia messaggio sull'avanzamento all'output standard (stdout) del server applicazioni, pertanto è possibile controllare quell'output durante l'esecuzione.

Se il server applicazioni si trova sulla macchina che è remota rispetto al client, potrebbe essere necessario modificare Sample1Client.java. Se non si utilizzano le impostazioni predefinite, è necessario modificare lo script runClient per mettere in corrispondenza il percorso di installazione locale e il nome del file jar distribuito.

## Esempio 2

Sample2EJB.java, nella directory sample2, esegue la stessa attività di sample1 e richiede gli stessi oggetti amministrati. A differenza di sample1, sample2 utilizza transazioni gestite dai bean per il controllo dei confini della transazione.

Se non è stato già eseguito sample1, assicurarsi di aver impostato gli oggetti amministrati et up QCF1 e Q1, come viene illustrato in "Esempio 1" a pagina 403.

I putMessage e getMessage si avviano ottenendo un'istanza di UserTransaction. Utilizzano quest'istanza per creare una transazione tramite il metodo UserTransaction.begin(). Dopodiché, il corpo principale del codice è lo stesso di sample1 fino alla fine di ciascun metodo. Alla fine di ciascun metodo, infatti, la transazione viene completata dalla chiamata UserTransaction.commit().

Nella directory sample2, è disponibile il semplice programma client, Sample2Client.java, per chiamare il bean EJB. Viene fornito anche uno script, runClient, che semplifica l'esecuzione di questo programma. E' possibile utilizzarli così come viene descritto nella sezione "Esempio 1" a pagina 403.

## Esempio 3

Sample3EJB.java, nella directory sample3, dimostra l'utilizzo dell'API publish/subscribe con WebSphere. La pubblicazione di un messaggio è molto simile allo scenario del point to point. Tuttavia esistono delle differenze nella ricezione dei messaggi tramite un TopicSubscriber.

I programmi di tipo publish/subscribe utilizzano generalmente i sottoscrittori non durevoli che esistono solo per la durata delle sessioni di appartenenza (o per un periodo inferiore se il sottoscrittore viene chiuso esplicitamente). Inoltre possono ricevere messaggi dal broker solo durante quella durata.

Per convertire sample1 in publish/subscribe, è possibile sostituire il QueueSender in putMessage con un TopicPublisher e il QueueReceiver in getMessage con un TopicSubscriber non durevole. Tuttavia questa operazione non riuscirebbe perché,

quando il messaggio viene inviato, il broker non conoscerebbe i sottoscrittori dell'argomento. Pertanto il messaggio verrebbe cancellato.

La soluzione consiste nel creare un sottoscrittore durevole prima della pubblicazione del messaggio. I sottoscrittori durevoli permangono come end-point consegnabili oltre la durata della sessione. Il messaggio può essere quindi recuperato durante la chiamata a getMessage().

Il bean EJB include due metodi ulteriori:

- createSubscription crea una sottoscrizione durevole
- destroySubscription elimina una sottoscrizione durevole

Questi metodi, insieme a putMessage e getMessage, devono essere distribuiti con l'attributo TX_REQUIRED.

Prima di eseguire sample3, è necessario memorizzare due oggetti amministrati nello spazio dei nomi JNDI WebSphere:

TCF1

T1

Entrambi gli oggetti devono essere collegati nel subcontesto jms/Samples.

Per impostare gli oggetti amministrati, è possibile utilizzare lo strumento di amministrazione MQ JMS e impostarli manualmente oppure è possibile utilizzare uno script. Lo script `admin.scp` viene fornito nella directory sample3.

Lo strumento di amministrazione MQ JMS deve essere configurato per accedere allo spazio dei nomi WebSphere. Per informazioni dettagliate su come configurare lo strumento di amministrazione, fare riferimento alla sezione "Configurazione per WebSphere" a pagina 37.

Per impostare gli oggetti amministrati con le impostazioni predefinite tipiche, è possibile immettere il seguente comando per eseguire lo script `admin.scp`:

```
JMSAdmin < admin.scp
```

Se è stato già eseguito admin.scp per impostare degli oggetti per sample1 o sample2, verranno visualizzati dei messaggi di errore all'esecuzione di admin.scp per sample3. Ciò si verifica quando si tenta di creare i subcontesti jms e Samples. E' comunque possibile ignorare senza alcun problema questi messaggi.

Inoltre, prima di eseguire sample3, assicurarsi che il broker publish/subscribe MQSeries (SupportPac MA0C) sia installato e in esecuzione.

Nella directory sample3, è disponibile il semplice programma client, Sample3Client.java, per chiamare il bean EJB. Viene fornito anche uno script, runClient, che semplifica l'esecuzione di questo programma. E' possibile utilizzarli così come viene descritto nella sezione "Esempio 1" a pagina 403.

# Appendice F. Utilizzo di MQ Java nelle applet con Java 1.2 o successiva

Potrebbe essere necessario eseguire task aggiuntive per eseguire un'applet utilizzando classi MQ Java in una JVM (Java virtual machine) nel livello Java 1.2 o maggiore. Tale operazione potrebbe essere necessaria poiché le regole di protezione predefinite per le applet con le JVM a questi livelli potrebbero essere modificate per ridurre il rischio di danneggiamenti mediante classi malevole o che funzionano in maniera anomala.

Vi sono due metodi differenti che si possono impiegare:

1. Modificare le impostazioni di protezione sul browser e sulla JVM per consentire l'utilizzo dei pacchetti MQ Java.
2. Copiare le classi MQ Java nella stessa posizione dell'applet che si desidera eseguire.

## Modifica delle impostazioni di protezione del browser

Differenti errori possono risultare dal tentativo di eseguire la stessa applet in ambienti diversi; per esempio, in IBM VisualAge per Java, in appletviewer (fornito con molti Development Kits Java) o in un browser del web come Internet Explorer. Le differenze esistono tra impostazioni di protezione differenti in ciascun ambiente. E' possibile modificare il comportamento degli ambienti per consentire l'accesso di un'applet nelle classi necessarie che sono memorizzate nei file del pacchetto.

Nelle seguenti istruzioni, esempi, si suppone di utilizzare la piattaformaWindows. Su altre piattaforme, le istruzioni necessitano di lievi modifiche.

**Per IBM VisualAge per Java:**

E' necessario modificare il file java.policy trovato in `<vaj_install_dir>\ide\program\lib\security`, dove "`<vaj_install_dir>`" è la directory in cui è stato installato IBM VisualAge per Java.

Eseguire le seguenti modifiche nelle autorizzazioni:

1. Commentare la riga

   ```
   autorizzazione java.net.SocketPermission
   "localhost:1024-", "listen";
   ```

   e sostituirla con la seguente:

   ```
   autorizzazione
   java.net.SocketPermission "*", "accept, connect, listen, resolve";
   ```

2. Aggiungere le seguenti righe:

```
autorizzazione java.util.PropertyPermission "MQJMS_LOG_DIR", "read";
autorizzazione java.util.PropertyPermission "MQJMS_TRACE_DIR", "read";
autorizzazione java.util.PropertyPermission "MQJMS_TRACE_LEVEL", "read";
autorizzazione java.lang.RuntimePermission "loadLibrary.*";
```

**Note:**

1. Potrebbe essere necessario riavviare VisualAge per Java se si verifica il messaggio di errore "Errore Java sconosciuto" dopo aver ripetuto le analisi.
2. Assicurarsi che `<ma88_install_dir>\java\lib` sia nel percorso classe area di lavoro.

### Modifica della protezione del browser

**Per appletviewer:**

Trovare il file dei criteri di protezione per JDK ed apportare le stesse modifiche come per IBM VisualAge per Java. Per esempio, nel Developer Kit IBM per Windows, Java Technology Edition, Versione 1.3, il file java.policy è stato trovato nella directory `<jdk_install_dir>\jre\lib\security`, dove "`<jdk_install_dir>`" è la directory in cui è stato installato Developer Kit.

**Per un browser web:**

Per ottenere un comportamento coerente per le applet all'interno di browser web differenti utilizzare il Sun Java plug-in.

1. Installare Sun Java plug-in 1.3.01 o successiva.

   Da questo livello è supportato anche Netscape 6.
2. Apportare le stesse modifiche nel file java.policy file come sopra elencato.

   Il file dei criteri di protezione è stato trovato in `<java_plugin_install_dir>\lib\security`.
3. Assicurarsi che le tag applet HTML siano state modificate per eseguire con il plug-in.

   Effettuare il download ed eseguire Sun HTML Converter v1.3 per apportare le modifiche necessarie.

## Copia file di classe del pacchetto

Quando un programma Java viene eseguito nel contesto di un 'applet (ciò che accade quando viene eseguito appletviewer o viene utilizzato un browser web), per impostazione predefinita il programma Java possiede significanti restrizioni di protezione ad esso applicate. Una di queste restrizioni è che tutte le variabili dell'ambiente quando viene avviato l'applet vengono ignorate. Essa include CLASSPATH.

Di conseguenza, se non vengono apportate le modifiche descritte in "Modifica delle impostazioni di protezione del browser" a pagina 407, quando viene eseguita un'applet, ogni classe necessaria deve essere disponibile per il download dalla stessa posizione come lo stesso codice dell'applet.

Per ottenere ciò su un sistema Windows, eseguire i seguenti passaggi (necessari agli utenti che non utilizzano Windows per eseguire task analoghe):

1. Eseguire il download ed instalare WINZIP (http://www.winzip.com) o l'equivalente utilità decomprimente del file
2. Trovare i file che contengono MQ Java, altri pacchetti o classi necessari all'applet.

   Per esempio, le classi MQ base Java sono in un file chiamato `com.ibm.mq.jar` di solito si trova nella cartella `C:\Program Files\IBM\MQSeries\Java\lib`.
3. Utilizzare l'utilità decomprimente installata nel passaggio 1, per estrarre *tutti* i file nel file .jar nella cartella che contiene l'applet.

   Per gli esempi forniti con MQ Java, la cartella da utilizzare è `C:\Program Files\IBM\MQSeries\Java\samples\base`

   Ciò risulta in una struttura sottocartella `com\ibm` che verrà creata.
4. Eseguire l'applet.

# Appendice G. Informazioni per SupportPac MA1G

Questa appendice contiene informazioni rilevanti per gli utenti di SupportPac MA1G "MQSeries per le classi MVS/ESA – MQSeries per Java". MA1G fornisce il supporto per le MQSeries classi per Java dalle versioni di OS/390 non supportate da MA88. Inoltre fornisce il supporto per CICS ed HPG (High Performance Java).

Gli utenti che intendono utilizzare MQ base Java con CICS Transaction Server per OS/390 dovrebbero:

- Conoscere i concetti CICS (Customer Information Control System)
- Utilizzare CICS API (Application Programming Interface)Java
- Eseguire i programmi Java dall'interno del CICS

Gli utenti che desiderano utilizzare VisualAge per Java per sviluppare applicazioni OS/390 UNIX System Services High Performance Java (HPJ) devono avere dimestichezza con Enterprise Toolkit per OS/390 (fornito con VisualAge per Java Enterprise Edition per OS/390, Versione 2).

## Ambienti supportati da SupportPac MA1G

SupportPac MA1G fornisce il supporto per MQ base Java dai seguenti ambienti:

- OS/390 V2R6 o superiore
- Java per OS/390, V1.1.8 o superiore
- IBM MQSeries per MVS/ESA, versione 1.2 o superiore
- HPJ (High Performance Java)

Inoltre, SupportPac MA1G fornisce il supporto per CICS TS1.3 o superiore. Il supporto per HPJ in questo ambiente richiede OS/390 V2R9 o superiore.

SupportPac MA1G *non* fornisce supporto per JMS.

## Recupero ed installazione del SupportPac MA1G

SupportPac MA1G potrebbe essere recuperato dall'indirizzo del MQSeries sito web http://www.ibm.com/software/mqseries. Seguire i collegamenti per effettuare il "download", poi, "SupportPacs" per trovare il codice MQ Java.

La seguente procedura installa MQSeries classi per Java. La directory utilizzata per l'installazione necessita almeno di 2MB di spazio nella memoria. Di seguito, sostituire "/u/joe/mqm" con il nome del percorso della directory scelta :

1. Rimuovere qualsiasi installazione precedente di questo prodotto utilizzando i seguenti comandi nella shell OpenEdition :

   ```
   cd /u/joe
   chmod -fR 700 mqm
   rm -rf mqm
   mkdir mqm
   ```

2. Utilizzare la modalità binaria FTP, caricare il file ma1g.tar.Z dalla stazione di lavoro nella directory HFS /u/joe/mqm.
3. Mentre nella shell OpenEdition, modificare la directory di installazione /u/joe/mqm.

4. Decomprimere il file con il comando
   ```
   tar -xpozf ma1g.tar.Z
   ```
5. Impostare CLASSPATH e LIBPATH come descritto in “Variabili di ambiente” a pagina 13.

### Verifica dell’installazione utilizzando il programma di esempio

Per verificare l’installazione di MA1G da USS (Unix System Services), seguire le istruzioni in “Verifica con l’applicazione di esempio” a pagina 18.

Per verificare l’installazione di MA1G da CICS Transaction Server:

1. Definire il programma di applicazione di esempio (MQIVP) su CICS.
2. Definire una transazione per eseguire l’applicazione di esempio.
3. Inserire il nome del gestore code nel file utilizzato per l’immissione standard.
4. Eseguire la transazione.

L’output del programma viene inserito nei file utilizzati per l’output standard e degli errori.

Fare riferimento alla documentazione di CICS per ulteriori informazioni sull’esecuzione dei programmi Java e sull’impostazione dei file di input e output.

## Funzioni non fornite da SupportPac MA1G

SupportPac MA1G fornisce un sottoinsieme di funzioni disponibili per altre applicazioni MQ base Java. In particolare, non supporta le funzioni ConnectionPooling descritte in “Capitolo 7. Scrittura di programmi MQ base Java” a pagina 57. Le classi ed i metodi seguenti, non sono supportati:

- Classi ed interfacce
  - MQPoolServices
  - MQPoolServicesEvent
  - MQPoolToken
  - MQSimpleConnectionManager
  - MQPoolServicesEventListener
  - MQConnectionManager
  - ManagedConnection
  - ManagedConnectionFactory
  - ManagedConnectionMetaData
- Metodi
  - MQEnvironment.getDefaultConnectionManager()
  - MQEnvironment.setDefaultConnectionManager()
  - MQEnvironment.addConnectionPoolToken()
  - MQEnvironment.removeConnectionPoolToken()
  - I sei costruttori MQQueueManager che permettono ad un ConnectionManager o ad un MQConnectionManager di essere specificato.

Nel tentativo di utilizzare queste classi, queste interfacce o questi metodi possono risultare errori di compilazione o eccezioni run-time.

## Esecuzione di applicazioni MQ base Java su CICS Transaction Server per OS/390

Per eseguire un'applicazione Java come transazione su CICS, è necessario:

1. Definire l'applicazione e la transazione su CICS utilizzando la transazione CEDA fornita.
2. Assicurarsi che l'adattatore MQSeries CICS venga installato nel sistemaCICS. (Vedere *MQSeries for OS/390 System Management Guide* per informazioni dettagliate.)
3. Assicurarsi che l'ambiente JVM specificato nel parametro DHFJVM del JCL (Job Control Language) CICS include le voci CLASSPATH e LIBPATH appropriate.
4. Iniziare la transazione utilizzando uno qualsiasi dei normali processi.

Per ulteriori informazioni sull'esecuzione delle transazioni CICS Java, fare riferimento alla documentazione del sistema CICS.

## Restrizioni su CICS Transaction Server

Nel CICS Transaction Server per l'ambiente OS/390, soltanto il thread principale (il primo) può eseguire chiamate CICS o MQSeries. Pertanto non è possibile condividere oggetti MQQueueManager o MQQueue tra thread in questo ambiente oppure creare un nuovo MQQueueManager su un thread secondario.

Tabella 12 a pagina 82 elenca la restrizioni e le variazioni che si riferiscono al MQSeries classi per Java quando è in esecuzione su un Queue Manager MQSeries OS/390. Inoltre, quando è in esecuzione su CICS, i metodi di controllo di transazione su MQQueueManager non sono supportati. Invece di emettere le applicazioni MQQueueManager.commit() o MQQueueManager.backout(), si potrebbero utilizzare i metodi di sincronizzazione della task JCICS, Task.commit() e Task.rollback(). La classe Task è supportata da JCICS nel package com.ibm.cics.server.

# Appendice H. Informazioni particolari

Queste informazioni sono state sviluppate per i prodotti e servizi offerti negli Stati Uniti. E' possibile che negli altri paesi l'IBM non offra i prodotti, i servizi o le funzioni illustrati in questo documento. 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 dall'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. E' responsabilità dell'utente valutare e verificare la possibilità di utilizzare altri programmi e/o prodotti, fatta eccezione per quelli espressamente indicati dall'IBM.

L'IBM può avere brevetti o domande di brevetto in corso relativi a quanto trattato nella presente pubblicazione. La fornitura di questa pubblicazione non implica la concessione di alcuna licenza su di essi. Chi desiderasse ricevere informazioni relative alle licenze può rivolgersi per iscritto a:

Director of Commercial Relations IBM Europe
IBM Europe
Schoenaicher str. 220
D-7030 Boeblingen
Deutschland

**Il seguente paragrafo non è valido per il Regno Unito o per tutti i paesi le cui leggi nazionali siano in contrasto con le disposizioni in esso contenute:**
L'INTERNATIONAL BUSINESS MACHINES CORPORATION FORNISCE QUESTA PUBBLICAZIONE "NELLO STATO IN CUI SI TROVA", SENZA ALCUNA GARANZIA, ESPLICITA O IMPLICITA, IVI INCLUSE EVENTUALI GARANZIE DI COMMERCIABILITA' ED IDONEITA' AD UNO SCOPO PARTICOLARE. Alcune stati non consentono la rinuncia a garanzie esplicite o implicite in determinate transazioni; quindi la presente dichiarazione potrebbe essere non essere a voi applicabile.

Questa pubblicazione potrebbe contenere imprecisioni tecniche o errori tipografici. Le informazioni incluse in questo documento vengono modificate su base periodica; tali modifiche verranno incorporate nelle nuove edizioni della pubblicazione. L'IBM si riserva il diritto di apportare miglioramenti e/o modifiche al prodotto o al programma descritto in questa pubblicazione in qualsiasi momento e senza preavviso.

Tutti i riferimenti a siti Web non dell'IBM contenuti in questo documento sono forniti solo per consultazione. I materiali disponibile presso i siti Web non fanno parte di questo prodotto e l'utilizzo di questi è a discrezione dell'utente.

Tutti i commenti e i suggerimenti inviati potranno essere utilizzati liberamente dall'IBM e dalla Selfin e diventeranno esclusiva delle stesse.

### Informazioni particolari

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 United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

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 ad esso relativo sono forniti dall'IBM nel rispetto delle condizioni previste dalla licenza d'uso.

Le informazioni relative a prodotti non IBM sono state ottenute dai fornitori di tali prodotti. L'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.

## Marchi

I seguenti termini sono marchi della International Business Machines Corporation:

| | | |
|---|---|---|
| AIX | AS/400 | BookManager |
| CICS | IBM | IBMLink |
| Language Environment | MQSeries | MVS/ESA |
| OS/2 | OS/390 | OS/400 |
| SecureWay | SupportPac | System/390 |
| S/390 | VisualAge | VSE/ESA |
| WebSphere | | |

Java, HotJava, JDK e tutti i marchi basati su Java sono marchi della Sun Microsystems, Inc.

Microsoft, Windows e Windows NT sono marchi della Microsoft Corporation.

UNIX è un marchio della Open Group.

Altri nomi di servizi, prodotti o società sono marchi o marchi di servizi di altre società.

# Glossario dei termini e delle abbreviazioni

Questo glossario descrive i termini utilizzati in questo manuale e le parole utilizzate con un significato diverso da quello ordinario. In alcuni casi, è possibile che una definizione non sia la sola applicabile ad un termine ma indichi piuttosto lo specifico senso in cui detto termine è utilizzato in questo manuale.

Se non si riesce a trovare il termine desiderato, consultare l'indice oppure la pubblicazione *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

**Abstract Window Toolkit per Java (AWT).** Una raccolta di componenti GUI (Graphical User Interface) implementata utilizzando le versioni di piattaforma native dei componenti.

**accodamento di messaggi.** Una tecnica di programmazione in cui ciascun programma in un'applicazione comunica con gli altri programmi inserendo messaggi nelle code.

**API.** Application Programming Interface.

**applet.** Un programma Java progettato per essere eseguito solo su una pagina Web.

**Application Programming Interface (API).** Una Application Programming Interface è composta dalle funzioni e dalle variabili che i programmatori possono utilizzare nelle loro applicazioni.

**AWT.** Abstract Window Toolkit per Java.

**Browser Web.** Un programma che formatta e visualizza le informazioni distribuite sul WWW.

**canale.** Vedere canale MQI.

**Canale MQI.** Un canale MQI connette un client MQSeries ad un Queue Manager su un sistema server e trasferisce le call e le risposte MQI in modo bidirezionale.

**casting.** Un termine utilizzato in Java per descrivere la conversione esplicita del valore di un oggetto o di un tipo primitivo in un altro tipo.

**classe.** Una classe è una raccolta incapsulata di dati e di metodi da applicare ai dati. E' possibile creare un'istanza di una classe per produrre un oggetto che è un'istanza della classe.

**classe superiore.** Una classe superiore è una classe estesa da altre classi. Le variabili ed i metodi pubblici della classe superiore sono disponibili per la sottoclasse.

**client.** In MQSeries, un client è un componente di runtime che fornisce l'accesso ai servizi di accodamento su un server per le applicazioni utente locali.

**coda.** Una coda è un oggetto MQSeries. Le applicazioni di accodamento dei messaggi possono eseguire operazioni di estrazione e di inserimento di messaggi in una coda.

**coda messaggi.** Vedere coda.

**Comandi MQSeries (MQSC).** Comandi leggibili, uniformi su tutte le piattaforme, utilizzati per operare sugli oggetti MQSeries.

**EJB.** Enterprise JavaBean.

**Enterprise JavaBeans (EJB).** Un'architettura di componenti server, distribuita dalla Sun Microsystems, per scrivere applicazioni di logica commerciale riutilizzabili e applicazioni aziendali trasportabili. I componenti Enterprise JavaBean sono scritti interamente in Java e possono essere eseguiti su server compatibili EJB.

**HTML.** Hypertext Markup Language

**Hypertext Markup Language (HTML).** Un linguaggio utilizzato per definire informazioni da visualizzare sul World Wide Web.

**IEEE.** Institute of Electrical and Electronics Engineers.

**IIOP.** Internet Inter-ORB Protocol.

**incapsulamento.** L'incapsulamento è un tecnica di programmazione a oggetti che rende i dati di un oggetto privati o sicuri e consente ai programmatori di accedere e modificare questi dati solo eseguendo call di metodi.

**interfaccia.** Un'interfaccia è una classe che contiene solo metodi astratti e nessuna variabile di istanza. Un'interfaccia fornisce un gruppo comune di metodi che possono essere implementati da sottoclassi di un certo numero di classi differenti.

**Internet.** Internet è una rete pubblica cooperativa di informazioni condivise. Fisicamente, Internet utilizza un sottoinsieme delle risorse totali di tutte le rete per le telecomunicazioni pubbliche esistenti simultaneamente. Tecnicamente, quello che distingue Internet come una rete pubblica cooperativa è l'utilizzo che essa fa di un

gruppo di protocolli detti TCP/IP (Transmission Control Protocol/Internet Protocol).

**Internet Inter-ORB Protocol (IIOP).** Uno standard per le comunicazioni TCP/IP tra ORB di differenti produttori.

**istanza.** Un'istanza è un oggetto. Quando si crea un'istanza di una classe per produrre un oggetto, si dice che l'oggetto è appunto un'istanza della classe.

**JAAS.** JAAS (Java Authentication ed Authorization Service).

**JAAS (Java Authentication ed Authorization Service).** Un servizio Java che fornisce l'autenticazione delle entità ed il controllo degli accessi.

**Java Message Service (JMS).** Una API della Sun Microsystem che consente di accedere ai sistemi di messaggistica aziendali dai programmi Java.

**Java Naming and Directory Service (JNDI).** Una API specificata nel linguaggio di programmazione Java. Fornisce funzioni di directory e di denominazione alle applicazioni scritte nel linguaggio di programmazione Java.

**Java 2 Platform, Enterprise Edition (J2EE).** Un gruppo di servizi, API e protocolli che forniscono la funzionalità necessaria per sviluppare applicazioni Web distribuite su più macchine.

**Java Runtime Environment (JRE).** Un sottoinsieme del Java Development Kit (JDK) che contiene i file e gli eseguibili principali che costituiscono la piattaforma standard Java. Il JRE comprende la Java Virtual Machine, le classi principali ed i file di supporto.

**Java Transaction API (JTA).** Una API che consente alla applicazioni ed ai server J2EE di accedere alle transazioni.

**Java Transaction Service (JTS).** Un programma di gestione delle transazioni che supporta JTA ed implementa la mappatura Java della specifica OMG Object Transaction Service 1.1 sotto il livello della API.

**Java Virtual Machine (JVM).** Un'implementazione software di una CPU (Central Processing Unit) che esegue il codice Java compilato (applet ed applicazioni).

**JDK.** Java Development Kit.

**JDK (Java Development Kit).** Un pachetto di software distribuito per gli sviluppatori Java, attraverso Sun Microsystems o altri. Questo pacchetto comprende l'interpreter Java, le classi Java e gli strumenti di sviluppo Java: compiler, debugger, disassembler, appletviewer, stub file generator e documentation generator.

**J2EE.** Java 2 Platform, Enterprise Edition.

**JMS.** Java Message Service.

**JNDI.** Java Naming and Directory Service.

**JRE.** Java Runtime Environment.

**JTA.** Java Transaction API.

**JTS.** Java Transaction Service.

**JVM.** Java Virtual Machine.

**LDAP.** Lightweight Directory Access Protocol.

**Lightweight Directory Access Protocol (LDAP).** Un protocollo client-server per accedere ad un servizio di directory.

**messaggio.** Nelle applicazioni di accodamento dei messaggi, un messaggio è una comunicazione inviata tra programmi.

**metodo.** Il metodo è un termine della programmazione a oggetti per una funzione o una procedura.

**MQDLH.** MQSeries dead letter header. Consultare il manuale *MQSeries Application Programming Reference*.

**MQMD.** MQSeries Message Descriptor.

**MQSC.** Comandi MQSeries.

**MQSeries.** MQSeries è una famiglia di programmi su licenza della IBM che fornisce servizi di accodamento dei messaggi.

**MQSeries Message Descriptor (MQMD).** Informazioni di controllo che descrivono le proprietà ed il formato dei messaggi, inviate come parte di un messaggio MQSeries.

**Object Management Group (OMG).** Un consorzio che definisce gli standard nella programmazione ad oggetti.

**Object Request Broker (ORB).** Un framework applicativo che fornisce interoperatività tra oggetti, creati in linguaggi differenti, in esecuzione su macchine differenti, in ambienti distribuiti eterogenei.

**oggetto.** (1) In Java, un oggetto è un'istanza di una classe. Una classe modella un gruppo di cose; un oggetto modella un particolare membro di detto gruppo. (2) In MQSeries, un oggetto è un Queue Manager, una coda oppure un canale.

**OMG.** Object Management Group.

**ORB.** Object Request Broker.

**pacchetto.** Un pacchetto in Java è un modo per fornire una porzione di codice Java di accesso ad uno specifico gruppo di classi. Il codice Java che fa parte di un determinato pacchetto dispone dell'accesso a tutte le

classi contenute nel pacchetto e a tutti i campi ed i metodi non privati contenuti nelle classi.

**privato.** Un campo privato non è visibile fuori dalla propria classe.

**protetto.** Un campo protetto è visibile solo nella propria classe, in una sottoclasse o nei pacchetti di cui fa parte la classe.

**pubblico/pubblica.** Una classe o un'interfaccia pubblica è visibile dovunque. Un metodo pubblico o una variabile pubblica sono visibili dovunque sia visibile la relativa classe

**queue manager.** Un Queue Manager è un programma di sistema che fornisce i servizi di accodamento dei messaggi alle applicazioni.

**Red Hat Package Manager (RPM).** Un sistema di impacchettamento del software da utilizzare sulle piattaforme Red Hat Linux e sulle altre piattaforme Linux e UNIX.

**RFC.** (Request For Comment) Un documento che definisce uno standard, ossia una parte del TCP/IP dei protocolli. Una RFC si avvia come una proposta e non tutte le RFC vengono adottate ed implementate.

**RPM.** Red Hat Package Manager.

**server.** (1) In MQSeries, un server è un Queue Manager che fornisce i servizi di accodamento dei messaggi alle applicazioni client in esecuzione su una stazione di lavoro remota. (2) Più genericamente, un server è un programma che risponde alle richieste di informazioni nello specifico modello di flusso di informazioni tra due programmi client/server. (3) Il computer su cui viene eseguito un programma server.

**servlet.** Un programma Java progettato per essere eseguito solo su un server Web.

**sottoclasse.** Una sottoclasse è una classe che estende un'altra classe. La sottoclasse eredita le variabili ed i metodi pubblici e protetti della sua classe superiore.

**sovraccarico.** La situazione in cui un ID fa riferimento a più elementi nello stesso ambito. In Java, è possibile sovraccaricare i metodi ma non le variabili o gli operatori.

**TCP/IP.** Transmission Control Protocol/Internet Protocol.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** Un gruppo di protocolli per le comunicazioni che supporta le funzioni di connettività peer-to-peer sia per le LAN che per le WAN.

**Uniform Resource Locator (URL).** Una sequenza di caratteri che rappresenta le risorse di informazioni su un computer o in una rete come Internet.

**URL.** Uniform Resource Locator.

**VisiBroker per Java.** Un Object Request Broker (ORB) scritto in Java.

**Web.** Vedere World Wide Web.

**World Wide Web (Web).** Il World Wide Web è un servizio Internet, basato su un gruppo comune di protocolli, che consente ad un computer server specificamente configurato di distribuire documenti tramite Internet in un modo standard.

# Glossario

# Bibliografia

In questa sezione verrà indicata la documentazione disponibile per tutti i prodotti MQSeries correnti.

## Pubblicazioni valide per più piattaforme MQSeries

La maggior parte di queste pubblicazioni, a cui talvolta viene fatto riferimento come libri della famiglia MQSeries "", sono validi per tutti i prodotti MQSeries Livello 2. I prodotti MQSeries Livello 2 più recenti sono:

- MQSeries per AIX, V5.2
- MQSeries per AS/400, V5.2
- MQSeries per AT&T GIS UNIX, V2.2
- MQSeries per Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries per Compaq Tru64 UNIX, V5.1
- MQSeries per HP-UX, V5.2
- MQSeries per Linux, V5.2
- MQSeries per OS/2 Warp, V5.1
- MQSeries per OS/390, V5.2
- MQSeries per SINIX e DC/OSx, V2.2
- MQSeries per Sun Solaris, V5.2
- MQSeries per Sun Solaris, Intel Platform Edition, V5.1
- MQSeries per Tandem NonStop Kernel, V2.2.0.1
- MQSeries per VSE/ESA, V2.1.1
- MQSeries per Windows, V2.0
- MQSeries per Windows, V2.1
- MQSeries per Windows NT e Windows 2000, V5.2

Le pubblicazioni MQSeries valide per tutte le piattaforme sono:

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries Clients*, GC33-1632
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries Using Java*, SC34-5456
- *MQSeries Application Messaging Interface*, SC34-5604

## Pubblicazioni specifiche per la piattaforma MQSeries

Ogni prodotto MQSeries è documentato in almeno una pubblicazione specifica della piattaforma, oltre ai testi della famiglia MQSeries.

**MQSeries per AIX, V5.2**

*MQSeries per AIX Quick Beginnings*, GC33-1867

**MQSeries per AS/400, V5.2**

*MQSeries for AS/400 Quick Beginnings*, GC34-5557

*MQSeries for AS/400 System Administration*, SC34-5558

*MQSeries for AS/400 Application Programming Reference (ILE RPG)*, SC34-5559

**MQSeries per AT&T GIS UNIX, V2.2**

*MQSeries for AT&T GIS UNIX System Management Guide*, SC33-1642

**MQSeries per Compaq (DIGITAL) OpenVMS, V2.2.1.1**

*MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide*, GC33-1791

**MQSeries per Compaq Tru64 UNIX, V5.1**

*MQSeries per Compaq Tru64 UNIX Quick Beginnings*, GC34-5684

**MQSeries per HP-UX, V5.2**

*MQSeries per HP-UX Quick Beginnings*, GC33-1869

**MQSeries per Linux, V5.2**

*MQSeries per Linux Quick Beginnings*, GC34-5691

**MQSeries per OS/2 Warp, V5.1**

*MQSeries per OS/2 Warp Quick Beginnings*, GC33-1868

**MQSeries per OS/390, V5.2**

*MQSeries for OS/390 Concepts and Planning Guide*, GC34-5650

*MQSeries for OS/390 System Setup Guide*, SC34-5651

*MQSeries for OS/390 System Administration Guide*, SC34-5652

*MQSeries for OS/390 Problem Determination Guide*, GC34-5892

*MQSeries for OS/390 Messages and Codes*, GC34-5891

*MQSeries for OS/390 Licensed Program Specifications*, GC34-5893

*MQSeries for OS/390 Program Directory*

**MQSeries collegamento per R/3, Versione 1.2**

*MQSeries link for R/3 User's Guide*, GC33-1934

**MQSeries per SINIX e DC/OSx, V2.2**

*MQSeries for SINIX and DC/OSx System Management Guide*, GC33-1768

**MQSeries per Sun Solaris, V5.2**

*MQSeries per Sun Solaris Quick Beginnings*, GC33-1870

**MQSeries per Sun Solaris, Intel Platform Edition, V5.1**

*MQSeries per Sun Solaris, Intel Platform Edition Quick Beginnings*, GC34-5851

**MQSeries per Tandem NonStop Kernel, V2.2.0.1**

*MQSeries for Tandem NonStop Kernel System Management Guide*, GC33-1893

**MQSeries per VSE/ESA, V2.1.1**

*MQSeries for VSE/ESA Licensed Program Specifications*, GC34-5365

*MQSeries for VSE/ESA System Management Guide*, GC34-5364

**MQSeries per Windows, V2.0**

*MQSeries for Windows User's Guide*, GC33-1822

**MQSeries per Windows, V2.1**

*MQSeries for Windows User's Guide*, GC33-1965

**MQSeries per Windows NT e Windows 2000, V5.2**

*MQSeries per Windows NT e Windows 2000 Quick Beginnings*, GC34-5389

*MQSeries for Windows NT Using the Component Object Model Interface*, SC34-5387

*MQSeries LotusScript Extension*, SC34-5404

## Manuali in formato elettronico

La maggior parte dei manuali MQSeries è fornita sia in formato cartaceo che elettronico.

## Formato HTML

La documentazione relativa a MQSeries è fornita in formato HTML con questi prodotti MQSeries:

- MQSeries per AIX, V5.2
- MQSeries per AS/400, V5.2
- MQSeries per Compaq Tru64 UNIX, V5.1
- MQSeries per HP-UX, V5.2
- MQSeries per Linux, V5.2
- MQSeries per OS/2 Warp, V5.1
- MQSeries per OS/390, V5.2
- MQSeries per Sun Solaris, V5.2
- MQSeries per Sun Solaris, Intel Platform Edition, V5.1
- MQSeries per Windows NT e Windows 2000, V5.2 (HTML compilato)
- Collegamento MQSeries per R/3, V1.2

I manuali MQSeries sono disponibili anche in formato HTML presso il sito Web della famiglia di prodotti MQSeries all'indirizzo:

```
http://www.ibm.com/software/mqseries/
```

## Portable Document Format (PDF)

I file PDF possono essere visualizzati e stampati mediante Adobe Acrobat Reader.

Per scaricare Adobe Acrobat Reader, o per ricevere le informazioni più aggiornate sulle piattaforme che supportano Acrobat Reader, visitare il sito Web della Adobe Systems Inc. all'indirizzo:

```
http://www.adobe.com/
```

Le versioni PDF dei manuali relativi a MQSeries vengono fornite con i seguenti prodotti MQSeries:

- MQSeries per AIX, V5.2
- MQSeries per AS/400, V5.2
- MQSeries per Compaq Tru64 UNIX, V5.1
- MQSeries per HP-UX, V5.2

- MQSeries per Linux, V5.2
- MQSeries per OS/2 Warp, V5.1
- MQSeries per OS/390, V5.2
- MQSeries per Sun Solaris, V5.2
- MQSeries per Sun Solaris, Intel Platform Edition, V5.1
- MQSeries per Windows NT e Windows 2000, V5.2
- Collegamento MQSeries per R/3, V1.2

Le versioni PDF di tutti i manuali MQSeries correnti sono disponibili anche presso il sito Web della famiglia di prodotti MQSeries all'indirizzo:

```
http://www.ibm.com/software/mqseries/
```

## Formato BookManager

La libreria MQSeries viene fornita in formato IBM BookManager su diversi library collection kit, incluso *Transaction Processing and Data*, SK2T-0730. E' possibile visualizzare i manuali in versione elettronica in formato IBM BookManager utilizzando i seguenti programmi forniti su licenza IBM:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ per Windows

## Formato PostScript

La raccolta di pubblicazioni MQSeries viene fornita in formato PostScript (.PS) con molti prodotti MQSeries Versione 2. I manuali in formato PostScript possono essere stampati su una stampante PostScript oppure visualizzati in un apposito programma di visualizzazione.

## Formato della Guida in linea Windows

La *MQSeries for Windows User's Guide* viene fornita nel formato della Guida di Windows con MQSeries per Windows, Versione 2.0 and MQSeries for Windows, Versione 2.1.

---

## Informazioni su MQSeries disponibili su Internet

Il sito Web della famiglia di prodotti MQSeries è disponibile all'indirizzo:

```
http://www.ibm.com/software/mqseries/
```

Seguendo i link contenuti in questo sito Web è possibile:

- Ottenere le informazioni più aggiornate sulla famiglia di prodotti MQSeries.
- Accedere ai manuali MQSeries nei formati HTML e PDF.
- Eseguire il download del SupportPac di MQSeries .

# Indice analitico

## D



## E



## F



## G



## H



## I

## J



## L



## M

## N



## O



## P

# Q



# R



# S



# T



# U



# V

# W



# X



# Z

# Riservato ai commenti del lettore

**MQSeries**
**Utilizzo di Java**

**Pubblicazione N. SC13-2958-07**

Commenti relativi alla pubblicazione in oggetto potranno contribuire a migliorarla. Sono graditi commenti pertinenti alle informazioni contenute in questo manuale ed al modo in cui esse sono presentate. Si invita il lettore ad usare lo spazio sottostante citando, ove possibile, i riferimenti alla pagina ed al paragrafo.

Si prega di non utilizzare questo foglio per richiedere informazioni tecniche su sistemi, programmi o pubblicazioni e/o per richiedere informazioni di carattere generale.

Per tali esigenze si consiglia di rivolgersi al punto di vendita autorizzato o alla filiale IBM della propria zona oppure di chiamare il ″Supporto Clienti″ IBM al numero verde 167-017001.

I suggerimenti ed i commenti inviati potranno essere usati liberamente dall'IBM e dalla Selfin e diventeranno proprietà esclusiva delle stesse.

Commenti:

Si ringrazia per la collaborazione.

Per inviare i commenti è possibile utilizzare uno dei seguenti modi.

- Spedire questo modulo all'indirizzo indicato sul retro.
- Inviare un fax al numero: +39-081-660236
- Spedire una nota via email a: translationassurance@selfin.it

Se è gradita una risposta dalla Selfin, si prega di fornire le informazioni che seguono:

Nome

Indirizzo

Società

Numero di telefono

Indirizzo e-mail

Indicandoci i Suoi dati, Lei avrà l'opportunità di ottenere dal responsabile del Servizio di Translation Assurance della Selfin S.p.A. le risposte ai quesiti o alle richieste di informazioni che vorrà sottoporci. I Suoi dati saranno trattati nel rispetto di quanto stabilito dalla legge 31 dicembre 1996, n.675 sulla "Tutela delle persone e di altri soggetti rispetto al trattamento di dati personali". I Suoi dati non saranno oggetto di comunicazione o di diffusione a terzi; essi saranno utilizzati "una tantum" e saranno conservati per il tempo strettamente necessario al loro utilizzo.

Selfin S.p.A.
Translation Assurance

Via F. Giordani, 7

80122 NAPOLI

Printed in Denmark by IBM Danmark A/S

SC13-2958-07

Spine information:

MQSeries

MQSeries Utilizzo di Java