9.4

Desenvolvendo Aplicativos para IBM MQ

Nota

Antes de usar estas informações e o produto que elas suportam, leia as informações em “Avisos” na página 1311.

Esta edição se aplica à versão 9, liberação 4 do IBM® MQ e a todas as liberações e modificações subsequentes, até que seja indicado de outra forma em novas edições

Índice

Desenvolvendo aplicativos para o IBM MQ

É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

Novo para desenvolver aplicativos para o IBM MQ?

Para saber mais sobre como desenvolver aplicativos para o IBM MQ, visite o IBM Developer:

- IBM MQ Developer Essentials *(aprenda o básico, execute uma demo, codifique um app, faça tutoriais mais avançados)*
- IBM MQ Downloads para desenvolvedores *(incluindo edições grátis do desenvolvedor e versões de avaliação)*

Você também poderá achar mais fácil desenvolver os seus aplicativos se você estiver familiarizado com os conceitos descritos nas seções a seguir:

- “Conceitos de desenvolvimento de aplicativos” na página 7
- “Considerações de design para aplicativos IBM MQ” na página 50

Suporte para linguagens e estruturas orientadas a objetos

O IBM MQ fornece suporte principal para aplicativos desenvolvidos nas linguagens e estruturas a seguir:

- JMS
- Java
- C++
- .NET

Consulte também “Aplicativos orientados a objetos” na página 16.

O.NET suporta aplicativos desenvolvidos em vários idiomas. Para ilustrar usando as classes do IBM MQ para o .NET acessar filas do IBM MQ, a documentação do produto do MQ contém informações para as linguagens a seguir:

- C# código de exemplo e aplicativos de amostra
- Aplicativos de amostra C++
- Visual Basic aplicativos de amostra

Consulte “Gravando e implementando programas do IBM MQ .NET” na página 589.

O IBM MQ suporta .NET Core para aplicativos em Windows ambientes do IBM MQ 9.1.1 e para aplicativos em Linux® ambientes do IBM MQ 9.1.2. Para mais informações, consulte “Instalando IBM MQ classes for .NET” na página 566.

ALW IBM MQ também suporta os clientes AMQP que implementam o protocolo OASIS AMQP 1.0.

MQ Light, Apache Os clientes Qpid como Apache Qpid Proton e Apache APIs Qpid JMS baseiam-se nesse protocolo.

As APIs MQ Light estão disponíveis em IBM MQ Light.

Os clientes Apache Qpid estão disponíveis em QPid Proton.

As ligações entre linguagens a seguir são fornecidas no estado em que se encontram:

- uma Ligação de Ligação
- uma Implementação de API de JavaScript que funciona com aplicativos Node.js

Suporte para APIs de REST programáticas

O IBM MQ fornece suporte para as APIs de REST programática a seguir para enviar e receber mensagens:

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

Consulte “Desenvolvendo aplicativos REST com o IBM MQ” na página 714 e também o tutorial Introdução à API de REST do sistema de mensagens IBM MQ na área do IBM MQ do IBM Developer. Este tutorial inclui exemplos nas linguagens a seguir, fornecidas no estado em que se encontram, para uso com o IBM MQ messaging REST API:

- Exemplo de Go que usa a API de REST do sistema de mensagens do MQ
- Exemplo de Node.js que usa o módulo HTTPS
- Exemplo de Node.js com o módulo Promise

Suporte para linguagens de programação processual

O IBM MQ fornece suporte para aplicativos desenvolvidos nas linguagens de programação processuais a seguir:

- C
- Windows Visual Basic (apenas sistemasWindows)
- COBOL
- z/OS Assembler (apenasIBM MQ for z/OS)
- z/OS PL/I (apenasIBM MQ for z/OS)
- IBM i RPG (apenasIBM MQ for IBM i)

Esses idiomas usar a Interface da Fila de Mensagens (MQI) para acessar serviços de enfileiramento de mensagens. Consulte “Desenvolvendo aplicativos MQI com o IBM MQ” na página 729. Observe que o Modelo de Objeto do IBM MQ, usado pelas linguagens e estruturas orientadas a objetos, fornece funções adicionais que não estão disponíveis para as linguagens processuais que usam a MQI.

Especificando o nome do aplicativo

ALW

Antes de IBM MQ 9.1.2, seria possível especificar um nome de aplicativo em aplicativos clientes Java ou JMS. A partir da IBM MQ 9.1.2, também é possível especificar o nome do aplicativo em linguagens de programação adicionais. Para obter mais informações, consulte “Especificando o nome do aplicativo em linguagens de programação suportadas” na página 54.

Tarefas relacionadas

“Desenvolvendo aplicativos para o MQ Telemetry” na página 1255

“Desenvolvendo aplicativos Microsoft Windows Communication Foundation com o IBM MQ” na página 1279

O canal customizado do Microsoft Windows Communication Foundation (WCF) para IBM MQ envia e recebe mensagens entre clientes e serviços WCF.

Referências relacionadas

“Desenvolvendo aplicativos para o Managed File Transfer” na página 1233

Especifique programas para executar com Managed File Transfer, use o Apache Ant com o Managed File Transfer, customize o Managed File Transfer com saídas de usuários e controle o Managed File Transfer ao colocar mensagens na fila de comandos do agente.

Conceitos de desenvolvimento de aplicativos

É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

Para obter informações sobre os tipos de aplicativos que você pode escrever para o IBM MQ, veja “Desenvolvendo aplicativos para o IBM MQ” na página 5 e “Ações que seus aplicativos podem executar” na página 7.

Conceitos relacionados

“Considerações de design para aplicativos IBM MQ” na página 50
Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

Ações que seus aplicativos podem executar

É possível desenvolver aplicativos para enviar e receber mensagens necessárias para suportar seus processos de negócios. É possível também desenvolver os aplicativos para gerenciar os gerenciadores de fila e recursos relacionados.

Ações que seus aplicativos podem executar em IBM MQ for Multiplatforms

Em Multiplataformas, é possível gravar aplicativos que executam as ações a seguir:

- Enviar mensagens para outros aplicativos em execução sob os mesmos sistemas operacionais. Os aplicativos podem estar no mesmo sistema ou em outro.
- Enviar mensagens para aplicativos que são executados em outras plataformas IBM MQ.
- Use o enfileiramento de mensagens de dentro do CICS para os sistemas a seguir:
  - 

TXSeries for AIX
  - 

IBM i
  - 

Windows
- Utilize o enfileiramento de mensagens de dentro do Encina para os sistemas a seguir:
  - 

AIX
  - 

Windows
- Use o enfileiramento de mensagens de dentro do Tuxedo para os sistemas a seguir:
  - 

AIX
  - AT&T
  - 

Windows
- Use o IBM MQ como um gerenciador de transações, coordenando as atualizações feitas pelos gerenciadores de recursos externos nas unidades de trabalho do IBM MQ. Os gerenciadores de recursos externos a seguir são suportados e compatíveis com a interface X/OPEN XA
  - Db2
  - Informix
  - Oracle
  - Sybase
- Processe várias mensagens juntas como uma única unidade de trabalho que pode ser confirmada ou restaurada.

- Execute a partir de um ambiente integral do IBM MQ ou de um ambiente de cliente do IBM MQ.

Ações que seus aplicativos podem executar em IBM MQ for z/OS

z/OS

Em z/OS, é possível gravar aplicativos que executam as ações a seguir:

- Use o enfileiramento de mensagens no CICS ou IMS.
- Envie mensagens entre em aplicativos em lote CICS e IMS, selecionando o ambiente mais apropriado para cada função.
- Enviar mensagens para aplicativos que são executados em outras plataformas IBM MQ.
- Processe várias mensagens juntas como uma única unidade de trabalho que pode ser confirmada ou restaurada.
- Envie mensagens para e interagir com os aplicativos IMS por meio da ponte IMS.
- Participe das unidades de trabalho coordenadas por RRS.

Cada ambiente no z/OS tem as suas próprias características, vantagens e desvantagens. A vantagem de IBM MQ for z/OS é que os aplicativos não estão vinculados a nenhum ambiente, mas podem ser distribuídos para aproveitar os benefícios de cada ambiente. Por exemplo, você pode desenvolver interfaces do usuário final usando o TSO ou CICS, é possível executar módulos de processamento intensivo em z/OS em lote e você pode executar aplicativos de banco de dados no IMS ou CICS. Em todos os casos, as várias partes do aplicativo podem se comunicar usando mensagens e filas.

Os designers de aplicativos IBM MQ devem estar cientes das diferenças e limitações impostas por esses ambientes. Por exemplo:

- O IBM MQ fornece recursos que permitem intercomunicação entre gerenciadores de filas (isto é conhecido como *enfileiramento distribuído*).
- Os métodos de confirmação e restauração de mudanças diferem entre o lote e ambientes do CICS.
- O IBM MQ for z/OS fornece suporte no ambiente do IMS para message processing programs (MPPs), interactive fast path programs (IFPs) e programas de processamento de mensagens em lote (BMPs) on-line. Se você estiver gravando programas DL/I em lote, siga a orientação dada em tópicos como “Building z/OS batch applications” na página 1036 e “z/OS batch considerations” na página 743 para programas em lote do z/OS.
- Apesar de diversas instâncias do IBM MQ for z/OS poderem existir em um único sistema z/OS, uma região CICS pode se conectar apenas um gerenciador de filas de cada vez. No entanto, mais de uma região CICS pode ser conectada no mesmo gerenciador de filas. Nos ambientes do IMS e z/OS em lote, os programas podem se conectar a mais de um gerenciador de filas.
- O IBM MQ for z/OS permite que filas locais sejam compartilhadas por um grupo de gerenciadores de filas, oferecendo melhor rendimento e disponibilidade. Essas filas são chamadas de *filas compartilhadas* e os gerenciadores de filas formam um *grupo de filas compartilhadas*, que pode processar mensagens nas mesmas filas compartilhadas. Os aplicativos em lote podem se conectar a um de vários gerenciadores de filas dentro de um grupo de filas compartilhadas, especificando o nome do grupo de filas compartilhadas, em vez de um nome de gerenciador de filas específico. Isso é conhecido como *anexar grupo lote* ou, mais simples, *anexação de grupo*. Consulte Filas compartilhadas e grupos de compartilhamento de filas.

z/OS As diferenças entre os ambientes suportados e suas limitações são mais bem explicadas em “Using and writing applications on IBM MQ for z/OS” na página 901.

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

“Considerações de design para aplicativos IBM MQ” na página 50

Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

"Escrevendo um aplicativo processual para enfileiramento" na página 732
Use essas informações para aprender como escrever aplicativos de enfileiramento, conectar e desconectar de um gerenciador de filas, publicar/assinar e abrir e fechar objetos.

"Escrevendo aplicativos clientes processuais" na página 924
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

"Usando IBM MQ classes for JMS/Jakarta Messaging" na página 83
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

"Usando o IBM MQ classes for Java" na página 353
Use o IBM MQ em um ambiente Java. O IBM MQ classes for Java permite que um aplicativo Java se conecte ao IBM MQ como um cliente IBM MQ ou se conecte diretamente a um gerenciador de filas do IBM MQ.

"Desenvolvendo aplicativos C++" na página 537
O IBM MQ fornece classes C++ equivalentes a objetos do IBM MQ e algumas classes adicionais equivalentes aos tipos de dados de matriz. Ele fornece inúmeros recursos não disponíveis por meio do MQI.

"Construindo um aplicativo processual" na página 1012
É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

Tarefas relacionadas

"Usando os programas processuais de amostra do IBM MQ" na página 1070
Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

"Desenvolvendo aplicativos do .NET" na página 565
IBM MQ classes for .NET permitir que .NET aplicativos se conectem ao IBM MQ como um IBM MQ MQI client ou se conectem diretamente a um servidor IBM MQ .

"Desenvolvendo aplicativos Microsoft Windows Communication Foundation com o IBM MQ" na página 1279
O canal customizado do Microsoft Windows Communication Foundation (WCF) para IBM MQ envia e recebe mensagens entre clientes e serviços WCF.

Assegurando

Multi

Aplicativos, nomes de aplicativos e instâncias de aplicativos

Antes de começar a projetar e a escrever seus aplicativos, familiarize-se com os conceitos básicos de aplicativos, nomes de aplicativos e instâncias de aplicativos.

Aplicativos

Multi

As conexões com um gerenciador de filas serão consideradas do mesmo *aplicativo* se elas fornecerem o mesmo *nome do aplicativo*. O nome do aplicativo é exibido como o atributo APPLTAG do comando DISPLAY CONN(*) TYPE CONN.

Notas:

1. Para aplicativos que usam uma versão do IBM MQ client anterior a IBM MQ 9.1.2, o nome do aplicativo é automaticamente configurado pelo IBM MQ client. Seu valor O depende da linguagem

de programação do aplicativo e da plataforma em que o aplicativo está em execução. Consulte PutApplName para obter mais informações.

2. Para os aplicativos IBM MQ client usando um IBM MQ client em IBM MQ 9.1.2 ou mais recente, é possível configurar o nome do aplicativo para um valor específico. Na maioria dos casos, isso não requer alterações no código do aplicativo ou uma necessidade de recompilar o aplicativo. Consulte “Usando o nome do aplicativo em linguagens de programação suportadas” na página 55 para obter informações adicionais.

Instâncias do aplicativo

Multi

As conexões são subdivididas em *instâncias de aplicativos*. Uma instância de um aplicativo é um conjunto de conexões estreitamente relacionadas que fornecem uma 'unidade de execução' para esse aplicativo.. Geralmente, este é um único processo do sistema operacional, que pode ter um número de encadeamentos e conexões IBM MQ associadas.

No IBM MQ for Multiplatforms, uma instância do aplicativo está associada a uma Tag de conexão específica. O gerenciador de filas associa automaticamente novas conexões a uma instância de aplicativo existente, quando ela pode ver que elas estão relacionadas.

Notas:

- Se estiver usando conexões do cliente, esses processos poderão se conectar ao gerenciador de filas por meio de um ou mais canais em execução.
- Em aplicativos JMS , uma instância de aplicativo é mapeada para uma conexão JMS específica e todas as sessões JMS associadas.

As instâncias do aplicativo são particularmente importantes no IBM MQ for Multiplatforms ao usar o cluster uniforme balanceamento de aplicativo automático. Em plataformas IBM MQ for Multiplatforms, é possível visualizar instâncias do aplicativo conectadas atualmente usando o comando DISPLAY APSTATUS.

Em alguns casos, o gerenciador de filas não pode executar corretamente a conexão com a associação de instância do aplicativo, em particular:

- Se várias conexões forem feitas em uma conversa compartilhada a partir do mesmo processo, usando nomes de aplicativos diferentes.
- Se as bibliotecas do cliente de nível mais antigo estiverem em uso. Por exemplo, as instalações do cliente IBM MQ JMS no IBM MQ 9.1.2 e anterior.

Nessas situações, se os aplicativos não se definirem como reconectáveis, isso será permitido, mas alguns dos agrupamentos de instâncias do aplicativo podem estar incorretos. Se qualquer uma das conexões for declarada como MQCNO_RECONNECT, isso afetará significativamente o balanceamento de aplicativo; portanto, a chamada MQCONN será, portanto, rejeitada com MQCNO_RECONNECT_INCOMPATIBLE.

Conceitos relacionados

“Especificando o nome do aplicativo em linguagens de programação suportadas” na página 54
Antes do IBM MQ 9.2.0, você já poderia especificar um nome do aplicativo nos aplicativos clientes do Java ou do JMS. A partir da IBM MQ 9.2.0, esse recurso é estendido para outras linguagens de programação no IBM MQ for Multiplatforms.

Programas aplicativos usando a MQI

Programas de aplicativoIBM MQ precisam de determinados objetos antes que eles possam ser executados com sucesso.

Figura 1 na página 11 mostra um aplicativo que remove mensagens de uma fila, processa-as e, em seguida, envia alguns resultados para outra fila no mesmo gerenciador de filas.

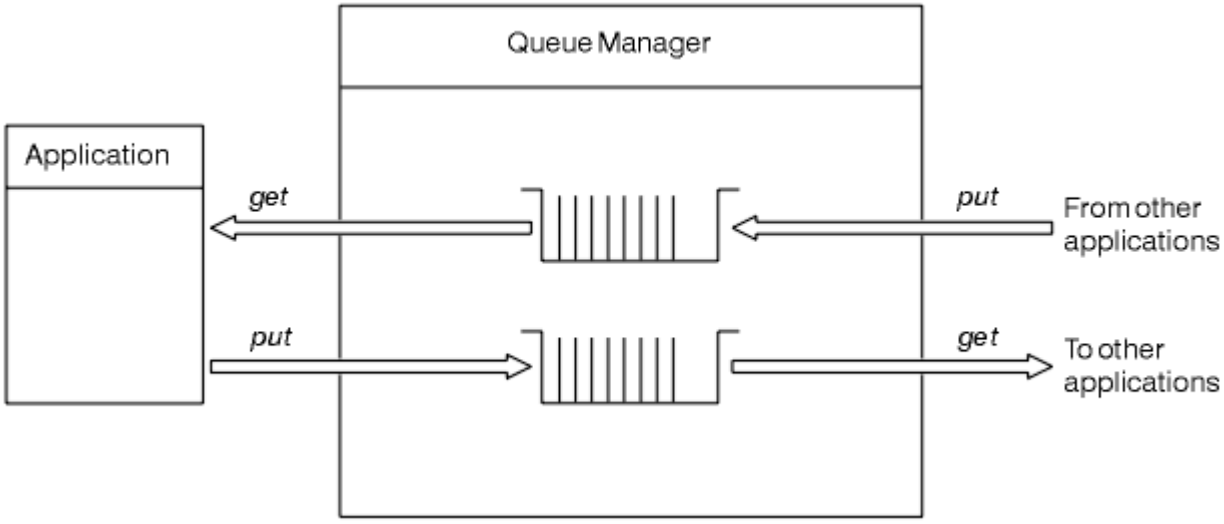


Figura 1. Filas, mensagens e aplicativos

Considerando que os aplicativos podem colocar mensagens em filas locais ou remotas (usando MQPUT), eles podem obter mensagens somente diretamente de filas locais (usando MQGET).

Antes que esse aplicativo possa ser executado, as condições a seguir devem ser satisfeitas:

- O gerenciador de filas deve existir e estar em execução.
- A primeira fila do aplicativo, da qual as mensagens devem ser removidas, deve ser definida.
- A segunda fila, na qual o aplicativo coloca as mensagens, também deve ser definida.
- O aplicativo deve ser capaz de se conectar ao gerenciador de filas. Para fazer isso, ele deve ser vinculado ao IBM MQ. Consulte o "Construindo um aplicativo processual" na página 1012.
- Os aplicativos que colocam as mensagens na primeira fila também devem se conectar a um gerenciador de filas. Se eles forem remotos, também devem ser configurados com filas de transmissão e canais. Esta parte do sistema não é mostrada em Figura 1 na página 11.

Usando conexões do cliente para se conectar a vários gerenciadores de filas do IBM MQ

É possível configurar aplicativos conectados do cliente para se conectar a mais de um gerenciador de filas (por motivos de balanceamento de carga ou de disponibilidade de serviço)

Os mecanismos primários para conseguir isso no cliente IBM MQ são o uso de tabelas de definições de canal de cliente, consulte Configurando tabelas de definições de canal de clienteou listas de conexão...

Também é possível atingir um comportamento semelhante usando produtos de balanceamento de carga externos ou agrupando o código de conexão do IBM MQ em um 'stub' que pode redirecionar nomes de host ou endereços IP.

Cada uma dessas técnicas vem com algumas restrições e podem ser mais ou menos adequadas a requisitos específicos do aplicativo As seções a seguir, embora não exaustivas, descrevem aspectos específicos que você deve considerar e o efeito dessas diferentes abordagens sobre esses aspectos.

Os clusters uniformes do IBM MQ , consulte Sobre clusters uniformes, fornecem um mecanismo poderoso para atingir o ajuste de escala horizontal de aplicativos em vários gerenciadores de filas construídos no mecanismo básico da CCDT para fornecer diversos destinos Os clusters uniformes podem fornecer recursos além do que é possível usando um balanceador de carga externo desconhecendo os protocolos subjacentes do IBM MQ e evitar alguns dos problemas discutidos abaixo, portanto, considere usar um cluster uniforme em preferência a outras técnicas, quando aplicável.

Atenção: Você deve usar com cuidado aplicativos usando IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, incluindo aqueles usando um dos IBM MQ Resource Adapters, que

se conectam a gerenciadores de filas usando tecnologias de balanceamento de carga. Se você encontrar problemas, recrie esses problemas sem tentar usar o balanceamento de carga

Há vários problemas envolvidos, o que significa que tais conexões são, na melhor das hipóteses, problemáticas e, na pior das hipóteses, totalmente não confiáveis:

- É necessário um cuidado especial ao conectar qualquer aplicativo que faça várias conexões com o gerenciador de filas usando qualquer forma de balanceamento de carga Isso inclui todos os aplicativos que usam as IBM MQ Classes para JMS/Jakarta Messaging , pois elas criam várias conexões IBM MQ em uso geral Se estiver usando um balanceador de carga externo ou stub de código customizado, isso deverá rotear conexões da mesma instância do aplicativo para o mesmo gerenciador de filas o tempo todo.
- O uso de gerenciamento de transações XA ou JTA (Java Transaction API) depende da capacidade de se conectar consistentemente ao mesmo gerenciador de filas-na prática, é improvável que isso seja prático com qualquer forma de balanceamento de carga.
- -O gerenciamento de cluster uniforme depende da capacidade de instruir os clientes a se reconectarem a gerenciadores de filas específicos sem interferência. Não é aconselhável tentar combinar balanceamento de carga externo com o uso de Clusters Uniformes

Você deve usar a funcionalidade de cluster uniforme do IBM MQ para atingir o ajuste de escala horizontal de aplicativos em vários gerenciadores de filas, em vez de tecnologias de equilíbrio de carga externas. Consulte Configurando um cluster uniformee os tópicos a seguir, para obter informações sobre clusters uniformes, incluindo como criar e usar clusters uniformes.

Termos usados nessas informações

CCDT- multi-QMGR
Representa um arquivo CCDT que contém vários canais de conexão do cliente (CLNTCONN) com o mesmo grupo, ou seja, o atributo de conexão do cliente de nome do gerenciador de filas (QMNAME CLNTCONN), no qual diferentes entradas CLNTCONN são resolvidas para gerenciadores de filas diferentes.

Isso difere de um arquivo CCDT que contém diversas entradas CLNTCONN que são simplesmente endereços IP ou nomes de host diferentes para o mesmo gerenciador de filas de várias instâncias, o que representa uma abordagem que pode ser escolhida para combinar com um stub de código.

Se você optar por uma abordagem de gerenciador de filas multi-CCDT, será necessário escolher entre priorizar as entradas ou ter o gerenciamento de carga de trabalho (WLM) escolhido a esmo:

Priorizado
Use diversas entradas ordenadas alfabeticamente com os atributos CLNTWGHT(1) e AFFINITY(PREFERRED) para se lembrar da última conexão válida.

Escolhido a esmo
Use os atributos CLNTWGHT(1) e AFFINITY(NONE). É possível ajustar o peso do WLM em servidores IBM MQ escalados de forma diferente ajustando o CLNTWGHT

Nota: É necessário evitar grandes diferenças no CLNTWGHT entre os canais.

Equilibrador de Carga
Significa um dispositivo de rede com um endereço IP Virtual (VIP) configurado com monitoramento de porta dos listeners TCP/IP de vários gerenciadores de filas do IBM MQ. A forma de configuração do VIP no dispositivo de rede depende do dispositivo de rede que está sendo usado.

As opções a seguir se relacionam apenas a aplicativos que enviam mensagens ou iniciam o sistema de mensagens de solicitação e de resposta síncronas. As considerações para aplicativos que atendem essas mensagens e solicitações, por exemplo, os listeners são completamente separados e discutidos em detalhes em "Conectando um listener de mensagem a uma fila".

Escala de mudança de código necessária para aplicativos existentes que se conectam a um único gerenciador de filas

Lista de CONNAME, CCDT multi-QMGR e Balanceador de carga
MQCONN("QMNAME") para MQCONN("*QMNAME")

O nome do gerenciador de filas pode estar na configuração do Java Naming and Directory Interface (JNDI) para aplicativos Java Platform, Enterprise Edition (Java EE). Caso contrário, isso requererá uma mudança de código de um caractere.

Stub de código
Substitua a lógica de conexão de JMS ou MQI existente por um stub de código.

Suporte para diferentes estratégias do WLM

Lista de CONNAME
Priorizado apenas.

É provável que isso tenha um efeito negativo no código.

CCDT multi-QMGR
Priorizado ou aleatório.

É provável que isso não tenha nenhum efeito no código.

Equilibrador de Carga
Qualquer um, incluindo cada conexão para todas as mensagens.

É provável que isso tenha um efeito positivo no código.

Stub de código
Qualquer um, incluindo cada mensagem para todas as mensagens.

É provável que isso tenha um efeito positivo no código.

Sobrecarga de desempenho enquanto o gerenciador de filas primário está indisponível

Lista de CONNAME
Sempre tenta o primeiro da lista.

É provável que isso tenha um efeito negativo no código.

CCDT multi-QMGR
Lembra-se da última conexão válida.

É provável que isso tenha um efeito positivo no código.

Equilibrador de Carga
O monitoramento de porta evita gerenciadores de filas inválidos.

É provável que isso tenha um efeito positivo no código.

Stub de código
Pode lembrar-se da última conexão válida e tentar novamente de forma inteligente.

É provável que isso tenha um efeito positivo no código.

suporte a transações XA

Lista de CONNAME, CCDT multi-QMGR e Balanceador de carga
O gerenciador de transações precisa armazenar informações de recuperação que se reconectam ao mesmo recurso do gerenciador de filas.

Uma chamada MQCONN resolvida para gerenciadores de filas diferentes geralmente invalida isso. Por exemplo, no Java EE, um único connection factory deve ser resolvido para um único gerenciador de filas ao usar XA.

É provável que isso tenha um efeito negativo no código.

Stub de código
O stub de código pode atender aos requisitos de XA de um gerenciador de transações, por exemplo, diversos connection factories.

É provável que isso tenha um efeito positivo no código.

Flexibilidade do administrador para ocultar mudanças de infraestrutura de apps

Lista de CONNAME
Apenas DNS.

É provável que isso tenha um efeito negativo no código.

CCDT multi-QMGR
DNS e sistema de arquivo compartilhado, sistema de arquivo compartilhado ou push de arquivo CCDT.

É provável que isso não tenha nenhum efeito no código.

Equilibrador de Carga
Endereço IP virtual (VIP) dinâmico.

É provável que isso tenha um efeito positivo no código.

Stub de código
Entradas CCDT do DNS ou do gerenciador de filas único.

É provável que isso não tenha nenhum efeito no código.

Evitando interrupções na manutenção planejada

Há outra situação a ser considerada e para a qual é necessário se planejar: como evitar a interrupção de aplicativos, por exemplo, erros e tempos limite visíveis para os usuários finais, durante a manutenção planejada de um gerenciador de filas. A melhor abordagem para evitar a interrupção é remover todo o trabalho de um gerenciador de filas antes que ele seja interrompido.

Considere um cenário de solicitação e resposta. Você deseja que todas as solicitações em andamento sejam concluídas e as respostas sejam processadas pelo aplicativo, mas não deseja que nenhum trabalho adicional seja submetido ao sistema. Simplesmente colocar o gerenciador de filas no modo quiesce não preenche essa necessidade, uma vez que os aplicativos codificados adequadamente recebem uma exceção de código de retorno RC2161 MQRC_Q_MGR_QUIESCING antes de receberem suas mensagens de resposta para solicitações em andamento.

É possível configurar PUT(DISABLED) nas filas de solicitações usadas para enviar trabalho, e ao mesmo tempo, deixar as filas de resposta como PUT(ENABLED) e GET(ENABLED). Dessa maneira, é possível monitorar a profundidade das filas de solicitações, de transmissão e de resposta. Depois que todas se estabilizam, ou seja, as solicitações em andamento são concluídas ou atingem o tempo limite, é possível parar o gerenciador de filas.

No entanto, é necessária uma codificação adequada nos aplicativos solicitantes para manipular uma fila de solicitações PUT(DISABLED), o que resulta no erro de código de retorno RC2051 MQRC_PUT_INHIBITED ao tentar enviar uma mensagem.

Observe que a exceção não ocorre ao criar a conexão com o IBM MQ ou ao abrir a fila de solicitações. A exceção ocorre apenas quando é feita uma tentativa para realmente enviar uma mensagem, usando a chamada MQPUT.

Construir um stub de código que inclua essa lógica de manipulação de erros para cenários de solicitação e de resposta e pedir que as equipes de aplicativos usem esse stub de código no futuro pode ajudar a desenvolver aplicativos com um comportamento consistente.

Desenvolvendo aplicativos clientes flexíveis e escaláveis

Por tolerância a falhas e escalabilidade, implementar aplicativos clientes que suportam opções de conexões em clusters uniformes permite que as instâncias do aplicativo sejam rebalanceadas entre os gerenciadores de filas.

Consulte Sobre clusters uniformes para uma visão geral sobre clusters uniformes.

Idealmente, esse rebalanceamento é invisível para o aplicativo, mas apenas determinados tipos de aplicativos são adequados para esse tipo de implementação e alguma consideração pode ser necessária em relação ao design do aplicativo.

Essas considerações se enquadram em duas categorias principais:

- *Caminhos de erro* raros que já podem existir para aplicativos reconectáveis, mas tornam-se mais prováveis quando implementados em um cluster uniforme. Por exemplo, após uma reconexão, qualquer unidade de trabalho em andamento é afastada e os cursores de procura se reconfiguram. Estes podem ser um evento raro para o seu aplicativo reconectável em seu ambiente atual e, portanto, não manipulado da forma mais eficiente possível pelo código do aplicativo. Revisar a lógica do aplicativo, para garantir que a manipulação adequada esteja sendo feita para tais situações, ajuda a evitar que problemas inesperados ocorram.
- *Afinidades* para um gerenciador de filas em particular. Se você tiver conhecimento que um aplicativo deve sempre se conectar de volta ao mesmo ou a um gerenciador de filas específico, o aplicativo deverá ser configurado para se reconectar a esse gerenciador de filas ou não ter sua conexão com aquele gerenciador de filas ativada. No entanto, essas afinidades podem ser temporárias, como a espera por uma mensagem de resposta. Influenciar o algoritmo de balanceamento para explicar essas afinidades a partir do código do aplicativo é discutido na seção a seguir. Para obter mais detalhes sobre essas opções, bem como realizar uma abordagem semelhantes por meio de configuração, em vez do código do aplicativo, consulte Influenciar o reequilíbrio do aplicativo em clusters uniformes.

Influenciando opções de reconexão no MQI

Consulte Opções de reconexão para obter mais informações sobre MQCNO_RECONNECT.

Se você tiver conhecimento de um aplicativo que sempre deve se conectar de volta ao mesmo ou a um gerenciados de filas específico, ele deverá ser configurado como MQCNO_RECONNECT_Q_MGR ou MQCNO_RECONNECT_DISABLED.

Influenciando o algoritmo de balanceamento no MQI

No entanto, você pode desejar controlar ou influenciar esse comportamento de rebalanceamento para se adequar às necessidades de tipos específicos de aplicativos; por exemplo, ao minimizar interrupções em transações em andamento ou garantir que aplicativos solicitantes recebam suas respostas antes de serem movidos.

Determinados comportamentos desejáveis padrão são assumidos e discutidos em Influenciando o rebalanceamento de aplicativos em clusters uniformes. Também é possível influenciar o comportamento para aplicativos específicos no momento da configuração ou da implementação por meio do arquivo client.ini conforme discutido nesse tópico.

Em outras situações, pode fazer mais sentido tornar o comportamento e os requisitos do balanceamento parte da lógica do aplicativo. Nestes casos, as mesmas características relevantes do aplicativo podem ser fornecidas ao IBM MQ ao conectar-se ao gerenciador de filas na chamada MQCONNX, em uma estrutura chamada MQBNO (opções de balanceamento).

Se você fornecer uma estrutura MQBNO, ela deve fornecer todas as informações necessárias pelo IBM MQ para tomar uma decisão sobre como e quando o aplicativo deve ser solicitado a se reconectar a um gerenciador de filas diferente.

Deve-se fornecer:

- O **Type** do aplicativo
- O **Timeout** no qual a instância é rebalanceada, independentemente do estado
- Qualquer **BalanceOptions** especial

A exceção a isso, é que você pode deixar o tempo limite como MQBNO_TIMEOUT_DEFAULT se necessário. Neste caso, o tempo limite se resolve para qualquer valor no arquivo client.ini, aplicativo ou sub-rotinas globais, se fornecidos, caso contrário, para o padrão base de 10 segundos.

Consulte MQBNO para obter detalhes sobre o formato dessa estrutura.

Para aplicativos .NET, consulte Influenciando o rebalanceamento de aplicativos em .NET para obter mais informações.

Aplicativos orientados a objetos

IBM MQ fornece suporte para JMS, Java, C++ e .NET. Esses idiomas e estruturas usam o IBM MQ Object Model, cujas classes fornecem a mesma funcionalidade que chamadas e estruturas do IBM MQ.

Algumas das linguagens e estruturas que usam o IBM MQ Object Model fornecem funções adicionais que não estão disponíveis quando você usa linguagens processuais com a Interface da Fila de Mensagens (MQI).

Para obter detalhes das classes, métodos e propriedades fornecidos por este modelo, consulte "O modelo de objeto IBM MQ" na página 17.

JMS

O IBM MQ fornece classes que implementam as especificações Jakarta Messaging 3.0 e Java Message Service 2.0 Para obter detalhes de IBM MQ classes for JMS, consulte Usando IBM MQ classes for JMS.. Para obter informações sobre as diferenças entre IBM MQ classes for Java e IBM MQ classes for JMS, para ajudá-lo a decidir qual usar, consulte "Desenvolvendo aplicativos JMS/Jakarta Messaging e Java" na página 82.

O IBM MQ Message Service Client (XMS) for C/C++ e o IBM MQ Message Service Client (XMS) for .NET fornecem uma interface de programação de aplicativos (API) chamada XMS que tem o mesmo conjunto de interfaces que a API do Java Message Service (JMS). Para obter informações adicionais, consulte "Desenvolvendo aplicativos do XMS .NET" na página 625.

Java

Consulte Usando IBM MQ classes for Java para obter informações sobre codificação de programas usando o IBM MQ Object Model no Java.

Stabilized A IBM não fará aprimoramentos adicionais no IBM MQ classes for Java e ele ficará funcionalmente estabilizado no nível enviado no IBM MQ 8.0. Para obter informações sobre as diferenças entre o IBM MQ classes for Java e o IBM MQ classes for JMS para ajudá-lo a decidir qual usar, veja "Desenvolvendo aplicativos JMS/Jakarta Messaging e Java" na página 82.

C++

O IBM MQ fornece classes C++ equivalentes a objetos do IBM MQ e algumas classes adicionais equivalentes aos tipos de dados de matriz. Ele fornece inúmeros recursos não disponíveis por meio do MQI. Veja Usando C++ para obter informações sobre codificar programas usando o Modelo de Objeto IBM MQ em C++. Os Clientes de Serviço de Mensagem para C/C++ e .NET fornecem uma interface de programação de aplicativos (API) chamada XMS que possui o mesmo conjunto de interfaces que a API Java Message Service (JMS).

.NET

Consulte Desenvolvendo aplicativos .NET para obter informações sobre codificação de programas do .NET usando as classes do IBM MQ .NET. O Message Service Clients para C/C++ e .NET fornecem uma interface de programação de aplicativos (API) chamada XMS que possui o mesmo conjunto de interfaces que a API Java Message Service (JMS).

Conceitos relacionados

"Desenvolvendo aplicativos MQI com o IBM MQ" na página 729
IBM MQ fornece suporte para C, Visual Basic, COBOL, Assembler, RPG, pTAL e PL/I. Essas linguagens processuais utilizam a interface de fila de mensagens (MQI) para acessar serviços de enfileiramento de mensagens.

Visão geral técnica

"Conceitos de desenvolvimento de aplicativos" na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

Referências relacionadas

Desenvolvendo a Referência do Aplicativo

O modelo de objeto IBM MQ

O IBM MQ Object Model consiste em classes, métodos e propriedades.

O modelo de objeto do IBM MQ consiste em:

- *Classes* que representam conceitos familiares do IBM MQ, como gerenciadores de filas, filas e mensagens.
- *Métodos* em cada classe correspondendo a chamadas MQI.
- *Propriedades* em cada classe correspondendo as atributos de objetos do IBM MQ.

Ao criar um aplicativo IBM MQ usando o modelo de objeto do IBM MQ, você cria instâncias dessas classes no aplicativo. Uma instância de uma classe em programação orientada a objetos é chamada de *objeto*. Quando um objeto tiver sido criado, você interage com o objeto examinando ou configurando os valores das propriedades do objeto (o equivalente a emitir uma chamada MQINQ ou MQSET) e fazendo chamadas de método com relação ao objeto (o equivalente a emitir as outras chamadas MQI).

Classes

O IBM MQ Object Model fornece o conjunto de classes base a seguir.

A implementação real do modelo varia um pouco entre os diferentes ambientes suportados orientados a objetos.

MQQueueManager
Um objeto da classe MQQueueManager representa uma conexão com um gerenciador de filas. Ele tem métodos para Connect(), Disconnect(), Commit() e Backout() (o equivalente de MQCONN ou MQCONNX, MQDISC, MQCMIT e MQBACK). Ele tem propriedades correspondentes aos atributos de um gerenciador de filas. Acessar uma propriedade de atributo do gerenciador de filas implicitamente conecta ao gerenciador de filas se ainda não estiver conectado. Destruir um objeto MQQueueManager implicitamente desconecta do gerenciador de filas.

MQQueue
Um objeto da classe MQQueue representa uma fila. Ele tem métodos para efetuar Put() e Get() de mensagens na e da fila (o equivalente de MQPUT e MQGET). Ele tem propriedades correspondentes aos atributos de uma fila. Acessar uma propriedade de atributo da fila ou emitir uma chamada de método Put() ou Get(), implicitamente abre a fila (o equivalente de MQOPEN). Destruir um objeto MQQueue implicitamente fecha a fila (o equivalente de MQCLOSE).

MQTopic
Um objeto da classe MQTopic representa um tópico. Ele tem métodos para efetuar Put() (publicar) e Get() (receber ou assinar) mensagens do tópico (o equivalente de MQPUT e MQGET). Ele tem propriedades correspondentes aos atributos de um tópico. Um objeto MQTopic só pode ser acessado para publicação ou assinatura, não ambas simultaneamente. Quando usado para receber mensagens, o objeto MQTopic pode ser criado com uma assinatura não gerenciada ou gerenciada e como um assinante durável ou não durável - diversos construtores sobrecarregados são fornecidos para esses cenários diferentes.

MQMessage
Um objeto da classe MQMessage representa uma mensagem a ser colocada em uma fila ou obtida de uma fila. Ele contém um buffer e engloba dados do aplicativo e MQMD. Ele tem propriedades correspondentes aos campos do MQMD e métodos que permitem gravar e ler dados do usuário de tipos diferentes (por exemplo, sequências, números inteiros longos, números inteiros curtos, bytes únicos) no buffer e a partir dele.

MQPutMessageOptions
Um objeto da classe MQPutMessageOptions representa a estrutura MQPMO. Ele tem propriedades correspondentes aos campos do MQPMO.

MQGetMessageOptions
Um objeto da classe MQGetMessageOptions representa a estrutura MQGMO. Ele tem propriedades correspondentes aos campos do MQGMO.

MQProcess
Um objeto da classe MQProcess representa uma definição de processo (usada com acionamento). Ele tem propriedades que representam os atributos de uma definição de processo.

MQDistributionList
Um objeto da classe MQDistributionList representa uma lista de distribuição (usada para enviar múltiplas mensagens com um único MQPUT). Ele contém uma lista de objetos MQDistributionListItem.

MQDistributionListItem
Um objeto da classe MQDistributionListItem representa um único destino de lista de distribuição. Ele contém as estruturas MQOR, MQRR e MQPMR e tem propriedades correspondentes aos campos dessas estruturas.

Referências do objeto

Em um programa IBM MQ que usa a MQI, o IBM MQ retorna as manipulações de conexões e de objeto para o programa.

Esses identificadores devem ser passados como parâmetros em chamadas subsequentes do IBM MQ. Com o IBM MQ Object Model, esses identificadores são ocultados do programa de aplicativo. Em vez disso, a criação de um objeto a partir de uma classe resulta em uma referência do objeto que está sendo retornada ao programa de aplicativo. É esta referência do objeto que é usada ao fazer chamadas de método e acessos de propriedade com relação ao objeto.

Códigos de retorno

Emitir uma chamada de método ou configurar um valor de propriedade resulta em códigos de retorno serem configurados.

Esses códigos de retorno são um código de conclusão e um código de razão e são eles próprios propriedades do objeto. Os valores de código de conclusão e de código de razão são os mesmos que aqueles definidos para a MQI, com alguns valores adicionais específicos para o ambiente orientado a objetos.

Mensagens IBM MQ

Uma mensagem do IBM MQ consiste em propriedades de mensagem e dados do aplicativo. O descritor de mensagens de enfileiramento de mensagens (MQMD) contém as informações de controle que acompanham os dados do aplicativo quando uma mensagem viaja entre os aplicativos de envio e de recebimento.

Partes de uma mensagem

As mensagens do IBM MQ consistem em duas partes:

- Propriedades da Mensagem
- Dados do aplicativo

O Figura 2 na página 19 representa uma mensagem e mostra como ele é logicamente dividido em propriedades de mensagem e dados do aplicativo.

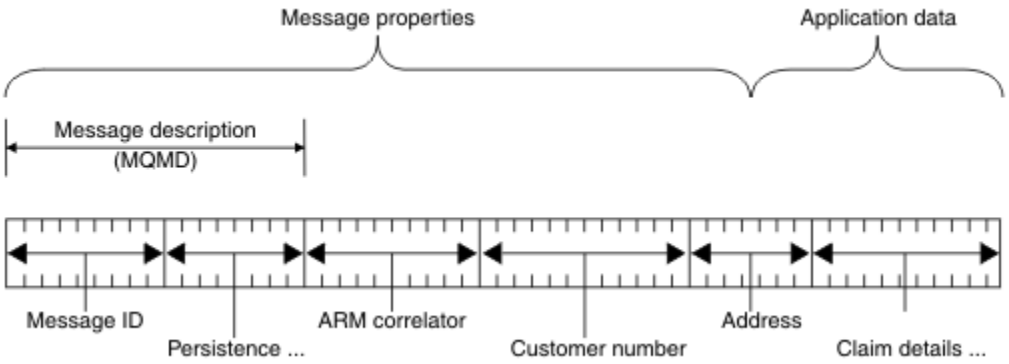


Figura 2. Representação de uma mensagem

Os dados do aplicativo que são transportados em uma mensagem do IBM MQ não são alterados por um gerenciador de filas, a menos que a conversão de dados seja efetuada nele. Além disso, o IBM MQ não coloca nenhuma restrição sobre o conteúdo destes dados. O comprimento dos dados em cada mensagem não pode exceder o valor do atributo **MaxMsgLength** da fila e do gerenciador de filas.

ALW No AIX, Linux, and Windows, o atributo *MaxMsgLength* do gerenciador de filas e a fila assumem o padrão de 4 MB (4 194 304 bytes) que é possível mudar para até um máximo de 100 MB (104 857 600 bytes), se necessário.

IBM i No IBM i, o atributo *MaxMsgLength* do gerenciador de filas e a fila assumem o padrão de 4 MB (4 194 304 bytes) que é possível mudar para até um máximo de 100 MB (104 857 600 bytes), se necessário. Se você estiver pretendendo usar mensagens do IBM MQ com mais de 15 MB no IBM i, consulte “Construindo seu aplicativo processual no IBM i” na página 1019.

z/OS No z/OS, o atributo **MaxMsgLength** do gerenciador de filas é fixado em 100 MB e o atributo **MaxMsgLength** da fila é padronizado para 4 MB (4.194.304 bytes) o que é possível mudar até um máximo de 100 MB se necessário.

Faça suas mensagens um pouco menores que o valor do atributo **MaxMsgLength** em algumas circunstâncias. Para obter mais informações, consulte “Os dados em sua mensagem” na página 769.

Você cria uma mensagem quando usa as chamadas MQPUT ou MQPUT1 MQI. Como entrada para essas chamadas, você fornece as informações de controle (como a prioridade da mensagem e o nome de uma fila de resposta) e seus dados e a chamada, então, coloca a mensagem em uma fila. Consulte MQPUT e MQPUT1 para obter mais informações sobre estas chamadas.

Descritor de Mensagens

É possível acessar informações de controle de mensagem usando a estrutura MQMD, que define o *descritor de mensagens*.

Para obter uma descrição integral da estrutura MQMD, consulte MQMD – Descritor de mensagens.

Para obter uma descrição de como usar os campos no MQMD que contêm informações sobre a origem da mensagem, consulte “Contexto da mensagem” na página 48.

Existem diferentes versões do descritor de mensagens. Informações adicionais para agrupamento e segmentação de mensagens (consulte “Grupos de mensagens” na página 45) são fornecidas na Versão 2 do descritor de mensagens (ou o MQMDE). É o mesmo que o descritor de mensagens na Versão 1, mas tem campos adicionais. Esses campos são descritos no MQMDE – Extensão do descritor de mensagens.

Tipos de Mensagem

Existem quatro tipos de mensagens definidos por IBM MQ.

Estas quatro mensagens são:

- Datagrama
- Mensagens de solicitação
- Mensagens de resposta
- Mensagens de relatório
  - Tipos de mensagem de relatório
  - Opções de mensagem de relatório

Aplicativos podem usar os três primeiros tipos de mensagens para passar as informações entre si. O quarto tipo, relatório, é para os aplicativos e os gerenciadores de fila usarem para relatar as informações sobre eventos como a ocorrência de um erro.

Cada tipo de mensagem é identificado por um valor MQMT_*. Também é possível definir seus próprios tipos de mensagens. Para o intervalo de valores que podem ser usados, consulte MsgType.

Datagramas

Use um *datagrama* quando não precisar de uma resposta do aplicativo que recebe a mensagem (ou seja, recebe a mensagem a partir da fila).

Um exemplo de um aplicativo que pode usar os datagramas é aquele que exibe as informações de voo em um salão do aeroporto. Uma mensagem pode conter os dados para uma tela inteira de informações do voo. É pouco provável que tal aplicativo solicite um reconhecimento para uma mensagem porque provavelmente não importa se uma mensagem não é entregue. O aplicativo envia uma mensagem de atualização após um curto período.

Mensagens de Pedidos

Use uma *mensagem de solicitação* quando desejar uma resposta do aplicativo que recebe a mensagem.

Um exemplo de um aplicativo que poderia usar mensagens de solicitação é um que exiba o saldo de uma conta de verificação. A mensagem de solicitação poderia conter o número da conta e a mensagem de resposta iria conter o saldo da conta.

Se desejar vincular sua mensagem de resposta a sua mensagem de solicitação, existem duas opções:

- Torne o aplicativo que trata da mensagem de solicitação responsável por assegurar que coloque as informações na mensagem de resposta relacionada à mensagem de solicitação.
- Use o campo de relatório no descritor de mensagens da sua mensagem de solicitação para especificar o conteúdo dos campos *MsgId* e *CorrelId* da mensagem de resposta:
  - É possível solicitar que o *MsgId* ou o *CorrelId* da mensagem original seja copiado para o campo *CorrelId* da mensagem de resposta (a ação padrão é copiar *MsgId*).
  - É possível solicitar que um novo *MsgId* seja gerado para a mensagem de resposta ou que o *MsgId* da mensagem original seja copiado para o campo *MsgId* da mensagem de resposta (a ação padrão é gerar um novo identificador de mensagem).

Mensagens de Resposta

Use uma *mensagem de resposta* ao responder outra mensagem.

Ao criar uma mensagem de resposta, respeite quaisquer opções que tenham sido configuradas no descritor de mensagens da mensagem à qual estiver respondendo. Opções de relatório especificam o conteúdo do identificador de mensagem (*MsgId*) e campos do identificador de correlação (*CorrelId*). Esses campos permitem que o aplicativo que recebe a resposta correlacioná-la com sua solicitação original.

Mensagens de relatório

Mensagens de relatório informam os aplicativos sobre os eventos como a ocorrência de um erro ao processar uma mensagem.

Elas podem ser geradas por:

- Um gerenciador de filas,
- Um agente do canal de mensagem (por exemplo, se não puder entregar a mensagem) ou
- Um aplicativo (por exemplo, se ele não puder usar os dados na mensagem).

Mensagens de relatório podem ser geradas a qualquer momento e podem chegar em uma fila quando o seu aplicativo não estiver esperando por elas.

Tipos de Mensagem de Relatório

Ao colocar uma mensagem em uma fila, é possível selecionar para receber:

- Uma *mensagem de relatório de exceção*. Isso é enviado em resposta a uma mensagem com o conjunto de sinalizações de exceção. Isso é gerado pelo agente do canal de mensagem (MCA) ou pelo aplicativo.
- Uma *mensagem de relatório de validação*. Isso indica que um aplicativo tentou recuperar uma mensagem que tinha atingido seu limite de expiração; a mensagem é marcada para ser descartada. Este tipo de relatório é gerado pelo gerenciador de filas.
- Uma *mensagem de relatório de confirmação de chegada (COA)*. Isso indica que a mensagem atingiu sua fila de destino. Ela é gerada pelo gerenciador de filas.
- Uma *mensagem de relatório de confirmação de entrega (COD)*. Isso indica que a mensagem foi recuperada por um aplicativo de recebimento. Ela é gerada pelo gerenciador de filas.
- Uma *mensagem de relatório de notificação de ação positiva (PAN)*. Isso indica que uma solicitação foi atendida com êxito (ou seja, a ação solicitada na mensagem foi executada com êxito). Este tipo de relatório é gerado pelo aplicativo.
- Uma *mensagem de relatório de notificação de ação negativa (NAN)*. Isso indica que uma solicitação não foi atendida com êxito (ou seja, a ação solicitada na mensagem não foi executada com êxito). Este tipo de relatório é gerado pelo aplicativo.

Nota: Cada tipo de mensagem de relatório contém um dos seguintes:

- A mensagem original inteira
- Os primeiros 100 bytes de dados na mensagem original
- Nenhum dado da mensagem original

É possível solicitar mais de um tipo de mensagem de relatório ao colocar uma mensagem em uma fila. Se forem selecionadas a mensagem de relatório de confirmação de entrega e as opções de mensagem de relatório de exceção, caso a mensagem falhe ao ser entregue, você receberá uma mensagem de relatório de exceção. No entanto, se for selecionada apenas a opção da mensagem de relatório de confirmação de entrega e a mensagem não for entregue, você não receberá uma mensagem de relatório de exceção.

As mensagens de relatório solicitadas, quando os critérios para gerar uma mensagem específica forem atendidos, serão as únicas que você receberá.

Opções da Mensagem de Relatório

É possível *descartar* uma mensagem depois que uma exceção tiver suscitado. Se for selecionada a opção descartar e tiver sido solicitada uma mensagem de relatório de exceção, a mensagem de relatório vai para o *ReplyToQ* e *ReplyToQMgr* e a mensagem original será descartada.

Nota: Um benefício disso é que é possível reduzir o número de mensagens indo para a fila de mensagens não entregues. No entanto, isso significa que seu aplicativo, a menos que envie apenas mensagens de datagrama, tem que lidar com mensagens retornadas. Quando uma mensagem de relatório de exceção é gerada, ela herda a persistência da mensagem original.

Se uma mensagem de relatório não puder ser entregue (se a fila estiver cheia, por exemplo), a mensagem de relatório será colocada na fila de mensagens não entregues.

Se desejar receber uma mensagem de relatório, especifique o nome de sua fila de resposta no campo *ReplyToQ*; caso contrário, o MQPUT ou MQPUT1 de sua mensagem original falhará com MQRC_MISSING_REPLY_TO_Q.

É possível usar outras opções de relatório no descritor de mensagens (MQMD) de uma mensagem para especificar o conteúdo dos campos *MsgId* e *CorrelId* de quaisquer mensagens de relatório que sejam criadas para a mensagem:

- É possível solicitar que o *MsgId* ou o *CorrelId* da mensagem original seja copiado ao campo *CorrelId* da mensagem de relatório. A ação padrão é copiar o identificador de mensagem. Use MQRO_COPY_MSG_ID_TO_CORRELID porque ele permite que o emissor de uma mensagem correlacione a mensagem de resposta ou relatório à mensagem original. O identificador de correlação da mensagem de resposta ou de relatório é idêntico ao identificador de mensagem da mensagem original.
- É possível solicitar que um novo *MsgId* seja gerado para a mensagem de relatório ou que o *MsgId* da mensagem original seja copiado para o campo *MsgId* da mensagem de relatório. A ação padrão é gerar um novo identificador de mensagem. Use MQRO_NEW_MSG_ID porque ele assegura que cada mensagem no sistema tenha um identificador de mensagem diferente e possa ser distinguida de maneira inequívoca de todas as outras mensagens no sistema.
- Aplicativos especializados podem precisar usar MQRO_PASS_MSG_ID ou MQRO_PASS_CORREL_ID. No entanto, é necessário designar o aplicativo que lerá as mensagens da fila para assegurar que funcionará corretamente quando, por exemplo, a fila contiver diversas mensagens com o mesmo identificador de mensagem.

  Aplicativos do servidor devem verificar as configurações dessas sinalizações na mensagem de solicitação e configurar os campos *MsgId* e *CorrelId* na mensagem de resposta ou de relatório conforme apropriado.

  Aplicativos que agem como intermediários entre um aplicativo do solicitante e um aplicativo do servidor não precisam verificar as configurações dessas sinalizações. Isso ocorre porque esses aplicativos geralmente precisam redirecionar a mensagem para o aplicativo do servidor com os campos *MsgId*, *CorrelId* e *Report* inalterados. Isto permite que o aplicativo do servidor copie o *MsgId* da mensagem original no campo *CorrelId* da mensagem de resposta.

Ao gerar um relatório sobre uma mensagem, os aplicativos do servidor devem testar para ver se alguma dessas opções foi configurada.

Para obter mais informações sobre como usar mensagens de relatório, consulte Relatório.

Para indicar a natureza do relatório, os gerenciadores de filas usam um intervalo de códigos de feedback. Eles colocam esses códigos no campo *Feedback* do descritor de mensagens de uma mensagem de relatório. Gerenciadores de filas também podem retornar códigos de razão MQI no campo *Feedback*. O IBM MQ define uma faixa de códigos de feedback para os aplicativos usarem.

Para obter mais informações sobre feedback e códigos de razão, consulte Feedback.

Um exemplo de um programa que poderia usar um código de feedback é um que monitore as cargas de trabalho de outros programas que atendem a uma fila. Se houver mais de uma instância de um programa atendendo uma fila, o número de mensagens chegando na fila não mais justificará isso, tal programa pode enviar uma mensagem de relatório (com o código de feedback MQFB_QUIT) para um dos programas de atendimento para indicar que o programa deve terminar sua atividade. (Um programa de monitoramento poderia usar a chamada MQINQ para descobrir quantos programas estão atendendo uma fila.)

Multi *Relatórios e mensagens segmentadas*

Se uma mensagem for segmentada e você pedir que relatórios sejam gerados, poderá receber mais relatórios do que teria recebido se a mensagem não tivesse sido segmentada. Os relatórios sobre mensagens segmentadas estão disponíveis apenas em Multiplataformas

Para obter uma descrição de mensagens segmentadas, consulte “Segmentação de mensagem” na página 804.

Para relatórios gerados pelo IBM MQ

Se você segmentar suas mensagens ou permitir que o gerenciador de filas faça isso, há apenas um caso em que é possível esperar receber um único relatório para a mensagem inteira. Isso ocorre quando você tiver solicitado apenas relatórios COD e tiver especificado MQGMO_COMPLETE_MSG no aplicativo de get.

Em outros casos, seu aplicativo deve estar preparado para lidar com diversos relatórios; geralmente, um para cada segmento.

Nota: Se segmentar suas mensagens e precisar que somente os primeiros 100 bytes dos dados da mensagem original sejam retornados, mude a configuração das opções de relatório para solicitar relatórios sem dados para segmentos que têm um deslocamento de 100 ou mais. Se não fizer isso e deixar a configuração para que cada segmento solicite 100 bytes de dados e você recuperar as mensagens de relatório com um único MQGET especificando MQGMO_COMPLETE_MSG MQGET, os relatórios serão montados em uma grande mensagem contendo 100 bytes de dados lidos em cada deslocamento apropriado. Se isso acontecer, precisará de um buffer grande ou especificar MQGMO_ACCEPT_TRUNCATED_MSG.

Para relatórios gerados por aplicativos

Se o seu aplicativo gerar relatórios, sempre copie os cabeçalhos do IBM MQ que estão presentes no início dos dados da mensagem original para os dados da mensagem de relatório.

Em seguida, inclua nenhum, 100 bytes ou todos os dados da mensagem original (ou qualquer outra quantia que você normalmente incluiria) nos dados da mensagem de relatório.

É possível reconhecer os cabeçalhos do IBM MQ que devem ser copiados consultando os nomes de Format sucessivos, começando com o MQMD e continuando até quaisquer cabeçalhos presentes. Os nomes de `Format` a seguir indicam esses cabeçalhos do IBM MQ:

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* significa qualquer nome que começa com os caracteres MQH.

O nome de `Format` ocorre em posições específicas para MQDLH e MQXQH, mas para os outros cabeçalhos do IBM MQ ocorre na mesma posição. O comprimento do cabeçalho está contido em um campo que também ocorre na mesma posição para MQMDE, MQIMS e todos os cabeçalhos MQH*.

Se estiver usando um MQMD Versão 1 e estiver relatando sobre um segmento, uma mensagem em um grupo ou uma mensagem para a qual a segmentação é permitida, os dados do relatório devem começar com um MQMDE. Configure o campo *`OriginalLength`* para o comprimento dos dados da mensagem original, excluindo os comprimentos de quaisquer cabeçalhos do IBM MQ que encontrar.

Recuperando relatórios

Se você solicitar relatórios para COA ou COD, é possível solicitar que sejam remontados com MQGMO_COMPLETE_MSG.

Um MQGET com MQGMO_COMPLETE_MSG é satisfeito quando mensagens de relatório suficientes (de um único tipo, por exemplo COA e com o mesmo *`GroupId`*) estão presentes na fila para representar uma mensagem original completa. Isso é verdadeiro mesmo se as mensagens de relatório em si não contiverem dados originais completos; o campo *`OriginalLength`* em cada mensagem de relatório fornece o comprimento dos dados originais representados por essa mensagem de relatório, mesmo se os dados em si não estiverem presente.

É possível usar essa técnica, mesmo se houver vários tipos de relatórios diferentes presentes na fila (por exemplo, ambos COA e COD), porque um MQGET com MQGMO_COMPLETE_MSG remontará as mensagens de relatório apenas se tiverem o mesmo código *Feedback*. No entanto, não é possível geralmente usar esta técnica para relatórios de exceção, porque, em geral, eles têm códigos de *Feedback* diferentes.

É possível utilizar essa técnica para obter uma indicação positiva que a mensagem inteira tiver chegado. Entretanto, na maioria dos casos, é necessário atender a possibilidade de que alguns segmentos chegam enquanto outras podem gerar uma exceção (ou expiração, se você tiver permitido isso). Não é possível utilizar MQGMO_COMPLETE_MSG nesse caso, porque, em geral, você pode obter diferentes *Feedback* códigos para diferentes segmentos e, você pode obter mais de um relatório para um segmento. É possível, no entanto, utilizar MQGMO_ALL_SEGMENTS_AVAILABLE.

Para permitir isso, você pode precisar recuperar relatórios conforme eles chegam e construir uma imagem em seu aplicativo de o que aconteceu com a mensagem original. É possível usar o *GroupId* campo na mensagem de relatório para correlacionar relatórios com o *GroupId* da mensagem original e o *Feedback* campo para identificar o tipo de cada mensagem de relatório. A maneira na qual você faz isso depende de seus requisitos de aplicativo.

Uma abordagem é conforme a seguir:

- Peça para relatórios COD e relatórios de exceção.
- Após um tempo específico, verifique se um conjunto completo de relatórios de confirmação de entrega foi recebido usando MQGMO_COMPLETE_MSG. Se sim, seu aplicativo sabe que a mensagem inteira foi processada.
- Se não, e relatórios de exceções relacionados a esta mensagem estiverem presentes, manipule o problema como para mensagens não segmentadas, mas se assegure de limpar os segmentos órfãos em algum ponto.
- Se houver segmentos para o qual não há relatos de nenhum tipo, os segmentos originais (ou os relatórios) podem estar esperando que um canal seja reconectado ou a rede pode estar sobrecarregada em algum ponto. Se nenhuma exceção em todos os relatórios for recebida (ou se você achar que aquelas que você possui podem ser temporárias apenas), você poderá optar por permitir que seu aplicativo esperar um pouco mais.

  Como antes, isso é similar às considerações que você tem quando lidar com mensagens não segmentadas, exceto que se deve também considerar a possibilidade de limpeza de segmentos órfão.

Se a mensagem original não é crítica (por exemplo, se for uma consulta ou uma mensagem que pode ser repetido mais tarde), configure um tempo de expiração para assegurar que os segmentos órfãos são removidos.

os gerenciadores de filas com versão anterior

Quando um relatório é gerado por um gerenciador de filas que suporta segmentação, mas é recebido em um gerenciador de filas que não suporta segmentação, a estrutura MQMDE (que identifica o *Offset* e o *OriginalLength* representados pelo relatório) é sempre incluída nos dados do relatório, além de zero, 100 bytes ou de todos os dados originais na mensagem.

No entanto, se um segmento de uma mensagem passa através de um gerenciador de filas que não suporta segmentação, se um relatório for gerado, a estrutura MQMDE na mensagem original será tratada apenas como dados. Não é, portanto, incluído nos dados de relatório se zero bytes dos dados originais foram solicitados. Sem o MQMDE, a mensagem de relatório não pode ser útil.

Pedido de pelo menos 100 bytes de dados em relatórios se houver uma possibilidade de que a mensagem pode percorrer através de um gerenciador de filas de nível anterior.

Formato de informações de controle de mensagem e de dados de mensagens

O gerenciador de filas está interessado apenas no formato das informações de controle dentro de uma mensagem, enquanto que os aplicativos que tratam da mensagem estão interessados no formato das informações de controle e dos dados.

Formato de informações de controle de mensagem

As informações de controle nos campos de sequência de caracteres do descritor de mensagens devem estar no conjunto de caracteres usado pelo gerenciador de filas.

O atributo **CodedCharSetId** do objeto do gerenciador de filas define esse conjunto de caracteres. As informações de controle devem estar nesse conjunto de caracteres, pois, quando aplicativos transmitem mensagens de um gerenciador de filas para outro, os agentes do canal de mensagens que transmitem as mensagens usam o valor desse atributo para determinar qual conversão de dados executar.

Formato dos dados da mensagem

É possível especificar qualquer uma das seguintes ações:

- O formato dos dados do aplicativo
- O conjunto de caracteres dos dados de caractere
- O formato de dados numéricos

Para fazer isso, use estes campos:

Format
Isso indica para o receptor de uma mensagem o formato dos dados do aplicativo na mensagem.

Quando o gerenciador de filas cria uma mensagem, em algumas circunstâncias ele usa o campo *Format* para identificar o formato dessa mensagem. Por exemplo, quando um gerenciador de filas não pode entregar uma mensagem, ele coloca a mensagem em uma fila de devoluções (mensagem não entregue). Ele inclui um cabeçalho (contendo mais informações de controle) na mensagem, e muda o campo *Format* para mostrar isso.

O gerenciador de filas possui vários *formatos integrados* com nomes que começam com MQ, por exemplo, MQFMT_STRING. Se isso não atender às suas necessidades, você poderá definir seus próprios formatos (*formatos definidos pelo usuário*), mas não deverá usar nomes que começam com MQ para isso.

Quando você criar e usar seus próprios formatos, deverá gravar uma saída de conversão de dados para suportar um programa obtendo a mensagem usando MQGMO_CONVERT.

CodedCharSetId
Isso define o conjunto de caracteres de dados de caracteres na mensagem. Se desejar configurar esse conjunto de caracteres como aquele do gerenciador de filas, poderá configurar esse campo como a constante MQCCSI_Q_MGR ou MQCCSI_INHERIT.

Quando você obtiver uma mensagem de uma fila, compare o valor do campo *CodedCharSetId* com o valor que seu aplicativo está esperando. Se os dois valores forem diferentes, talvez seja necessário converter os dados de caracteres na mensagem ou usar uma saída de mensagem de conversão de dados, se houver uma disponível.

Encoding
Isto descreve o formato dos dados de mensagens numéricas que contêm números inteiros binários, números inteiros decimais compactados e números de vírgula flutuante. Ele é, geralmente, codificado de acordo com a máquina específica na qual o gerenciador de filas está em execução.

Ao colocar uma mensagem em uma fila, você, geralmente, especifica a constante MQENC_NATIVE no campo *Encoding*. Isso significa que a codificação dos dados de mensagens é a mesma que a da máquina na qual seu aplicativo está em execução.

Quando você obtiver uma mensagem de uma fila, compare o valor do campo *Encoding* no descritor de mensagens com o valor da constante MQENC_NATIVE em sua máquina. Se os dois valores forem diferentes, talvez seja necessário converter os dados numéricos na mensagem ou usar uma saída de mensagem de conversão de dados, se houver uma disponível.

Conversão de Dados do Aplicativo

Dados do aplicativo podem precisar ser convertidos para o conjunto de caracteres e codificação requeridos por outro aplicativo em que diferentes plataformas são de interesse.

Ele pode ser convertido no gerenciador de filas de envio ou no gerenciador de filas de recebimento. Se a biblioteca de formatos integrados não atende suas necessidades, é possível definir a sua própria. O tipo de conversão depende do formato da mensagem que estiver especificado no campo formato do descritor de mensagens, MQMD.

Nota: Mensagens com MQFMT_NONE especificado não são convertidas.

Conversão no gerenciador de filas de envio

Configure o atributo do canal CONVERT para YES se for necessário que o agente do canal da mensagem enviada (MCA) converta os dados do aplicativo.

A conversão é executada no gerenciador de filas de envio para determinados formatos integrados e para formatos definidos pelo usuário se uma saída de usuário adequada for fornecida.

Formatos integrados

Isso inclui:

- As mensagens que sejam inteiramente caracteres (usando o nome de formato MQFMT_STRING)
- Mensagens definidas pelo IBM MQ, por exemplo formatos de comando programáveis

  O IBM MQ usa mensagens de formato de comando programável para mensagens de administração e eventos (o nome do formato MQFMT_ADMIN é usado neste caso). É possível usar o mesmo formato (usando o nome do formato MQFMT_PCF) para suas próprias mensagens e tirar proveito da conversão de dados integrados.

Os formatos integrados do gerenciador de filas todos possuem nomes que começam com MQFMT. Eles são listados e descritos em Formato.

Formatos definidos pelo aplicativo

Para formatos definidos pelo usuário, a conversão de dados do aplicativo deve ser executada por um programa de saída de conversão de dados (para obter informações adicionais, consulte “Escrevendo saídas de conversão de dados” na página 995). Em um ambiente de cliente/servidor, a saída é carregada no servidor e a conversão ocorre ali.

Conversão no gerenciador de filas de recebimento

Dados da mensagem do aplicativo podem ser convertidos pelo gerenciador de filas de recebimento para ambos formatos, integrados e definidos pelo usuário.

A conversão é desempenhada durante o processamento de uma chamada MQGET se for especificada a opção MQGMO_CONVERT. Para obter detalhes, consulte as Opções

Conjuntos de caracteres codificados

Produtos IBM MQ suportam os conjuntos de caracteres codificados que são fornecidos pelo sistema operacional subjacente.

Ao criar um gerenciador de filas, o ID do conjunto de caracteres codificados do gerenciador de filas (CCSID) usado será baseado naquele do ambiente subjacente. Se esta é uma página de códigos mistos, o IBM MQ usa a parte SBCS da página de códigos mistos como o gerenciador de filas CCSID.

Para conversão de dados gerais, se o sistema operacional subjacente suportar páginas de código DBCS, o IBM MQ pode usá-las.

Consulte a documentação de seu sistema operacional para obter detalhes dos conjuntos de caracteres codificados que ele suporta.

É necessário considerar a conversão de dados do aplicativo, os nomes de formato, e saídas de usuário ao gravar aplicativos que abrangem várias plataformas. Consulte “Escrevendo saídas de conversão de dados” na página 995 para obter informações sobre como chamar e gravar saídas de conversão de dados.

Prioridades de mensagens

É possível configurar a prioridade de mensagem para um valor numérico ou deixar a mensagem levar a prioridade padrão da fila.

Configure a prioridade de uma mensagem (no campo *Priority* da estrutura MQMD) quando você colocar a mensagem em uma fila. É possível configurar um valor numérico para a prioridade ou deixar a mensagem obter a prioridade padrão da fila.

O atributo **MsgDeliverySequence** da fila determina se as mensagens na fila são armazenadas na sequência FIFO (first in, first out) ou em FIFO dentro da sequência de prioridade. Se este atributo estiver configurado como MQMDS_PRIORITY, as mensagens serão enfileiradas com a prioridade especificada no campo *Priority* de seus descritores de mensagens; mas se ele estiver configurado como MQMDS_FIFO, as mensagens serão enfileiradas com a prioridade padrão da fila. As mensagens de prioridade igual são armazenadas na fila em ordem de chegada.

O atributo **DefPriority** de uma fila configura o valor da prioridade padrão para mensagens que estão sendo colocadas nessa fila. Esse valor é configurado quando a fila é criada, mas pode ser mudado posteriormente. Filas de alias e definições locais de filas remotas podem ter prioridades padrão diferentes das filas de base para a qual elas resolvem. Se houver mais de uma definição de fila no caminho de resolução (consulte “Resolução do Nome” na página 756), a prioridade padrão será obtida do valor (no momento da operação put) do atributo **DefPriority** da fila especificada no comando open.

O valor do atributo **MaxPriority** do gerenciador de filas é a prioridade máxima que pode ser designada a uma mensagem processada por esse gerenciador de filas. Não é possível mudar o valor desse atributo. No IBM MQ, o atributo tem o valor 9; é possível criar mensagens com prioridades entre 0 (a mais baixa) e 9 (a mais alta).

Propriedades da Mensagem

Use propriedades de mensagem para permitir que um aplicativo selecione mensagens para processar ou recuperar informações sobre uma mensagem sem acessar cabeçalhos MQMD ou MQRFH2. Elas também facilitam a comunicação entre aplicativos IBM MQ e JMS.

Uma propriedade de mensagem é dados associados a uma mensagem, consistindo em um nome textual e um valor de um tipo específico. Propriedades de mensagens são usadas pelos seletores de mensagens para filtrar publicações para tópicos ou para obter seletivamente as mensagens das filas. Propriedades de mensagens podem ser usadas para incluir dados de negócios ou informações de estado sem precisar armazená-las nos dados do aplicativo. Os aplicativos não precisam acessar dados no MQ Message Descriptor (MQMD) ou nos cabeçalhos MQRFH2 porque os campos nessas estruturas de dados podem ser acessados como propriedades de mensagem usando chamadas de função Message Queue Interface (MQI).

O uso de propriedades de mensagem no IBM MQ imita o uso de propriedades no JMS. Isso significa que é possível configurar propriedades em um aplicativo JMS e recuperá-las em um aplicativo processual IBM MQ ou o contrário. Para disponibilizar uma propriedade para um aplicativo JMS, designe a ela o prefixo "usr"; ela estará, então, disponível (sem o prefixo) como uma propriedade de usuário de mensagem do JMS. Por exemplo, a propriedade do IBM MQ *usr.myproperty* (uma sequência de caracteres) está acessível para um aplicativo JMS usando a chamada do JMS `message.getStringProperty('myproperty')`. Observe que os aplicativos JMS são incapazes de acessar propriedades com o prefixo "usr" se contiverem um ou mais caracteres U+002E ("."). Uma propriedade sem prefixo e nenhum caractere U+002E (".") é tratada como se tivesse o prefixo "usr". Por outro lado, uma propriedade de usuário configurada em um aplicativo JMS pode ser acessada em um aplicativo IBM MQ incluindo o prefixo "usr." no nome da propriedade consultada em uma chamada MQINQMP.

Propriedades de mensagens e comprimento de mensagens

Utilize o atributo do gerenciador de filas *MaxPropertiesLength* para controlar o tamanho das propriedades que podem fluir com qualquer mensagem em um gerenciador de filas do IBM MQ.

Em geral, ao usar MQSETMP para configurar as propriedades, o tamanho de uma propriedade é o comprimento do nome da propriedade em bytes, mais o comprimento do valor da propriedade em bytes, conforme passado na chamada MQSETMP. É possível que o conjunto de caracteres do nome da propriedade e o valor da propriedade mudem durante a transmissão da mensagem para seu destino porque podem ser convertidos em Unicode; neste caso o tamanho da propriedade pode ser mudado.

Em uma chamada MQPUT ou MQPUT1, as propriedades da mensagem não contam para o comprimento da mensagem para a fila e o gerenciador de filas, mas contam para o comprimento das propriedades conforme observado pelo gerenciador de filas (se elas foram configuradas usando a propriedade de mensagem de chamadas MQI ou não).

Se o tamanho das propriedades exceder o comprimento máximo de propriedades, a mensagem será rejeitada com MQRC_PROPERTIES_TOO_BIG. Como o tamanho das propriedades depende de sua representação, é necessário configurar o comprimento máximo das propriedades em um nível bruto.

É possível para um aplicativo para colocar com sucesso uma mensagem com um buffer que seja maior que o valor de *MaxMsgLength*, se o buffer incluir propriedades. Isso ocorre porque, mesmo quando representadas como elementos MQRFH2, as propriedades da mensagem não contam para o comprimento da mensagem. Os campos de cabeçalho MQRFH2 incluem o comprimento de propriedades apenas se uma ou mais pastas estiverem contidas e cada pasta no cabeçalho contiver propriedades. Se uma ou mais pastas estiverem contidas no cabeçalho MQRFH2 e qualquer pasta não contiver propriedades, os campos de cabeçalho MQRFH2 contam para o comprimento da mensagem.

Em uma chamada MQGET, as propriedades da mensagem não contam para o comprimento da mensagem com relação à fila e ao gerenciador de filas. No entanto, como as propriedades são contadas separadamente, é possível que o buffer retornado por uma chamada MQGET seja maior que o valor do atributo *MaxMsgLength*.

Não faça seus aplicativos consultarem o valor de *MaxMsgLength* e, em seguida, alocar um buffer desse tamanho antes da chamada MQGET; em vez disso, aloque um buffer que você considere grande o suficiente. Se MQGET falhar, aloque um buffer guiado pelo tamanho do parâmetro *DataLength*.

O parâmetro *DataLength* da chamada MQGET retorna o comprimento em bytes dos dados do aplicativo e quaisquer propriedades retornadas no buffer que você forneceu, se um identificador de mensagem não for especificado na estrutura MQGMO.

O parâmetro *Buffer* da chamada MQPUT contém os dados da mensagem do aplicativo a ser enviada e quaisquer propriedades representadas nos dados da mensagem.

Há um limite de comprimento de 100 MB para propriedades de mensagem, excluindo o descritor de mensagens ou extensão para cada mensagem.

O tamanho de uma propriedade em sua representação interna é o comprimento do nome, mais o tamanho de seu valor, além de alguns dados de controle para a propriedade. Há também alguns dados de controle para o conjunto de propriedades após uma propriedade ser incluída na mensagem.

Nomes de propriedades

Um nome da propriedade é uma sequência de caracteres. Determinadas restrições se aplicam a seu comprimento e ao conjunto de caracteres que podem ser usados.

Um nome da propriedade é uma sequência de caracteres com distinção entre maiúsculas e minúsculas, limitado a +4095 caracteres, a menos que restrito de outra forma pelo contexto. Este limite está contido na constante MQ_MAX_PROPERTY_NAME_LENGTH.

Se você exceder esse comprimento máximo ao usar uma chamada MQI propriedade de mensagem, a chamada falhará com o código de razão MQRC_PROPERTY_NAME_LENGTH_ERR.

Como não há comprimento máximo de nome de propriedade no JMS, é possível para um aplicativo JMS configurar um nome de propriedade válido do JMS que não seja um nome de propriedade válido do IBM MQ quando armazenado em uma estrutura MQRFH2.

Neste caso, quando analisado, apenas os primeiros 4095 caracteres do nome da propriedade são usados; os caracteres a seguir são truncados. Isso poderá fazer com que um aplicativo que está usando seletores falhe em corresponder a uma sequência de seleção ou em corresponder a uma sequência quando não esperado, uma vez que mais de uma propriedade pode ser truncada para o mesmo nome. Quando um nome de propriedade é truncado, o WebSphereMQ emite uma mensagem de log de erros.

Todos os nomes de propriedades devem seguir as regras definidas pela Especificação de linguagem de Java para Identificadores do Java, com a exceção de que o caractere Unicode U+002E (.) é permitido como parte do nome, mas não como início. As regras para Identificadores do Java são iguais àquelas contidas na especificação do JMS para nomes de propriedades.

Os caracteres de espaço em branco e operadores de comparação são proibidos. Os nulos integrados são permitidos em um nome de propriedade, mas não é recomendado. Se você usar os nulos integrados, isso impede o uso da constante MQVS_NULL_TERMINATED quando usada com a estrutura MQCHARV para especificar sequências de comprimento variável.

Mantenha os nomes de propriedade simples, porque os aplicativos podem selecionar as mensagens com base nos nomes de propriedade e a conversão entre o conjunto de caracteres do nome e do seletor pode fazer com que a seleção falhe inesperadamente.

Os nomes de propriedades do IBM MQ usam o caractere U+002E (.) para agrupamento lógico de propriedades. Isso divide o namespace para as propriedades. Propriedades com os prefixos a seguir, em qualquer combinação de minúsculas ou maiúsculas são reservadas para uso pelo produto:

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

Uma boa maneira de evitar conflitos de nomes é assegurar que todos os aplicativos prefixem suas propriedades de mensagem com seu nome de domínio da Internet. Por exemplo, se estiver desenvolvendo um aplicativo usando o nome de domínio `ourcompany.com`, deverá denominar todas as propriedades com o prefixo `com.ourcompany`. Esta convenção de nomenclatura também permite a seleção fácil de propriedades; por exemplo, um aplicativo pode consultar sobre todas as propriedades de mensagem iniciando `com.ourcompany.%`.

Consulte Restrições de nomes de propriedades para obter informações adicionais sobre o uso de nomes de propriedades.

Restrições de nome da propriedade
Ao nomear uma propriedade, deve-se observar certas regras.

As restrições a seguir se aplicam aos nomes de propriedade:

1. Uma propriedade não deve começar com as seguintes sequências:
   - "JMS" - reservado para uso pelo IBM MQ classes for JMS.
   - "usr.JMS" - não é válido.

   As únicas exceções são as seguintes propriedades fornecendo sinônimos para propriedades JMS:

Propriedade	**Sinônimo para**
JMSCorrelationID	Root .MQMD.CorrelId ou jms.Cid
JMSDeliveryMode	Root .MQMD.Persistence ou jms.Dlv

Propriedade	Sinônimo para
JMSDestination	jms.Dst
JMSExpiration	Root .MQMD.Expiry ou jms.Exp
JMSMessageID	Root .MQMD.MsgId
JMSPriority	Root .MQMD.Priority ou jms.Pri
JMSRedelivered	Root .MQMD.BackoutCount
JMSReplyTo (uma sequência codificada como um URI)	Root .MQMD.ReplyToQ ou Root .MQMD.ReplyToQMgr ou jms.Rto
JMSTimestamp	Root .MQMD.PutDate ou Root .MQMD.PutTime ou jms.Tms
JMSType	mcd.Type ou mcd.Set ou mcd.Fmt
JMSXAppID	Root .MQMD.PutApplName
JMSXDeliveryCount	Root .MQMD.BackoutCount
JMSXGroupID	Root .MQMD.GroupId ou jms.Gid
JMSXGroupSeq	Root .MQMD.MsgSeqNumber ou jms.Seq
JMSXUserID	Root .MQMD.UserIdentifier

Esses sinônimos permitem que um aplicativo MQI acesse as propriedades do JMS de forma semelhante ao aplicativo cliente do IBM MQ classes for JMS. Dessas propriedades, somente JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID e JMSXGroupSeq podem ser configurados usando o MQI.

Observe que as propriedades JMS_IBM_* disponíveis a partir do IBM MQ classes for JMS não estão disponíveis usando o MQI. Os campos que as propriedades JMS_IBM_* referenciam podem ser acessados de outras maneiras por aplicativos MQI.

2. Uma propriedade não deve ser chamada, em qualquer combinação de letras maiúsculas ou minúsculas, "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" e "ESCAPE". Esses são os nomes das palavras-chave SQL usadas em sequências de seleção.
3. Um nome da propriedade iniciada em " mq " em qualquer mistura de minúsculas ou maiúsculas e não iniciada em "mq_usr" pode conter apenas um "." caractere (U+002E). Vários "." caracteres não são permitidos em propriedades com esses prefixos.
4. Dois "." caracteres devem conter outros caracteres entre eles; não é possível ter um ponto vazio na hierarquia. Semelhantemente, nome de propriedade não pode terminar com um "." "."
5. Se um aplicativo configura a propriedade "a.b" e, em seguida, a propriedade "a.b.c", não estará claro se na hierarquia "b" contém um valor ou outro agrupamento lógico. Essa hierarquia é "conteúdo misto " e isso não é suportado. Configurando uma propriedade que causa com que o conteúdo misto não seja permitido.

Estas restrições são impingidas pelo mecanismo de validação conforme segue:

- Os nomes de propriedade são validados quando configurando uma propriedade usando a chamada MQSETMP-Configurar propriedade de mensagem se uma validação tiver sido solicitada quando a manipulação de mensagem foi criada. Se feita uma tentativa de validar uma propriedade for empreendida e falhar devido a um erro na especificação do nome da propriedade, o código de conclusão será MQCC_FAILED com a razão:
  - MQRC_PROPERTY_NAME_ERROR para razões 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED para a razão 5.
- Os nomes de propriedades especificadas diretamente como elementos MQRFH2 não são garantidas ao ser validadas pela chamada MQPUT.

Campos do descritor de mensagens como propriedades
A maioria dos campos do descritor de mensagens pode ser tratada como propriedades. O nome da propriedade é construído pela inclusão de um prefixo no nome do campo descritor de mensagens.

Se um aplicativo MQI deseja identificar uma propriedade de mensagem contida em um campo do descritor de mensagens, por exemplo, em uma sequência do seletor ou usando as APIs de propriedade da mensagem, use a seguinte sintaxe:

Nome da Propriedade	Campo do descritor de mensagens
Root.MQMD.*Field*	*Campo*

Especifique *Field* com o mesmo caso que para os campos de estrutura do MQMD na declaração de linguagem C. Por exemplo, o nome de propriedade `Root.MQMD.AccountingToken` acessa o campo `AccountingToken` do descritor de mensagens.

Os campos `StrucId` e `Version` do descritor de mensagens não estão acessíveis usando a sintaxe mostrada.

Campos do descritor de mensagens nunca são representados em um cabeçalho MQRFH2 como para outras propriedades.

Se os dados da mensagem começarem com um MQMDE que é atendido pelo gerenciador de filas, os campos MQMDE poderão ser acessados usando a notação `Root.MQMD.`*`Field`* descrita. Neste caso, os campos MQMDE são tratados como parte, logicamente, do MQMD a partir de uma perspectiva de propriedades. Consulte Visão Geral de MQMDE.

Tipos e valores de dados de propriedades
Uma propriedade pode ser um booleano, uma sequência de bytes, uma sequência de caracteres ou um número de vírgula flutuante ou inteiro. A propriedade pode armazenar qualquer valor válido no intervalo do tipo de dados, a menos que restringido de outra forma pelo contexto.

O tipo de dados de um valor de propriedade deve ser um dos valores a seguir:

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

Uma propriedade pode existir, mas não têm nenhum valor definido; é uma propriedade nula. Uma propriedade nula é diferente de uma propriedade de byte (MQBYTE[ ]) ou propriedade sequência de caracteres (MQCHAR[ ]), pois tem um valor definido, mas vazio, ou seja, um valor de comprimento zero.

Sequência de bytes não é um tipo de dados da propriedade válida no JMS ou XMS. Você é aconselhado a não usar as propriedades da sequência de bytes na pasta *usr*.

Selecionando mensagens nas filas

É possível selecionar mensagens nas filas usando os campos MsgId e CorrelId em uma chamada MQGET ou usando uma SelectionString em uma chamada MQOPEN ou MQSUB.

Seletores
Um seletor de mensagem é uma sequência de comprimento variável usada por um aplicativo para registrar seu interesse apenas naquelas mensagens que têm propriedades que satisfazem a consulta Linguagem de Consulta Estruturada (SQL) que a sequência de seleção representa.

Seleção usando as chamadas de função MQSUB e MQOPEN

É possível usar *SelectionString*, que é uma estrutura de tipo MQCHARV, para fazer seleções usando as chamadas MQSUB e MQOPEN.

A estrutura de *SelectionString* é usada para passar uma sequência de seleção de comprimento variável para o gerenciador de filas.

O CCSID associado à sequência do seletor é configurado por meio do campo VSCCSID da estrutura MQCHARV. O valor usado deve ser um CCSID que é suportado para sequências de seletor. Consulte Conversão de página de códigos para obter uma lista de páginas de códigos suportadas.

Especificar um CCSID para o qual não há suporte conversão de Unicode suportada pelo IBM MQ, resulta em um erro de MQRC_SOURCE_CCSID_ERROR. Esse erro é retornado no momento em que o seletor é apresentado ao gerenciador de filas, isso é, na chamada MQSUB, MQOPEN ou MQPUT1.

O valor padrão para o campo *VSCCSID* é MQCCSI_APPL, o que indica que o CCSID da sequência de seleção é igual ao CCSID do gerenciador de filas ou ao CCSID do cliente, se estiver conectado por meio de um cliente. A constante MQCCSI_APPL pode, no entanto, ser substituída por um aplicativo que a esteja redefinindo antes da compilação.

Se o seletor MQCHARV representa uma sequência NULL, nenhuma seleção ocorre para esse consumidor de mensagens e as mensagens serão entregues como se um seletor não tivesse sido usado.

O comprimento máximo de uma sequência de seleção é limitado somente por aquilo que pode ser descrito pelo campo MQCHARV *VSLength*.

SelectionString é retornado na saída de uma chamada MQSUB usando a opção de assinatura MQSO_RESUME, se você tiver fornecido um buffer e houver um comprimento de buffer positivo em VSBufSize. Se você não fornecer um buffer, somente o comprimento da sequência de seleção será retornado no campo VSLength do MQCHARV. Se o buffer fornecido for menor que o espaço necessário para retornar o campo, somente VSBufSize bytes serão retornados no buffer fornecido.

Um aplicativo não pode mudar uma sequência de seleção sem antes fechar o identificador para a fila (para MQOPEN) ou assinatura (para MQSUB). Uma nova seleção de sequência pode ser especificada em uma chamada MQOPEN ou MQSUB subsequente.

MQOPEN
: Use MQCLOSE para fechar o identificador aberto, em seguida, especifique uma sequência de seleção nova em uma chamada MQOPEN subsequente.

MQSUB
: Use MQCLOSE para fechar o identificador de assinatura retornada (hSub) e, em seguida, especifique uma sequência de seleção nova em uma chamada MQSUB subsequentes.

Figura 3 na página 33 mostra o processo de seleção usando a chamada MQSUB.

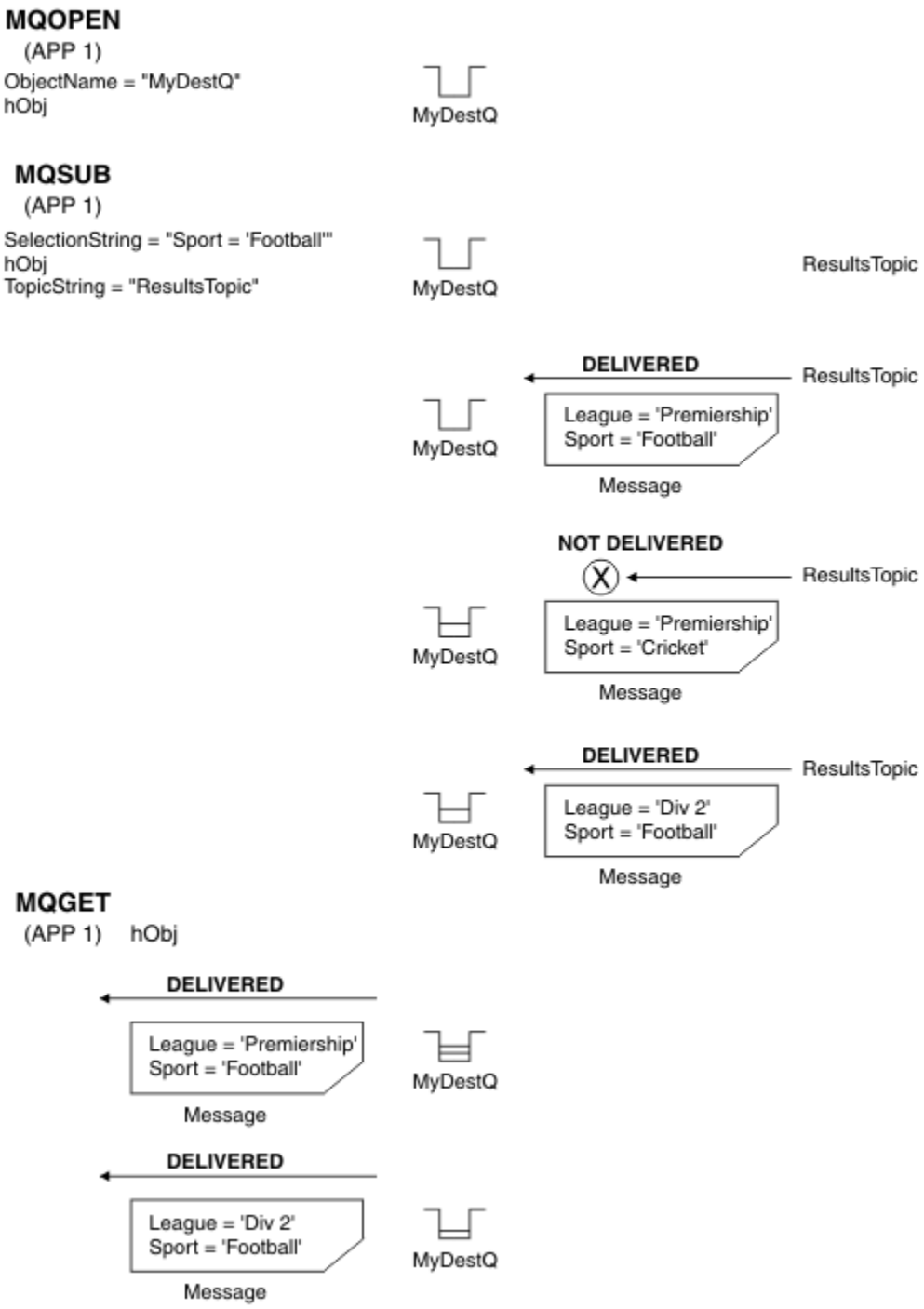


Figura 3. Seleção usando a chamada MQSUB

Um seletor pode ser passado na chamada para MQSUB usando o campo *SelectionString* na estrutura MQSD. O efeito de passar um seletor no MQSUB é que somente as mensagens publicadas para o tópico que está sendo assinado, que correspondem a uma sequência de seleção fornecida, serão disponibilizadas na fila de destino.

Figura 4 na página 34 mostra o processo de seleção usando a chamada MQOPEN.

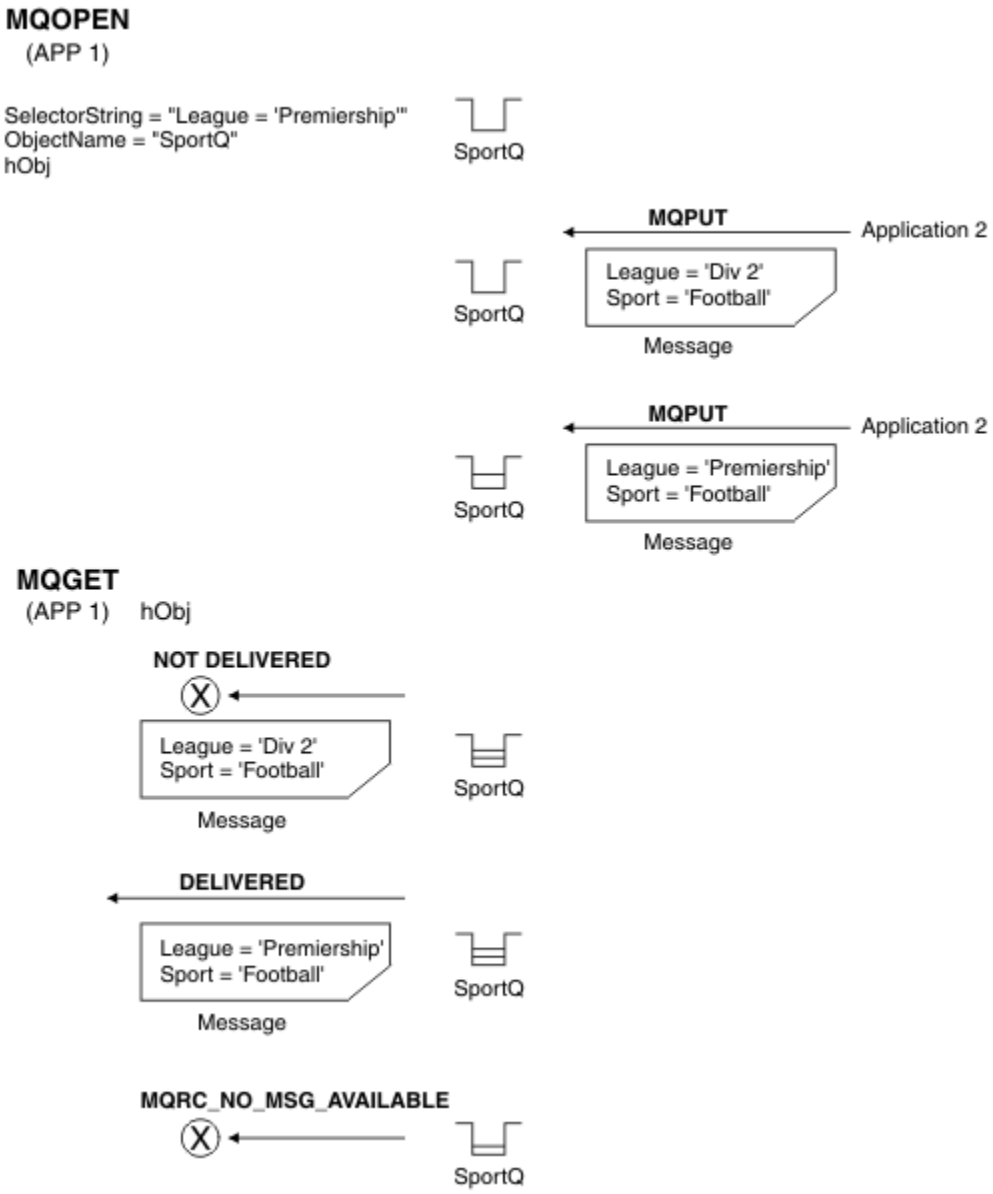


Figura 4. Seleção usando a chamada MQOPEN

Um seletor pode ser passado na chamada para MQOPEN usando o campo *SelectionString* na estrutura MQOD. O efeito de passar um seletor na chamada MQOPEN é que somente as mensagens na fila aberta, que correspondem a um seletor, serão entregues ao consumidor de mensagens.

O uso principal para o seletor na chamada MQOPEN é para o caso ponto a ponto em que um aplicativo pode optar por receber em uma fila somente as mensagens que correspondem a um seletor. O exemplo anterior mostra um cenário simples em que duas mensagens são colocadas em uma fila aberta por MQOPEN, mas somente uma é recebida pelo aplicativo que a está obtendo, uma vez que ele é o único que corresponda a um seletor.

Observe que chamadas MQGET subsequentes resultam em MQRC_NO_MSG_AVAILABLE, pois nenhuma mensagem adicional existe na fila que corresponde ao seletor fornecido.

Conceitos relacionados

“Regras e restrições de sequência de seleção” na página 41
Familiarize-se com essas regras sobre como as sequências de seleção são interpretadas e as restrições de caracteres para evitar problemas em potencial ao usar seletores.

Comportamento de seleção

Visão geral do comportamento de seleção do IBM MQ.

Os campos em uma estrutura MQMDE são considerados como propriedades de mensagem para as propriedades do descritor de mensagens correspondentes se o MQMD:

- Tiver o formato MQFMT_MD_EXTENSION
- For seguido imediatamente por uma estrutura MQMDE válida
- For uma versão ou contiver a versão padrão de dois campos somente

É possível que uma sequência de seleção seja resolvida para TRUE ou FALSE antes de ocorrer qualquer correspondência com relação a propriedades de mensagens. Por exemplo, pode ser o caso se a sequência de caracteres de seleção for configurada como `"TRUE <> FALSE"`. Essa avaliação inicial é garantida que ocorre somente quando não houver referências de propriedades de mensagem na sequência de seleção.

Se uma sequência de seleção for resolvida para TRUE antes de quaisquer propriedades de mensagem serem consideradas, todas as mensagens publicadas no tópico assinado pelo consumidor serão entregues. Se uma sequência de seleção for resolvida para FALSE antes de quaisquer propriedades de mensagem serem consideradas, um código de razão MQRC_SELECTOR_ALWAYS_FALSE e o código de conclusão MQCC_FAILED serão retornados na chamada de função que apresentou o seletor.

Mesmo se uma mensagem não contiver nenhuma propriedade de mensagem (além de propriedades de cabeçalho), ela ainda pode estar elegível para seleção. Se uma sequência de seleção fizer referência a uma propriedade de mensagem que não existe, essa propriedade será assumida como tendo o valor NULL ou 'Unknown'.

Por exemplo, uma mensagem ainda pode satisfazer uma sequência de seleção como `'Color IS NULL'`, em que `'Color'` não existe como uma propriedade de mensagem na mensagem

A seleção pode ser executada somente nas propriedades associadas a uma mensagem, não à própria mensagem, a menos que um provedor de seleção de mensagem estendida esteja disponível. A seleção poderá ser executada na carga útil da mensagem apenas se um provedor de seleção de mensagem estendida estiver disponível.

Cada propriedade de mensagem tem um tipo associado a ele. Ao executar uma seleção, deve-se assegurar que os valores usados em expressões para testar as propriedades de mensagens são do tipo correto. Se ocorrer uma incompatibilidade de tipos, a expressão em questão será resolvida para FALSE.

É de sua responsabilidade assegurar que a sequência de seleção e as propriedades de mensagem usem tipos compatíveis.

Critérios de seleção continuam a ser aplicados em nome de assinantes duráveis inativos, de modo que somente as mensagens que correspondem à sequência de seleção que foi originalmente fornecida serão mantidas.

Sequências de seleção não podem ser mudadas quando uma assinatura durável é continuada com alteração (MQSO_ALTER). Se uma sequência de seleção diferente for apresentada quando um assinante durável continuar a atividade, então, MQRC_SELECTOR_NOT_ALTERABLE será retornado ao aplicativo.

Os aplicativos recebem um código de retorno de MQRC_NO_MSG_AVAILABLE se não houver mensagem em uma fila que atenda aos critérios de seleção.

Se um aplicativo tiver especificado uma sequência de seleção contendo valores de propriedades, somente aquelas mensagens que contêm propriedades correspondentes serão elegíveis para seleção. Por exemplo, um assinante especifica uma sequência de caracteres de seleção de "a = 3" e uma mensagem é publicada contendo nenhuma propriedade ou propriedades em que 'a' não existe ou não é igual a 3. O assinante não recebe essa mensagem para sua fila de destino.

Desempenho de mensagens

Selecionar mensagens de uma fila requer que o IBM MQ inspecione sequencialmente cada mensagem na fila. As mensagens são inspecionadas até que uma mensagem seja localizada que corresponda aos

critérios de seleção ou não haja mais mensagens para examinar. Portanto, o desempenho do sistema de mensagens será prejudicado se a seleção de mensagem for usada em filas profundas.

Para otimizar a seleção de mensagem em filas profundas quando a seleção se baseia em JMSCorrelacionID ou JMSMessageID, use uma sequência de seleção no formato:

- JMSCorrelationID ='ID:*correlation_id*'
- JMSMessageID='ID:*message_id*'

em que:

- *correlation_id* é uma sequência que contém um identificador de correlação padrão do IBM MQ.
- *message_id* é uma sequência que contém um identificador de mensagens padrão do IBM MQ.

Nota: O seletor deve fazer referência apenas a uma das propriedades. O uso de um seletor que tem um desses formatos oferece uma melhoria significativa no desempenho ao selecionar JMSCorrelationID e uma melhoria de desempenho marginal para JMSMessageID. Para obter informações adicionais, consulte "Seletores de mensagens no JMS" na página 146.

Usando seletores complexos

Os seletores podem conter vários componentes, por exemplo:

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

O uso de tais seletores complexos pode ter sérias implicações no desempenho e requisitos de recurso excessivos. Dessa forma, o IBM MQ protegerá o sistema deixando de processar seletores muito complexos que poderiam resultar em uma falta de recursos do sistema. A proteção pode ocorrer em sequências de seleção que contêm mais de 100 testes ou quando o IBM MQ detecta que o limite para o tamanho da pilha do sistema operacional está sendo atingido. É necessário tentar e testar completamente o uso de sequências de seleção com muitos componentes, nas plataformas adequadas, para assegurar que os limites de proteção não sejam atingidos.

O desempenho e a complexidade de seletores podem ser melhorados simplificando-os usando parênteses adicionais para combinar componentes. Por exemplo:

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

Conceitos relacionados

"Regras e restrições de sequência de seleção" na página 41
Familiarize-se com essas regras sobre como as sequências de seleção são interpretadas e as restrições de caracteres para evitar problemas em potencial ao usar seletores.

Sintaxe do seletor de mensagem

Um seletor de mensagem do IBM MQ é uma sequência com sintaxe que é baseada em um subconjunto da sintaxe de expressão condicional SQL92.

A ordem na qual um seletor de mensagem é avaliado é da esquerda para a direita dentro de um nível de precedência. É possível usar parênteses para mudar essa ordem. Literais do seletor e nomes de operadores predefinidos estão gravados aqui em maiúsculas; no entanto, não fazem distinção entre maiúsculas e minúsculas.

Se o seletor for fornecido por meio da API, o IBM MQ verificará a correção sintática de um seletor de mensagens no momento em que ele for apresentado. Se a sintaxe da sequência de seleção estiver incorreta ou um nome de propriedade não for válido, e um provedor de seleção de mensagens estendido não estiver disponível, `MQRC_SELECTION_NOT_AVAILABLE` será retornado ao aplicativo. Se a sintaxe da sequência de seleção estiver incorreta ou um nome de propriedade não for válido quando uma assinatura for retomada, um `MQRC_SELECTOR_SYNTAX_ERROR` será retornado ao aplicativo. Se a validação do nome da propriedade foi desativada quando a propriedade foi configurada (configurando `MQCMHO_NONE` em vez de `MQCMHO_VALIDATE`) e um aplicativo subsequentemente coloca uma mensagem com um nome de propriedade inválido, esta mensagem nunca será selecionada.

Nenhum erro será retornado no momento em que o seletor for apresentado se o IBM MQ determinar que um seletor de assinatura definido administrativamente está usando a sintaxe de mensagem estendida, conforme indicado pelo **DISPLAY SUB** parâmetro **SELTYPE** tendo o valor EXTENDED. Nesse caso, a verificação da sintaxe da sequência de seleção será deferida até o tempo de publicação (consulte MQRC_SELECTION_NOT_AVAILABLE).

Um seletor pode conter:

- Literais:
  - Literais de sequência são colocados entre aspas simples. Duas aspas simples consecutivas representam uma aspa simples. Os exemplos são: 'literal' e 'literal''s'. Como literais de sequência do Java, eles usam a codificação de caracteres Unicode. Não é possível usar aspas duplas para delimitar um literal de sequência. Qualquer sequência de bytes pode ser usada entre as aspas simples.
  - Uma sequência de bytes é um ou mais pares de caracteres hexadecimais colocados entre aspas duplas e com o prefixo 0x. Os exemplos são "0x2F1C" ou "0XD43A". O comprimento de uma sequência de bytes deve ser de pelo menos um byte. Se uma sequência de bytes do seletor tiver correspondência com uma propriedade de mensagem do tipo MQTYPE_BYTE_STRING, nenhuma ação especial será executada no zero à esquerda ou à direita. Os bytes são tratados como outro caractere. Endianness também não é considerado. O comprimento de ambas as sequências de bytes do seletor e da propriedade deve ser igual e a sequência de bytes deve ser a mesma.

    Exemplos de seleções de sequências de bytes (suponha que *myBytes* = 0AFC23) que têm correspondência são:

    - "myBytes = "0x0AFC23"" = TRUE

    As seguintes seleções de sequência não têm correspondência:

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (porque o número de bytes não é múltiplo de dois)
    - "myBytes = "0x0AFC2300"" = FALSE (porque o zero à direita é significativo na comparação)
    - "myBytes = "0x000AFC23"" = FALSE (porque o zero à esquerda é significativo na comparação)
    - "myBytes = "0x23FC0A"" = FALSE (porque endianness não é considerado)
  - Os números hexa começam com um zero, seguido por um x maiúsculo ou minúsculo. O restante do literal contém um ou mais caracteres hexa válidos. Os exemplos são 0xA, 0xAF, 0X2020.
  - Um zero à esquerda seguido por um ou mais dígitos no intervalo 0-7 é sempre interpretado como sendo o início de um número octal. Não é possível representar um número decimal com prefixo zero dessa forma, por exemplo, 09 retorna um erro de sintaxe, pois 9 não é um dígito octal válido. Exemplos de números octais são 0177, 0713.
  - Um literal numérico exato é um valor numérico sem um ponto decimal, como 57, -957 e +62. Um literal numérico exato pode ter um L maiúsculo ou minúsculo à direita; isso não afeta como o número é armazenado ou interpretado. O IBM MQ suporta numerais exatos no intervalo -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.
  - Um literal numérico aproximado é um valor numérico em notação científica, como 7E3 ou -57.9E2, ou um valor numérico com um decimal, como 7., -95.7ou +6.2. O IBM MQ suporta números no intervalo -1.797693134862315E+308 a 1.797693134862315E+308.

    O significando deve seguir um caractere de sinal opcional (+ ou -). O significando deve ser um número inteiro ou uma fração. Uma parte fracionária do significando não precisa ter um dígito inicial.

    Um E maiúsculo ou minúsculo indica o início de um expoente opcional. O expoente possui um radix decimal e a parte do número do expoente pode ser prefixada por um caractere de sinal opcional.

    Literais numéricos aproximados podem ser finalizados por um caractere F ou D (não fazem distinção entre maiúsculas e minúsculas). Essa sintaxe existe para suportar o método de linguagem cruzada de identificação de números de precisão simples ou duplos. Esses caracteres são opcionais e não afetam como um literal numérico aproximado é armazenado ou processado. Esses números são sempre armazenados e processados usando precisão dupla.

– Os literais booleanos TRUE e FALSE.

Nota: As representações IEEE-754 não finitas, como `NaN`, `+Infinity`, `-Infinity`, não são suportadas nas sequências de seleção. Portanto, não é possível usar esses valores como operandos em uma expressão. Zero negativo é tratado da mesma maneira que zero positivo para operações matemáticas.

- Identificadores:

  Um identificador é uma sequência de caracteres de comprimento variável que deve começar com um caractere inicial identificador válido, seguido por zero ou mais caracteres de partes do identificador válidos. As regras para nomes de identificador são iguais àquelas para nomes de propriedades de mensagens, consulte “Nomes de propriedades” na página 28 e “Restrições de nome da propriedade” na página 29 para obter informações adicionais.

  Nota: A seleção poderá ser executada na carga útil da mensagem apenas se um provedor de seleção de mensagem estendida estiver disponível.

  Identificadores são referências de campo de cabeçalho ou referências de propriedade. O tipo de um valor de propriedade em um seletor de mensagens deve corresponder ao tipo usado para configurar a propriedade, embora a promoção numérica seja executada quando possível. Se ocorrer uma incompatibilidade de tipos, então, o resultado da expressão será FALSE. Se uma propriedade que não existe em uma mensagem for referenciada, seu valor será `NULL`.

  Conversões de tipo que se aplicam aos métodos get para propriedades não se aplicam quando uma propriedade é usada em uma expressão do seletor de mensagem. Por exemplo, se você configurar uma propriedade como um valor de sequência e, em seguida, usar um seletor para consultá-lo como um valor numérico, a expressão retornará FALSE.

  Os nomes de propriedade e campo do JMS que são mapeados para os nomes de propriedade ou nomes de campo `MQMD` também são identificadores válidos em uma sequência de seleção. O IBM MQ mapeia os nomes de propriedade e campo do JMS reconhecidos para os valores de propriedade de mensagem. Consulte a “Seletores de mensagens no JMS” na página 146 para obter mais informações. Como um exemplo, a sequência de seleção `"JMSPriority >="` seleciona na propriedade `Pri` localizada na pasta `jms` da mensagem atual.

- Estouro/estouro negativo:

  Para ambos os números, decimal e numérico aproximado, as opções a seguir são indefinidas:

  – Especificar um número que está fora do intervalo definido
  – Especificar uma expressão aritmética que pode causar estouro ou estouro negativo

  Nenhuma verificação é executada para essas condições.

- Espaço em branco:

  Definido como um espaço, alimentação de formulário, nova linha, retorno de linha, tabulação horizontal ou vertical. Os seguintes caracteres Unicode são reconhecidos como espaço em branco:

  – `\u0009 to \u000D`
  – `\u0020`
  – `\u001C`
  – `\u001D`
  – `\u001E`
  – `\u001F`
  – `\u1680`
  – `\u180E`
  – `\u2000` a `\u200A`
  – `\u2028`
  – `\u2029`

  - \u202F
  - \u205F
  - \u3000
- Expressões:
  - Um seletor é uma expressão condicional. Um seletor avaliado como true tem correspondência; um seletor avaliado como false ou unknown não tem correspondência.
  - As expressões aritméticas são compostas por si mesmas, por operações aritméticas, por identificadores (o valor do identificador é tratado como um literal numérico) e por literais numéricos.
  - Expressões condicionais são compostas por si mesmas, por operações de comparação e por operações lógicas.
- O uso padrão de parênteses () para configurar a ordem na qual as expressões são avaliadas é suportado.
- Os operadores lógicos em ordem de precedência: NOT, AND, OR.
- Operadores de comparação: =, >, >=, <, <=, <> (diferente).
  - Sequências de dois bytes são iguais apenas se as sequências tiverem o mesmo comprimento e a sequência de bytes for igual.
  - Somente valores do mesmo tipo podem ser comparados. Uma exceção é que ele é válido para comparar valores numéricos exatos e aproximar os valores numéricos, (a conversão de tipo requerida é definida pelas regras de promoção numérica do Java). Se houver uma tentativa de comparar tipos diferentes, o seletor será sempre false.
  - A comparação de sequência e booleano está restrita a = e <>. Duas sequências serão iguais apenas se contiverem a mesma sequência de caracteres.
- Operadores aritméticos em ordem de precedência:
  - +, - unário.
  - * multiplicação e / divisão.
  - + adição e - subtração.
  - Operações aritméticas em um valor NULL não são suportadas. Se forem tentadas, o seletor completo será sempre false.
  - Operações aritméticas devem usar promoção numérica Java.
- Operador de comparação arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 e arithmetic-expr3:
  - `Age BETWEEN 15 and 19` é equivalente a `age>= 15 AND age <= 19`.
  - `Age NOT BETWEEN 15 and 19` é equivalente a `age < 15 OR age > 19`.
  - Se qualquer uma das expressões de uma operação BETWEEN for NULL, o valor da operação será false. Se qualquer uma das expressões de uma operação NOT BETWEEN for NULL, o valor da operação será true.
- identificador `[NOT] IN (string-literal1, string-literal2,...)` operador de comparação em que o identificador tem um valor de Sequência ou NULL .
  - `Country IN ('UK', 'US', 'France')` é true para `'UK'` e false para `'Peru'`. Ele é equivalente à expressão `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` é false para `'UK'` e true para `'Peru'`. Ele é equivalente à expressão `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Se o identificador de uma operação IN ou NOT IN for NULL, o valor da operação será desconhecido.
- Operador de comparação `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]`, em que `identifier` tem um valor de sequência. *pattern-value* é um literal de sequência, em que _ representa qualquer caractere único e % representa qualquer sequência de caracteres (incluindo a sequência vazia). Todos os outros caracteres representam eles mesmos. O *escape-character* opcional é um literal de sequência de caracteres único que é usado para evitar o

significado especial de _ e % em *pattern-value*. O operador LIKE deve ser usado apenas para comparar dois valores de sequência.

  - `phone LIKE '12%3'` é true para 123 e 12993 e false para 1234.
  - `word LIKE 'l_se'` é true para lose e false para loose.
  - `underscored LIKE '\_%' ESCAPE '\'` é true para _foo e false para bar.
  - `phone NOT LIKE '12%3'` é false para 123 e 12993 e true para 1234.
  - Se o identificador de uma operação LIKE ou `NOT LIKE` for NULL, o valor da operação será desconhecido.

  Nota: O operador LIKE deve ser usado para comparar dois valores de sequência. O valor de `Root.MQMD.CorrelId` é uma matriz de bytes de 24 bytes, não uma sequência de caracteres. A sequência do seletor `Root.MQMD.CorrelId LIKE 'ABC%'` é aceita pelo analisador como sintaticamente válida, mas é avaliada como false. Quando você está comparando uma matriz de bytes com uma sequência de caracteres, LIKE, então, não pode ser usado.

- O operador de comparação `identifier IS NULL` testa para um valor de campo de cabeçalho NULL ou um valor de propriedade ausente.
- O operador de comparação `identifier IS NOT NULL` testa a existência de um valor de campo de cabeçalho ou um valor de propriedade não nulo.
- Valores nulos

  A avaliação de expressões do seletor que contêm valores NULL é definida pela semântica SQL 92 NULL, em resumo:

  - SQL trata um valor NULL como desconhecido.
  - Comparação ou aritmética com um valor desconhecido sempre resulta em um valor desconhecido.
  - Os operadores `IS NULL` e `IS NOT NULL` convertem um valor desconhecido nos valores TRUE e FALSE.

  Os operadores booleanos usam uma lógica de três valores (T=TRUE, F=FALSE, U=UNKNOWN)

Tabela 1. O valor do resultado do operador booleano quando a lógica é A AND B

Operador A	**Operador B**	**Resultado (A AND B)**
T	F	F
T	U	U
T	T	T
F	T	F
F	U	F
F	F	F
U	T	U
U	U	U
U	F	F

Tabela 2. O valor do resultado do operador booleano quando a lógica é A OR B

Operador A	**Operador B**	**Resultado (A OR B)**
T	F	T
T	U	T
T	T	T

Tabela 2. O valor do resultado do operador booleano quando a lógica é A OR B (continuação)		
Operador A	**Operador B**	**Resultado (A OR B)**
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

Tabela 3. O valor do resultado do operador booleano quando a lógica é NOT A	
Operador A	**Resultado (NOT A)**
T	F
F	T
U	U

O seletor de mensagem a seguir seleciona mensagens com um tipo de mensagem de carro, cor azul e peso maior que 2500 lbs:

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

Embora SQL suporte comparação e aritmética de decimal fixo, os seletores de mensagens não suportam. Por isso literais numéricos exatos são restritos àqueles sem um decimal. Por isso também há valores numéricos com um decimal como uma representação alternativa para um valor numérico aproximado.

Comentários SQL não são suportados.

Conceitos relacionados

“Propriedades da Mensagem” na página 27
Use propriedades de mensagem para permitir que um aplicativo selecione mensagens para processar ou recuperar informações sobre uma mensagem sem acessar cabeçalhos MQMD ou MQRFH2. Elas também facilitam a comunicação entre aplicativos IBM MQ e JMS.

Referências relacionadas

MsgHandle

MQBUFMH - Converter buffer em identificador de mensagens

Regras e restrições de sequência de seleção

Familiarize-se com essas regras sobre como as sequências de seleção são interpretadas e as restrições de caracteres para evitar problemas em potencial ao usar seletores.

- Seleção de mensagens para o sistema de mensagens de publicar/assinar ocorre na mensagem conforme enviada pelo publicador. Consulte sequências de Seleção.
- A equivalência é testada usando um único caractere de igual; por exemplo, a = b é correto, enquanto que a == b é incorreto.
- Um operador usado por muitas linguagens de programação para representar 'diferente de' é !=. Essa representação não é um sinônimo válido para <>; por exemplo, a <> b é válido, enquanto a != b não é válido.
- Aspas simples são reconhecidas somente se o caractere ' (U+0027) for usado. De forma semelhante, aspas duplas, válidas somente quanto usadas para englobar sequências de bytes, devem usar o caractere " (U+0022).

- Os símbolos &, &&, | e || não são sinônimos para conjunção / disjunção lógica; por exemplo, a `&& b` deve ser especificado como a `AND b`.
- Os caracteres curinga `*` e `?` não são sinônimos para `%` e `_`.
- Seletores contendo expressões compostas, como `20 < b < 30`, não são válidos. O analisador avalia os operadores que têm a mesma precedência da esquerda para a direita. O exemplo se tornaria, portanto, `(20 < b) < 30`, o que não faz sentido. Em vez disso, a expressão deve ser gravada como `(b > 20) AND (b < 30)`
- Sequências de bytes devem ser colocadas entre aspas duplas; se as aspas simples forem usadas, a sequência de bytes é considerada uma sequência literal. O número de caracteres (não o número que os caracteres representam) seguindo `0x` deve ser um múltiplo de dois.
- A palavra-chave `IS` não é um sinônimo do caractere de igual. Assim, as sequências de seleção a `IS 3` e `b IS 'red'` não são válidas. A palavra-chave `IS` existe somente para suportar casos de `IS NULL` e `IS NOT NULL`.

Conceitos relacionados

"Comportamento de seleção" na página 35
Visão geral do comportamento de seleção do IBM MQ.

Referências relacionadas

Sequências de seleção

Considerações sobre UTF-8 e Unicode ao usar seletores de mensagens

Caracteres, não entre aspas simples, que compõem o palavras-chave reservadas de uma sequência de seleção devem ser inserido em Basic Latin Unicode (variando de caractere U+0000 para U+0007F). Não é válido usar outras representações de ponto de código de caracteres alfanuméricos. Por exemplo, o número 1 deve ser expressado como U+0031 em Unicode, não é válido usar o equivalente de dígito de largura total U+FF11 nem o equivalente árabe U+0661.

Os nomes de propriedades de mensagem podem ser especificados usando qualquer sequência válida de caracteres Unicode. Nomes de propriedades de mensagem contidos dentro de sequências de seleção que são codificados em UTF-8 serão validados mesmo se eles contiverem caracteres de multibyte. A validação de UTF-8 multibyte é rígida e deve-se assegurar que as sequências de UTF-8 válidas sejam usadas para nomes de propriedades de mensagem. Caracteres além do Unicode Basic Multilingual Plane (aqueles acima de U+FFFF), representados em UTF-16 por pontos de código substitutos (X'D800' a X'DFFF') ou quatro bytes em UTF-8, não são suportados em nomes de propriedade de mensagem.

Nenhum processamento extra é executado em nomes de propriedades ou valores ao comparar igualdade. Isso significa, por exemplo, que não pré/decomposição ocorre e ligações não têm nenhum significado especial concedido. Por exemplo, o caractere til pré-composto U+00FC não é considerado como equivalente a U+0075 + U+0308 e a sequência de caracteres ff não é considerada equivalente ao Unicode U+FB00 (LATIN SMALL LIGATURE FF)

Dados de propriedades colocados entre aspas simples podem ser representados por qualquer sequência de bytes e não são validados.

Selecionando no conteúdo de uma mensagem

É possível assinar com base em uma seleção de conteúdo de carga útil de mensagem (também conhecido como filtragem de conteúdo), mas a decisão sobre quais mensagens devem ser entregues para essa assinatura não pode ser executada diretamente pelo IBM MQ; em vez disso, um provedor de seleção de mensagem estendida, por exemplo IBM Integration Bus, é necessário para processar as mensagens.

Quando um aplicativo publica em uma sequência de tópicos, na qual um ou mais assinantes têm uma sequência de seleção selecionando no conteúdo da mensagem, o IBM MQ irá solicitar que o provedor de seleção de mensagem estendida analise a publicação e informe o IBM MQ se a publicação corresponde aos critérios de seleção especificados por cada assinante com um filtro de conteúdo.

Se o provedor de seleção de mensagem estendida determinar que a publicação corresponde à sequência de seleção do assinante, a mensagem continuará sendo entregue ao assinante.

Se o provedor de seleção de mensagem estendida determinar que a publicação não corresponde, a mensagem não será entregue ao assinante. Isso pode fazer com que a chamada MQPUT ou MQPUT1 falhe com o código de razão MQRC_PUBLICATION_FAILURE. Se o provedor de seleção de mensagem estendida não conseguir analisar a publicação, o código de razão MQRC_CONTENT_ERROR será retornado e a chamada MQPUT ou MQPUT1 falhará.

Se o provedor de seleção de mensagem estendida estiver indisponível ou não conseguir determinar se o assinante deve receber a publicação, o código de razão MQRC_SELECTION_NOT_AVAILABLE será retornado e a chamada MQPUT ou MQPUT1 falhará.

Quando uma assinatura estiver sendo criada com um filtro de conteúdo e o provedor de seleção de mensagem estendida não estiver disponível, a chamada MQSUB falhará com o código de razão MQRC_SELECTION_NOT_AVAILABLE. Se uma assinatura com um filtro de conteúdo estiver sendo retomada e o provedor de seleção de mensagem estendida não estiver disponível, a chamada MQSUB retornará um aviso de MQRC_SELECTION_NOT_AVAILABLE, mas a assinatura terá permissão para ser continuada.

Referências relacionadas

Sequências de seleção

Consumo assíncrono de mensagens do IBM MQ

O consumo assíncrono usa um conjunto de extensões da Message Queue Interface (MQI), as chamadas MQI MQCB e MQCTL, que permitem que um aplicativo MQI seja escrito para consumir mensagens de um conjunto de filas. As mensagens são entregues ao aplicativo chamando uma 'unidade de código' identificada pelo aplicativo passando a mensagem ou um token que representa a mensagem.

No mais direto dos ambientes de aplicativos, a unidade de código é definida por um ponteiro de função, no entanto, em outros ambientes, a unidade de código pode ser definida por um nome de programa ou módulo.

No consumo assíncrono de mensagens, os termos a seguir são usados:

Consumidor de mensagens
: Uma construção de programação que permite definir um programa, ou função, a ser chamado com uma mensagem quando uma que corresponda ao requisito dos aplicativos se torna disponível.

Manipulador de eventos
: Uma construção de programação que permite definir um programa ou função para chamar quando um evento assíncrono, como quiesce do gerenciador de filas, ocorre.

Retorno de chamada
: Um termo genérico usado para fazer referência a uma rotina do Consumidor de mensagens ou do Manipulador de eventos.

Consumo assíncrono pode simplificar o design e a implementação de novos aplicativos, principalmente aquelas que processam diversas filas de entrada ou assinaturas. No entanto, se você estiver usando mais de uma fila de entrada e estiver processando as mensagens na sequência de prioridade, a sequência de prioridade será observada independentemente em cada fila. Pode ser que você obtenha mensagens de prioridade baixa de uma fila antes de mensagens de prioridade alta de outra. A ordem das mensagens entre várias filas não é garantida. Observe também que se você usar saídas de API, poderá precisar mudá-las para incluir as chamadas de MQCB e MQCTL.

As ilustrações a seguir fornecem um exemplo de como é possível usar essa função.

Figura 5 na página 44 mostra um aplicativo multiencadeado consumindo mensagens a partir de duas filas. O exemplo mostra todas as mensagens que estão sendo entregues a uma única função.

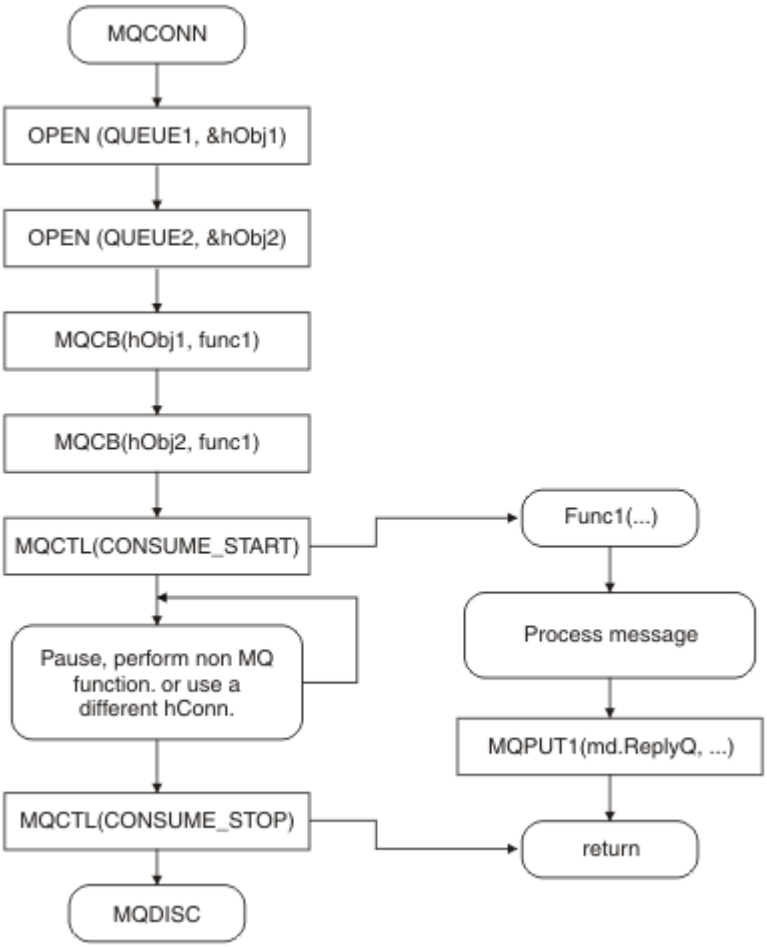


Figura 5. Aplicativo padrão acionado por mensagens consumindo de duas filas

z/OS No z/OS, o encadeamento de controle principal deve emitir uma chamada MQDISC antes de ser finalizado. Isso permite que quaisquer encadeamentos de retorno de chamada finalizem e liberem recursos do sistema.

Figura 6 na página 45Esse fluxo de amostra mostra um aplicativo de encadeamento único consumindo mensagens de duas filas. O exemplo mostra todas as mensagens que estão sendo entregues a uma única função.

A diferença do caso assíncrono é que o controle não retornará para o emissor de MQCTL até que todos os consumidores tiverem se desativado; ou seja, um consumidor emitiu uma solicitação MQCTL STOP ou o gerenciador de filas efetua quiesce.

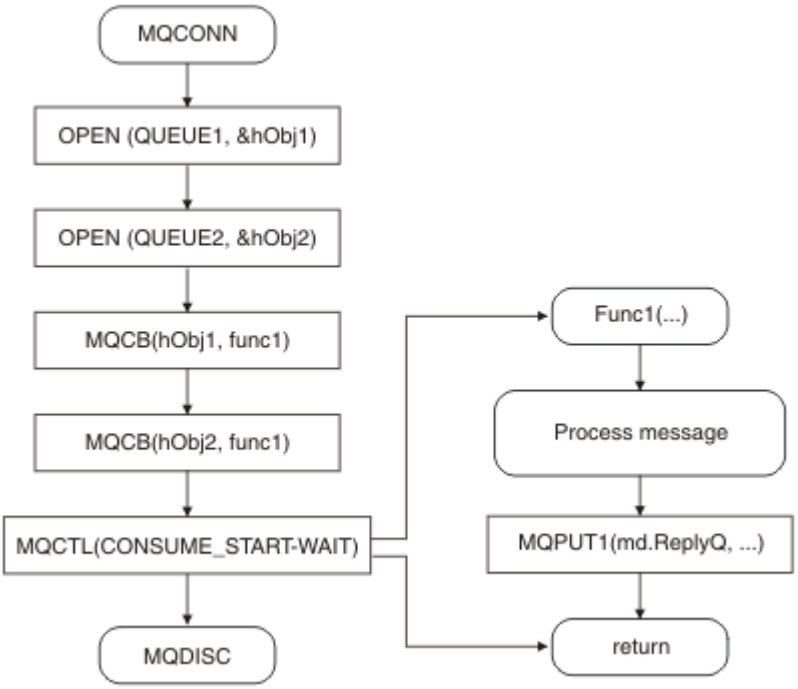


Figura 6. Aplicativo acionado por mensagens de encadeamento único consumindo de duas filas

Grupos de mensagens

As mensagens podem ocorrer dentro de grupos para permitir a ordenação de mensagens.

Os grupos de mensagens permitem que diversas mensagens sejam marcadas como relacionadas entre si e que uma ordem lógica seja aplicada ao grupo (consulte "Ordenação lógica e física" na página 786). Em Multiplataformas, a segmentação de mensagens permite que mensagens grandes sejam divididas em segmentos menores. Não é possível usar mensagens agrupadas ou segmentadas ao colocar em um tópico.

A hierarquia dentro de um grupo é a seguinte:

Group

Este é o nível mais alto na hierarquia e é identificado por um *GroupId*. Ele consiste em uma ou mais mensagens que contêm o mesmo *GroupId*. Essas mensagens podem ser armazenadas em qualquer lugar na fila.

Nota: O termo *mensagem* é usado aqui para denotar um item em uma fila, como seria retornado por uma única MQGET que não especifique MQGMO_COMPLETE_MSG.

Figura 7 na página 45 mostra um grupo de mensagens lógicas:

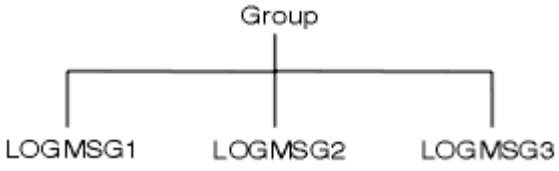


Figura 7. Grupo de mensagens lógicas

Ao abrir uma fila e especificar MQOO_BIND_ON_GROUP, você força todas as mensagens em um grupo que são enviadas para esta fila a serem enviadas à mesma instância da fila. Para obter mais informações sobre a opção BIND_ON_GROUP, consulte Manipulando afinidades de mensagens.

Mensagem lógica

As mensagens lógicas dentro de um grupo são identificadas pelos campos *GroupId* e *MsgSeqNumber*. O *MsgSeqNumber* começa em 1 para a primeira mensagem em um grupo e se uma mensagem não estiver em um grupo, o valor do campo é 1.

Use as mensagens lógicas dentro de um grupo para:

- Certificar-se da ordenação (se isto não for garantido sob as circunstâncias nas quais a mensagem é transmitida).
- Permitir que os aplicativos agrupem mensagens semelhantes (por exemplo, todas aquelas que devem ser processadas pela mesma instância do servidor).

Cada mensagem dentro de um grupo consiste em uma mensagem física, a menos que seja dividida em segmentos. Cada mensagem é logicamente uma mensagem separada e somente os campos *GroupId* e *MsgSeqNumber* no MQMD precisam suportar qualquer relacionamento com outras mensagens no grupo. Outros campos no MQMD são independentes; alguns podem ser idênticos para todas as mensagens no grupo, enquanto outros podem ser diferentes. Por exemplo, as mensagens em um grupo podem ter nomes de formato diferente, CCSIDs e codificações.

Segmento

Os segmentos são usados para lidar com mensagens muito grandes para o aplicativo put ou get ou para o gerenciador de filas (incluindo gerenciadores de filas intervenientes pelos quais a mensagem passa). Para obter mais informações, consulte "Segmentação de mensagem" na página 804.

Uma mensagem individual é dividida em mensagens menores chamadas *segmentos*. Um segmento de uma mensagem é identificado pelos campos *GroupId*, *MsgSeqNumber*e *Offset* .. O campo *Offset* inicia em zero para o primeiro segmento em uma mensagem.

Cada segmento consiste em uma mensagem física que pode pertencer a um grupo (Figura 8 na página 46 mostra um exemplo de mensagens dentro de um grupo). Um segmento é logicamente parte de uma única mensagem, portanto, somente os campos *MsgId*, *Offset* e *MsgFlags* no MQMD devem ser diferentes entre segmentos separados da mesma mensagem. Se um segmento não chegar, o código de razão MQRC_INCOMPLETE_GROUP ou MQRC_INCOMPLETE_MSG é retornado conforme apropriado.

Figura 8 na página 46 mostra um grupo de mensagens lógicas, algumas das quais são segmentadas:

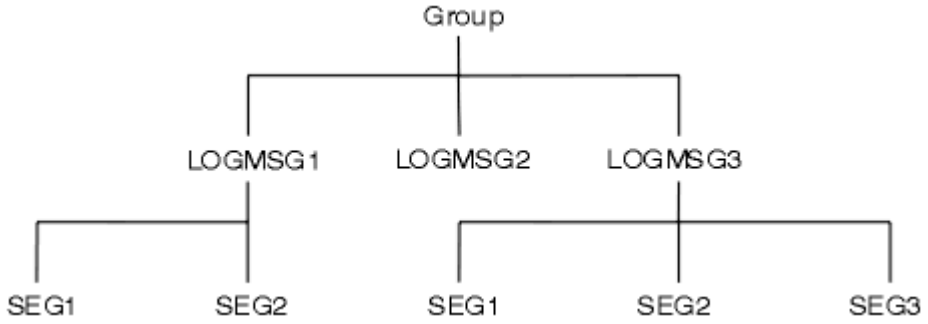


Figura 8. Mensagens segmentadas

z/OS A segmentação não é suportada no IBM MQ for z/OS.

Você não pode usar segmentos ou mensagens agrupadas com Publicação/Assinatura.

Conceitos relacionados

"Segmentação de mensagem" na página 804
Use estas informações para aprender sobre a segmentação de mensagens. Esse recurso não é suportado em IBM MQ for z/OS ou por aplicativos usando IBM MQ classes for JMS.

Referências relacionadas

"Ordenação lógica e física" na página 786
Dentro de cada nível de prioridade, as mensagens nas filas podem ocorrer na ordem *física* ou *lógica*

Persistência de mensagem

As mensagens persistentes são gravadas em logs e arquivos de dados de fila. Se um gerenciador de filas for reiniciado após uma falha, ele recuperará essas mensagens persistentes conforme necessário a partir dos dados registrados. As mensagens que não são persistentes serão descartadas se um gerenciador de filas parar, independentemente de a parada ser resultante de um comando do operador ou devido à falha de alguma parte do seu sistema.

z/OS As mensagens não persistentes armazenadas em um recurso de acoplamento (CF) no z/OS são uma exceção a isso. Eles persistem desde que o CF permaneça disponível.

Ao criar uma mensagem, se você inicializar o descritor de mensagens (MQMD) usando os padrões, a persistência de mensagem é obtida do atributo **DefPersistence** da fila especificada no comando MQOPEN. Como alternativa, é possível configurar a persistência da mensagem usando o campo *Persistence* da estrutura MQMD para definir a mensagem como persistente ou não persistente..

O desempenho do seu aplicativo é afetado ao usar as mensagens persistentes. A extensão do efeito depende das características de desempenho do subsistema de E/S do computador e de como usar as opções do ponto de sincronização em cada plataforma:

- Uma mensagem persistente, fora da unidade de trabalho atual, é gravada no disco em cada operação put e get. Consulte o “Confirmando e fazendo backup de unidades de trabalho” na página 865.
- z/OS ALW Para todas as plataformas, exceto IBM i, uma mensagem persistente dentro da unidade de trabalho atual é registrada somente quando a unidade de trabalho é confirmada e a unidade de trabalho pode conter muitas operações de fila.

As mensagens não persistentes podem ser usadas para o sistema de mensagens rápidas. Consulte Segurança de mensagens para obter informações adicionais sobre as mensagens rápidas.

Nota: Uma combinação de composição de mensagens persistentes em uma unidade de trabalho e composição de mensagens persistentes fora de uma unidade ou de trabalho, pode, potencialmente, causar problemas severos de desempenho em seus aplicativos. Isso se aplica, em especial, quando a mesma fila de destino é usada para ambas as operações.

Mensagens que falham na entrega

Quando um gerenciador de filas não consegue colocar uma mensagem em uma fila, você tem várias opções.

Você pode:

- Tentar colocar a mensagem na fila novamente.
- Solicitar que a mensagem seja retornada ao emissor.
- Colocar a mensagem na fila de mensagens não entregues.

Consulte “Manipulando erros de programa processual” na página 1050para obter mais informações.

Mensagens que são restauradas

Ao processar mensagens de uma fila sob o controle de uma unidade de trabalho, a unidade de trabalho pode consistir em uma ou mais mensagens. Se uma restauração ocorrer, as mensagens que tiverem sido recuperadas da fila serão recolocadas na fila e elas poderão ser processadas novamente em outra unidade de trabalho. Se o processamento de uma mensagem específica estiver causando o problema, a unidade de trabalho será restaurada novamente. Isso pode causar um loop de processamento. As mensagens que foram colocadas em uma fila são removidas da fila.

Um aplicativo pode detectar mensagens que são capturadas em um loop assim testando o campo *BackoutCount* do MQMD. O aplicativo pode corrigir a situação ou emitir um aviso para um operador.

Multi A contagem de restauração sempre permanece às reinicializações do gerenciador de filas. Qualquer mudança ao atributo **HardenGetBackout** será ignorada.

z/OS Para filas compartilhadas, a contagem de restauração sempre permanece às reinicializações do gerenciador de filas. Para todas as outras configurações no z/OS, para assegurar que a contagem de recuperação de filas privadas permaneça às reinicializações do gerenciador de filas, configure o atributo *HardenGetBackout* para MQQA_BACKOUT_HARDENED; caso contrário, se o gerenciador de filas tiver de ser reiniciado, ele não manterá uma contagem de restauração precisa para cada mensagem. A configuração do atributo desse modo inclui o custo de processamento extra.

Para obter mais informações sobre a consolidação e restauração de mensagens, consulte "Confirmando e fazendo backup de unidades de trabalho" na página 865.

Fila de resposta e gerenciador de filas

Existem ocasiões em que você pode receber mensagens em resposta a uma mensagem que você enviar:

- Uma mensagem de resposta em resposta a uma mensagem de pedido
- Uma mensagem de relatório sobre um evento inesperado ou de expiração
- Uma mensagem de relatório sobre um COA (Confirmação da Chegada) ou evento de um COD (Confirmação da Entrega)
- Uma mensagem de relatório sobre um PAN (Positive Action Notification) ou evento de um NAN (Negative Action Notification)

Usando a estrutura MQMD, especifique o nome da fila para a qual você deseja enviar as mensagens de resposta e de relatório no campo *ReplyToQ*. Especifique o nome do gerenciador de filas que possui a fila de resposta no campo *ReplyToQMgr*.

Se você deixar o campo *ReplyToQMgr* em branco, o gerenciador de filas configurará o conteúdo dos seguintes campos no descritor de mensagens na fila:

ReplyToQ
Se *ReplyToQ* for uma definição local de uma fila remota, o campo *ReplyToQ* será configurado para o nome da fila remota; caso contrário, esse campo não será mudado.

ReplyToQMgr
Se *ReplyToQ* for uma definição local de uma fila remota, o campo *ReplyToQMgr* será configurado para o nome do gerenciador de filas que possui a fila remota; caso contrário, o campo *ReplyToQMgr* será configurado para o nome do gerenciador de filas ao qual seu aplicativo está conectado.

Nota: É possível solicitar que um gerenciador de filas faça mais de uma tentativa de entregar uma mensagem e que a mensagem seja descartada se falhar. Se a mensagem, após falhar em ser entregue, não dever ser descartada, o gerenciador de filas remotas a colocará na fila de mensagens não entregues (veja "Usando a fila de mensagens não entregues" na página 1054).

Contexto da mensagem

As informações de *Contexto da Mensagem* permitem que o aplicativo recupere a mensagem para descobrir sobre o originador da mensagem.

O aplicativo de recuperação pode desejar:

- Verificar se o aplicativo de envio tem o nível correto de autoridade
- Executar algumas funções de contabilidade para que ele possa carregar o aplicativo de envio para qualquer trabalho que ele precise executar
- Manter uma trilha de auditoria de todas as mensagens com as quais ele trabalhou

Quando você usa a chamada MQPUT ou MQPUT1 para colocar uma mensagem em uma fila, é possível especificar se o gerenciador de filas deve incluir algumas informações de contexto padrão para o descritor de mensagens. Aplicativos que possuem o nível apropriado de autoridade podem incluir

informações de contexto adicionais. Para obter mais informações sobre como especificar informações de contexto, consulte “Controlando informações de contexto da mensagem” na página 771.

O contexto do usuário é usado pelo gerenciador de filas ao gerar os seguintes tipos de mensagem de relatório:

- Confirmar na entrega
- Expiração

Quando essas mensagens de relatório são geradas, o contexto do usuário é verificado para autoridade +put e +passid no destino do relatório. Quando o contexto do usuário possui autoridade insuficiente, a mensagem de relatório é colocada na fila de mensagens não entregues se uma fila tiver sido definida. Onde não houver uma fila de mensagens não entregues, a mensagem de relatório é descartada.

Todas as informações de contexto são armazenadas nos campos de contexto do descritor de mensagens. O tipo de informação é classificado em identidade, origem e informações de contexto do usuário.

Contexto de Identidade

A informação de *Contexto de identidade* identifica o usuário do aplicativo que coloca primeiro a mensagem em uma fila. Os aplicativos devidamente autorizados podem configurar os seguintes campos:

- O gerenciador de filas preenche o campo *UserIdentifier* com um nome que identifica o usuário. O modo que o gerenciador de filas pode fazer isso depende do ambiente no qual o aplicativo está sendo executado.
- O gerenciador de filas preenche o campo *AccountingToken* com um token ou número que é determinado a partir do aplicativo que coloca a mensagem.
- Os aplicativos podem usar o campo *ApplIdentityData* para qualquer informação adicional que desejam incluir sobre o usuário (por exemplo, uma senha criptografada).

Um Windows systems security identifier (SID) é armazenado no campo *AccountingToken* quando uma mensagem é criada em IBM MQ for Windows. O SID pode ser usado para suplementar o campo *UserIdentifier* e para estabelecer as credenciais de um usuário.

Para obter informações sobre como o gerenciador de filas preenche os campos *UserIdentifier* e *AccountingToken*, consulte as descrições desses campos em UserIdentifier e AccountingToken.

Os aplicativos que passam mensagens de um gerenciador de filas para outro devem também passar informações de contexto de identidade para que outros aplicativos saibam a identidade do originador da mensagem.

Contexto de origem

As informações de *Contexto de origem* descrevem o aplicativo que coloca a mensagem na fila na qual a mensagem está atualmente armazenada. O descritor de mensagens contém os seguintes campos para informações de contexto de origem:

- *PutApplType* define o tipo de aplicativo que coloca a mensagem (por exemplo, uma transação do CICS).
- *PutApplName* define o nome do aplicativo que coloca a mensagem (por exemplo, o nome de uma tarefa ou transação).
- *PutDate* define a data na qual a mensagem foi colocada na fila.
- *PutTime* define o horário no qual a mensagem foi colocada na fila.
- *ApplOriginData* define quaisquer informações extras que um aplicativo deseja incluir sobre a origem da mensagem. Por exemplo, ele pode ser configurado por aplicativos devidamente autorizados para indicar se os dados de identidade são confiáveis.

As informações de contexto de origem são geralmente fornecidas pelo gerenciador de filas. GMT (Horário de Greenwich) é usado para os campos *PutDate* e *PutTime*. Consulte as descrições desses campos em PutDate e PutTime.

Um aplicativo com autoridade suficiente pode fornecer seu próprio contexto. Isso permite que as informações de contabilidade sejam preservadas quando um único usuário tiver uma ID de usuário diferente em cada um dos sistemas que processam uma mensagem que eles originaram.

Objetos do IBM MQ

Estas informações fornecem detalhes sobre objetos do IBM MQ que incluem: gerenciadores de filas, grupos de filas compartilhadas, filas, objetos de tópico administrativo, listas de nomes, definições de processo, objetos de informações sobre autenticação, canais, classes de armazenamento, listeners e serviços.

Os gerenciadores de filas definem as propriedades (conhecidas como atributos) desses objetos. Os valores desses atributos afetam o modo no qual o IBM MQ processa esses objetos. Em seus aplicativos, use o Message Queue Interface (MQI) para controlar esses objetos. Os objetos são identificados por um *descritor de objeto* (MQOD) quando direcionados de um programa.

Ao usar comandos do IBM MQ para definir, alterar ou excluir objetos, por exemplo, o gerenciador de filas verificará se você tem o nível necessário de autoridade para executar essas operações. Da mesma forma, quando um aplicativo usa a chamada MQOPEN para abrir um objeto, o gerenciador de filas verifica se o aplicativo possui o nível necessário de autoridade antes que conceda acesso a esse objeto. As verificações são feitas no nome do objeto sendo aberto.

Conceitos relacionados

"Controlando informações de contexto da mensagem" na página 771
Quando você usa a chamada MQPUT ou MQPUT1 para colocar uma mensagem em uma fila, é possível especificar se o gerenciador de filas deve incluir algumas informações de contexto padrão para o descritor de mensagens. Aplicativos que possuem o nível apropriado de autoridade podem incluir informações de contexto adicionais. É possível usar o campo de opções na estrutura MQPMO para controlar informações de contexto.

Referências relacionadas

"Opções de MQOPEN relacionadas ao contexto da mensagem" na página 762
Se deseja conseguir associar as informações a uma mensagem ao colocá-la em uma fila, deve-se usar uma das opções de contexto da mensagem ao abrir a fila.

Windows Preparando e executando aplicativos Microsoft Transaction Server

Para preparar um aplicativo MTS para que seja executado como um aplicativo IBM MQ MQI client, siga estas instruções conforme apropriado para o seu ambiente.

Para obter informações gerais sobre como desenvolver aplicações do Microsoft Transaction Server (MTS) que acessam os recursos do IBM MQ, consulte a seção sobre MTS no IBM MQ Help Center.

Para preparar um aplicativo MTS para que seja executado como um aplicativo IBM MQ MQI client, faça o seguinte para cada componente do aplicativo:

- Se o componente usar as ligações de linguagem C para o MQI, siga as instruções no "Preparando programas C no Windows" na página 1029, mas vincule o componente à biblioteca mqicxa.lib em vez da mqic.lib.
- Se o componente usar as classes IBM MQ C++, siga as instruções em "Construindo programas C++ no Windows" na página 561, mas vincule o componente à biblioteca imqx23vn.lib em vez da imqc23vn.lib.
- Se o componente usar as ligações de linguagem Visual Basic para o MQI, siga as instruções no "Preparando programas Visual Basic no Windows" na página 1032, mas quando definir o projeto Visual Basic, digite `MqType=3` no campo **Argumentos de compilação condicional**.

Considerações de design para aplicativos IBM MQ

Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

Ao projetar um aplicativo IBM MQ considere as questões e opções a seguir:

Tipo de aplicativo

Qual é o propósito do seu aplicativo? Consulte os links a seguir para obter informações sobre os diferentes tipos de aplicativos que é possível desenvolver:

- Servidor
- Client
- Publicação/assinatura
- Serviços da Web
- Saídas de usuário, saídas de API e serviços instaláveis

Além disso, também é possível escrever seus próprios aplicativos para automatizar a administração do IBM MQ. Para obter mais informações, veja O IBM MQ Administration Interface (MQAI) e Automatizando as tarefas de administração.

Linguagem de programação

O IBM MQ suporta uma série de linguagens de programação diferentes para gravação de aplicativos. Para obter informações adicionais, consulte “Desenvolvendo aplicativos para o IBM MQ” na página 5.

Aplicativos para mais de uma plataforma

Seu aplicativo será executado em mais de uma plataforma? Você tem uma estratégia para mudar para uma plataforma diferente daquela que usa hoje? Se a resposta para uma dessas perguntas for sim, assegure que você codifique seus programas para independência de plataforma.

Por exemplo, se você estiver usando C, codifique no C no padrão ANSI. Use uma função padrão de biblioteca C em vez de uma função específica de plataforma equivalente, mesmo se a função específica da plataforma for mais rápida ou mais eficiente. A exceção é quando eficiência no código é o ponto mais importante, nesse caso, é necessário codificar para ambas as situações usando #ifdef. Por exemplo:

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

Tipos de filas

Deseja criar uma fila toda vez que precisar de uma ou deseja usar as filas que já foram configuradas? Deseja excluir uma fila quando tiver terminado de usá-la ou ela será usada novamente? Deseja usar as filas de alias para independência do aplicativo? Para ver quais tipos de filas são suportadas, consulte o Filas.

z/OS **Usando filas compartilhadas, grupos de filas compartilhadas e clusters de grupos de compartilhamento de filas (apenas IBM MQ for z/OS)**

Talvez você queira beneficiar-se da disponibilidade aumentada, da escalabilidade e do balanceamento de carga de trabalho que são possíveis quando você usa filas compartilhadas com grupos de filas compartilhadas. Consulte Filas compartilhadas e grupos de compartilhamento de filas para obter mais informações.

Você também pode desejar estimar os fluxos de mensagens médio e de pico e considerar o uso de clusters de grupos de filas compartilhadas para difundir a carga de trabalho. Consulte Filas compartilhadas e grupos de compartilhamento de filas para obter mais informações.

Usando Clusters do Gerenciador de Filas

Talvez você deseje tirar proveito da administração do sistema simplificada e maior disponibilidade, escalabilidade e balanceamento de carga de trabalho que são possíveis ao usar clusters.

Tipos de mensagens

Talvez você deseje usar datagramas para mensagens simples, mas mensagens de solicitação (para as quais se espera resposta) para outras situações. Pode ser que deseje designar prioridades diferentes a algumas de suas mensagens. Para obter mais informações sobre como projetar mensagens, consulte “Técnicas de design para mensagens” na página 60.

Usando sistema de mensagens publicar/assinar ou ponto a ponto

Usando sistema de mensagens publicar/assinar, um aplicativo de envio envia as informações que deseja compartilhar em uma mensagem do IBM MQ para um destino padrão gerenciado por publicar/assinar do IBM MQ e permite que o IBM MQ manipule distribuição dessas informações. O aplicativo de destino não precisa saber nada sobre a origem das informações que recebe, ele apenas registra um interesse em um ou mais tópicos e recebe as informações quando estiverem disponíveis. Para obter mais informações sobre o envio de mensagens de publicar/assinar, consulte Envio de mensagens de publicar/assinar.

Usando o sistema de mensagens ponto a ponto, um aplicativo de envio envia uma mensagem a uma fila específica, de onde sabe que um aplicativo de recebimento irá recuperá-la. Um aplicativo de recebimento recebe mensagens de uma fila específica e age em seu conteúdo. Um aplicativo frequentemente funcionará como um emissor e um receptor, enviando uma consulta a outro aplicativo e recebendo uma resposta.

Controlando seus programas IBM MQ

Você pode desejar iniciar alguns programas automaticamente ou fazer programas esperarem até que uma mensagem específica chegue em uma fila (usando o recurso do IBM MQ de *acionamento*, consulte “Iniciando aplicativos IBM MQ usando acionadores” na página 877). Como alternativa, pode desejar iniciar outra instância de um aplicativo quando as mensagens em uma fila não estão sendo processadas rápido o suficiente (usando o recurso do IBM MQ *eventos de instrumentação*, conforme descrito em Eventos de instrumentação).

Executando seu aplicativo em um cliente IBM MQ

A MQI completa é suportada no ambiente do cliente e praticamente qualquer aplicativo IBM MQ escrito em uma linguagem processual pode ser vinculado novamente para execução em um IBM MQ MQI client. Vincule o aplicativo no IBM MQ MQI client à biblioteca MQIC, em vez de à biblioteca MQI.

z/OS Get(signal) no z/OS não é suportado.

Nota: Um aplicativo em execução em um cliente IBM MQ pode se conectar a mais de um gerenciador de filas simultaneamente ou usar um nome de gerenciador de filas com um asterisco (*) em uma chamada MQCONN ou MQCONNX. Mude o aplicativo se desejar vincular a bibliotecas do gerenciador de filas em vez a bibliotecas do cliente, porque essa função não estará disponível.

Consulte “Executando aplicativos no ambiente do IBM MQ MQI client” na página 932 para obter mais informações.

Desempenho do aplicativo

As decisões de design podem afetar o desempenho do aplicativo, para obter sugestões para aprimorar o desempenho de aplicativos IBM MQ, consulte “Considerações de design e desempenho do aplicativo” na página 61 IBM i e “Considerações de design e desempenho de aplicativos IBM i” na página 65.

Técnicas avançadas do IBM MQ

Para aplicativos mais avançados, você pode querer usar algumas técnicas avançadas do IBM MQ, como correlacionar respostas e gerar e enviar informações de contexto do IBM MQ. Para obter mais informações, consulte “Técnicas de design para aplicativos avançados” na página 63.

Protegendo dados e mantendo sua integridade

É possível usar as informações de contexto que são passadas com uma mensagem para testar se a mensagem foi enviada a partir de uma origem aceitável. É possível usar recursos de definição de ponto de sincronização fornecidos pelo IBM MQ ou seu sistema operacional para assegurar que seus dados permaneçam consistentes com outros recursos (consulte “Confirmando e fazendo backup de unidades de trabalho” na página 865 para obter detalhes adicionais). É possível usar o recurso de *persistência* de mensagens do IBM MQ para assegurar a entrega de mensagens importantes.

Testando aplicativos IBM MQ

O ambiente de desenvolvimento de aplicativos para programas IBM MQ não é diferente daquele de qualquer outro aplicativo, portanto, é possível usar as mesmas ferramentas de desenvolvimento, assim como os recursos de rastreio do IBM MQ.

z/OS Ao testar aplicativos CICS com o IBM MQ for z/OS, é possível usar o CICS Execution Diagnostic Facility (CEDF). O CEDF efetua trap da entrada e da saída de cada chamada MQI, assim como chamadas para todos os serviços do CICS. Além disso, no ambiente do CICS, é possível escrever um programa de saída cruzada da API para fornecer informações de diagnóstico antes e depois de cada chamada MQI. Para obter informações sobre como fazer isso, consulte "Using and writing applications on IBM MQ for z/OS" na página 901.

IBM i Ao testar aplicativos IBM i, é possível usar o Depurador padrão. Para iniciá-lo, use o comando STRDBG.

Manipulando exceções e erros

É necessário considerar como processar as mensagens que não podem ser entregues e como resolver situações de erro que são relatadas para você pelo gerenciador de filas. Para alguns relatórios, deve-se configurar opções de relatório em MQPUT.

Conceitos relacionados

Visão Geral Técnica do IBM MQ

"Design and performance considerations for z/OS applications" na página 67
Application design is one of the most important factors affecting performance. Use this topic to understand some of the design factors involved in performance.

"Desenvolvendo aplicativos para o IBM MQ" na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

"Conceitos de desenvolvimento de aplicativos" na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

"Escrevendo um aplicativo processual para enfileiramento" na página 732
Use essas informações para aprender como escrever aplicativos de enfileiramento, conectar e desconectar de um gerenciador de filas, publicar/assinar e abrir e fechar objetos.

"Escrevendo aplicativos clientes processuais" na página 924
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

"Desenvolvendo aplicativos C++" na página 537
O IBM MQ fornece classes C++ equivalentes a objetos do IBM MQ e algumas classes adicionais equivalentes aos tipos de dados de matriz. Ele fornece inúmeros recursos não disponíveis por meio do MQI.

"Usando IBM MQ classes for JMS/Jakarta Messaging" na página 83
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

"Usando o IBM MQ classes for Java" na página 353
Use o IBM MQ em um ambiente Java. O IBM MQ classes for Java permite que um aplicativo Java se conecte ao IBM MQ como um cliente IBM MQ ou se conecte diretamente a um gerenciador de filas do IBM MQ.

Tarefas relacionadas

"Desenvolvendo aplicativos do .NET" na página 565
IBM MQ classes for .NET permitir que .NET aplicativos se conectem ao IBM MQ como um IBM MQ MQI client ou se conectem diretamente a um servidor IBM MQ .

Especificando o nome do aplicativo em linguagens de programação suportadas

Antes do IBM MQ 9.2.0, você já poderia especificar um nome do aplicativo nos aplicativos clientes do Java ou do JMS. A partir da IBM MQ 9.2.0, esse recurso é estendido para outras linguagens de programação no IBM MQ for Multiplatforms.

Como o nome do aplicativo é usado

O nome do aplicativo é saída de:

- comando runmqsc DISPLAY CONN APPLTAG
- APPLTAG do comando runmqsc DISPLAY QSTATUS (HANDLE)
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- Rastreio de atividade do aplicativo

O nome do aplicativo também é usado ao configurar o rastreio de atividade do aplicativo. O nome do aplicativo padrão para aplicativos nãoJava é o nome truncado do executável, exceto Windows e IBM i.

No Windows, o nome padrão é o nome completo do executável, truncado para 28 caracteres à esquerda.

No IBM i, o nome padrão é o nome da tarefa.

Para aplicativos do Java ele é o nome da classe prefixado pelo nome do pacote truncado à esquerda para 28 caracteres.

Para obter mais informações, veja PutApplName.

Aplicativos no IBM MQ for Multiplatforms podem configurar seus nomes de aplicativos administrativamente ou usando vários métodos de programação. Isso permite que os aplicativos forneçam um nome independente de plataforma mais significativo, quando você configurar o rastreio de atividade do aplicativo ou quando efetuar a saída de vários comandos **runmqsc**.

É possível reequilibrar aplicativos em um cluster uniforme. Os nomes do aplicativo significativos são usados para alcançar isso.

Caracteres Suportados

Consulte “Caracteres de nome do aplicativo recomendados” na página 55 para obter mais informações sobre como especificar o nome do aplicativo.

Linguagens de Programação

Consulte “Conexões de linguagem de programação” na página 57 para obter mais informações sobre como os aplicativos resolvem para as bibliotecas do IBM MQ em C e outras linguagens de programação, podem fornecer o nome do aplicativo.

Aplicativos do .NET gerenciados

Consulte “Aplicativos do .NET gerenciados” na página 58 para obter informações sobre como os aplicativos gerenciados do .NET podem fornecer o nome do aplicativo.

Aplicativos XMS

Consulte “XMS aplicativos” na página 59 para obter informações sobre como os aplicativos do XMS podem fornecer o nome do aplicativo.

Aplicativos de Ligações doJava e do JMS

ALW

Consulte “Aplicativos de Ligações doJava e do JMS” na página 59 para obter informações sobre como os aplicativos do Java e do JMS podem fornecer o nome do aplicativo.

Conceitos relacionados

Rastreio de atividade do aplicativo

Sobre clusters uniformes

Referências relacionadas

MQCNO

MQCNO no IBM i

Usando o nome do aplicativo em linguagens de programação suportadas

Use estas informações para saber como o nome do aplicativo é selecionado nas várias linguagens que o IBM MQ suporta.

Caracteres de nome do aplicativo recomendados

Os nomes de aplicativos devem estar no conjunto de caracteres fornecido pelo atributo **CodedCharSetId** do campo do gerenciador de filas Para obter mais informações sobre este atributo, consulte Atributos do gerenciador de filas.

No entanto, se o aplicativo estiver em execução como um IBM MQ MQI client, o nome do aplicativo deverá estar no conjunto de caracteres e na codificação do cliente.

Para assegurar uma transição suave do nome do aplicativo entre os gerenciadores de filas e para permitir o monitoramento de recursos do aplicativo por meio dos tópicos de monitoramento de recursos, os nomes do aplicativo devem conter apenas caracteres para impressão de byte único

Notas:

- Também é necessário evitar o uso de caracteres de barra e de e comercial (símbolo &) em nomes de aplicativos.
- Você deve evitar o uso do caractere e comercial (símbolo &) em nomes de aplicativos. As métricas STATAPP de tópico do sistema para nomes de aplicativos contendo um e comercial (símbolo &) não serão produzidas.

Isso limita o nome a:

- Caracteres alfanuméricos: `A-Z, a-ze 0-9`

  Nota: Os caracteres minúsculos `a-z` não devem ser usados em nomes de aplicativos em sistemas utilizando EBCDIC Katakana.
- O caractere de espaço
- Caracteres para impressão que são invariáveis em EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`
- O caractere /. Ao inscrever-se em métricas de rastreio de atividade ou de tópico do sistema STATAPP para um aplicativo cujo nome contém uma barra, você deve substituir qualquer caractere de barra por um caractere e comercial (símbolo &). Por exemplo, para receber métricas de STATAPP para um aplicativo chamado "DEPT1/APPS/STOCKQUOTE", você deve se inscrever na sequência de tópicos "$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE". Os aplicativos de amostra amqsact e amqsrua converterão automaticamente caracteres de barra para e comercial (símbolo &) ao criar suas assinaturas.

Como você configura os caracteres

A tabela a seguir resume os meios pelos quais o nome do aplicativo é escolhido em várias linguagens que o IBM MQ suporta. Os meios pelos quais o nome é escolhido estão em ordem de precedência, iniciando com o mais alto.

	Ligações e cliente C	**Ligações e cliente do Java**	**Ligações e cliente do JMS**	**Cliente .NET gerenciado**	**Ligações e cliente .NET não gerenciados**	**Cliente XMS gerenciado**	**Ligações e cliente .XMS não gerenciados**
Substituição de propriedade da conexão		Substituição de propriedade da conexão Java		Substituição de propriedade da conexão .NET	Substituição de propriedade da conexão .NET		
Propriedade substituída		Propriedade substituídaJava		Propriedade substituída .NET	Propriedade substituída .NET		
MQEnvironment		Java MQEnvironment		.NET MQEnvironment	.NET MQEnvironment		
Propriedade do connection factory			Propriedade do connection factory			Propriedade do connection factory	Propriedade do connection factory
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	Opções de conexão						
Variável de Ambiente	Variáveis de ambiente				Variáveis de ambiente		Variáveis de ambiente
mqclient.ini (Aplicável apenas às conexões do cliente)	Conexões de cliente				Conexões de cliente		Conexões de cliente
Nome de classe do Java		Nome de classeJava	Nome de classeJava				

	Ligações e cliente C	Ligações e cliente do Java	Ligações e cliente do JMS	Cliente .NET gerenciado	Ligações e cliente .NET não gerenciados	Cliente XMS gerenciado	Ligações e cliente .XMS não gerenciados
nome padrão	Nome Padrão			Nome Padrão .NET	Nome Padrão .NET	Nome Padrão .NET	Nome Padrão .NET

Nota: As ligações C e a coluna do cliente se também aplicam às linguagens de programação a seguir:

- COBOL
- Assembler
- Visual Basic
- RPG

Conexões de linguagem de programação

Os aplicativos que resolvem para as bibliotecas do IBM MQ em C e outras linguagens de programação podem fornecer o nome do aplicativo das maneiras a seguir.

Os métodos de conexão são listados em ordem de precedência, iniciando com o mais alto.

Multi **Opções de conexão**

- ALW MQCNO

  Nota: z/OS Ao se conectar a um gerenciador de filas do IBM MQ for z/OS , é possível configurar o nome do aplicativo apenas usando conexões do modo cliente ou usando aplicativos IBM MQ classes for JMS ou IBM MQ classes for Java .

- IBM i MQCNO no IBM i

ALW **Variáveis de ambiente**

Se você ainda não tiver selecionado um nome do aplicativo, poderá usar a variável de ambiente **MQAPPLNAME** para identificar a conexão com o gerenciador de filas. Por exemplo:

```
export MQAPPLNAME=ExampleApplName
```

Observe que apenas os 28 primeiros caracteres são utilizados e esses caracteres não devem ser todos nulos nem estar todos em branco.

Nota: O atributo se aplica apenas às linguagens de programação suportadas, conexões não gerenciadas .NETe não gerenciadas XMS .

ALW **Arquivo de Configuração do Cliente**

Se você ainda não tiver selecionado um nome de aplicativo e a conexão for uma conexão do cliente, será possível especificar as seguintes informações no arquivo de configuração do cliente (por exemplo, `mqclient.ini`) para identificar a conexão com o gerenciador de filas.

```
Connection:
    ApplName=ExampleApplName
```

Notas:

1. Apenas os primeiros 28 caracteres são usados e esses caracteres não devem ser todos brancos ou nulos.
2. O atributo aplica-se apenas a conexões do cliente nas linguagens de programação suportadas, somente conexões não gerenciadas .NETe não gerenciadas XMS .

Para obter mais informações, consulte IBM MQ MQI client arquivo de configuração, `mqclient.ini`.

Nome Padrão

Se você ainda não tiver escolhido o nome do aplicativo, o nome padrão continuará a ser usado, que contém o máximo do caminho e do nome do executável exibidos pelo sistema operacional. Para obter mais informações, veja PutApplName.

Aplicativos do .NET gerenciados

Os aplicativos do .NET gerenciados podem fornecer o nome do aplicativo das maneiras a seguir.

Os métodos de conexão são listados em ordem de precedência, iniciando com o mais alto.

Substituição de propriedade da conexão

É possível fornecer um arquivo de substituição de detalhes de conexão para os aplicativos da seguinte maneira:

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

O arquivo especificado por `overrideConnectionDetailsFile` contém uma lista de propriedades prefixadas por `mqj`. Os aplicativos precisam definir a propriedade **`mqj.APPNAME`** em que o valor da propriedade **`mqj.APPNAME`** especifica o nome usado para identificar a conexão com o gerenciador de filas.

Apenas os 28 primeiros caracteres do nome são utilizados. Por exemplo:

```
mqj.APPNAME=ExampleApplName
```

Propriedade substituída

Uma constante **`MQC.APPNAME_PROPERTY`** foi definida com o valor *APPNAME*. Agora é possível transmitir essa propriedade para o construtor **`MQQueueManager`** usando apenas os 28 primeiros caracteres do nome. Por exemplo:

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

Para obter informações adicionais, consulte “Operações gerenciadas e não gerenciadas no .NET” na página 644.

MQEnvironment

A propriedade *AppName* é incluída na classe **`MQEnvironment`**e os primeiros 28 caracteres são usados apenas... Por exemplo:

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

Nome Padrão

Se você não tiver fornecido o nome do aplicativo por qualquer um dos meios descritos no texto anterior, o nome do aplicativo será configurado automaticamente para ser o nome executável (e quanto do caminho que caberá).

XMS aplicativos

Os métodos de conexão são listados em ordem de precedência, iniciando com o mais alto.

Propriedade do connection factory

XMS aplicativos podem fornecer o nome do aplicativo no connection factory usando a propriedade **XMSC.WMQ_APPLICATIONNAME** ("*XMSC_WMQ_APPNAME*") em uma maneira semelhante ao JMS É possível especificar até 28 caracteres.

Para obter mais informações, veja "XMS .NET criando objetos administrados" na página 672 e "Propriedades de uma mensagem XMS" na página 680.

JMSAdmin

No conjunto de ferramentas administrativas, a propriedade é conhecida como "**APPLICATIONNAME**" ou "**APPNAME**".

Aplicativos de Ligações doJava e do JMS

Os métodos de conexão são listados em ordem de precedência, iniciando com o mais alto.

ALW Os aplicativos clientes do Java e do JMS já podem especificar um nome do aplicativo e isso foi estendido no IBM MQ for Multiplatforms para aplicativos de ligações, fazendo uso do campo **ApplName** do MQCNO.

Substituição de propriedade da conexão

A propriedade **Application name** foi incluída na lista de propriedades de conexão que podem ser substituídas. Para obter mais informações, consulte Usando substituição de propriedade da conexão IBM MQ..

Atenção: As propriedades da conexão e a maneira de usar o arquivo de Substituição de propriedade de conexão são as mesmas para o IBM MQ classes for Java e o .NET.

Propriedade substituída

Uma constante **MQC.APPNAME_PROPERTY** foi definida com o valor *APPNAME*. Agora é possível transmitir essa propriedade para o construtor **MQQueueManager** usando apenas os 28 primeiros caracteres do nome. Para obter mais informações, consulte Usando substituição de propriedade de conexão em IBM MQ classes for Java.

MQEnvironment

A propriedade *AppName* é incluída na classe **MQEnvironment** e os primeiros 28 caracteres são usados apenas...

Para obter informações adicionais, consulte "Configurando o ambiente do IBM MQ para o IBM MQ classes for Java" na página 381.

Nome de classeJava

Se você não tiver fornecido o nome do aplicativo por qualquer um dos meios no texto anterior, o nome do aplicativo será derivado do nome da classe principal.

Para obter informações adicionais, consulte "Configurando o ambiente do IBM MQ para o IBM MQ classes for Java" na página 381.

Atenção: IBM i No IBM i, não é possível consultar o nome da classe principal, portanto, o IBM MQ client for Java é usado.

Conceitos relacionados

"Configurando o ambiente do IBM MQ para o IBM MQ classes for Java" na página 381
Para que um aplicativo estabeleça conexão com um gerenciador de filas no modo cliente, o aplicativo deve especificar o nome do canal, o nome do host e o número da porta.

Referências relacionadas

MQCNO
MQCNO no IBM i

Técnicas de design para mensagens

Considerações para ajudá-lo a projetar mensagens, incluindo considerações para seletores e propriedades de mensagem.

Coisas a considerar no estágio de design

Você cria uma mensagem ao usar uma chamada MQI para colocar a mensagem em uma fila. Como entrada para a chamada, você fornece algumas informações de controle em um *descritor de mensagens* (MQMD) e os dados que você deseja enviar para outro programa. Mas no estágio de design, é necessário considerar o seguinte, pois afetam a maneira que você cria suas mensagens:

Tipo de mensagem a usar

Você está projetando um aplicativo simples no qual é possível enviar uma mensagem e depois não executar nenhuma ação adicional? Ou está solicitando uma resposta a uma pergunta? Se estiver fazendo uma pergunta, você pode incluir no descritor de mensagens o nome da fila na qual deseja receber a resposta.

Deseja que suas mensagens de solicitação e de resposta sejam síncronas? Isso sugere que você configura um período de tempo limite para a resposta à sua solicitação e, se não receber a resposta dentro desse período, isso será tratado como um erro.

Ou preferiria trabalhar de forma assíncrona, para que seus processos não precisem depender da ocorrência de eventos específicos, como sinais de sincronização comuns?

Outra consideração é se você tem todas as suas mensagens dentro de uma unidade de trabalho.

Atribuindo prioridades diferentes a mensagens

É possível designar um valor de prioridade a cada mensagem e definir a fila de forma que ela mantenha suas mensagens em ordem de prioridade. Se fizer isto, quando outro programa recuperar uma mensagem da fila, sempre obterá a mensagem com a prioridade mais alta. Se a fila não mantiver suas mensagens em ordem de prioridade, um programa que recupera mensagens da fila irá recuperá-las na ordem em que foram incluídas na fila.

Programas também podem selecionar uma mensagem usando o identificador que o gerenciador de filas designou quando a mensagem foi colocada na fila. Como alternativa, é possível gerar seus próprios identificadores para cada uma de suas mensagens.

O efeito de reiniciar o gerenciador de filas nas mensagens

O gerenciador de filas preserva todas as mensagens persistentes, recuperando-as quando necessário a partir de arquivos de log do IBM MQ, quando ele for reiniciado. Mensagens não persistentes e filas dinâmicas temporárias não são preservadas. Quaisquer mensagens que não deseje descartar devem ser definidas como persistentes quando forem criadas. Ao escrever um aplicativo para o IBM MQ for Windows ou o IBM MQ em sistemas AIX and Linux, certifique-se de que saiba como o seu sistema foi configurado com relação à alocação de arquivo de log para reduzir o risco de projetar um aplicativo que será executado para os limites do arquivo de log.

z/OS Como as mensagens nas filas compartilhadas (disponíveis somente no IBM MQ for z/OS) são mantidas no recurso de acoplamento (CF), as mensagens não persistentes são preservadas entre

reinicializações de um gerenciador de filas contanto que o CF permaneça disponível. Se o CF falhar, as mensagens não persistentes serão perdidas.

Fornecendo informações sobre si mesmo ao destinatário de mensagens
Geralmente, o gerenciador de filas configura o ID do usuário, mas os aplicativos devidamente autorizados também podem configurar esse campo, para que seja possível incluir seu próprio ID do usuário e outras informações que o programa de recebimento pode usar para propósitos de contabilidade ou segurança.

Quantia de filas de recebimento

Se uma mensagem pode precisar ser colocada em várias filas, é possível publicar em um tópico ou uma lista de distribuição.

Se uma mensagem pode precisar ser colocada em várias filas, é possível publicar em um tópico.

Seletores e propriedades de mensagem

As mensagens podem ter metadados associados a elas ao lado da carga útil da mensagem principal. Essas propriedades de mensagem podem ser úteis para fornecer dados adicionais.

Há dois aspectos para esses dados adicionais que é importante saber:

- As propriedades não estão sujeitas à proteção do Advanced Message Security (AMS). Se você deseja usar o AMS para proteger seus dados, coloque-o na carga útil e não nas propriedades de mensagem.
- As propriedades podem ser usadas para executar a seleção de mensagens.

É importante observar que o uso de seletores divide a convenção de mensagem padrão de primeiro a entrar, primeiro a sair. Como o gerenciador de filas está otimizado para esta carga de trabalho, fornecer seletores complexos não é aconselhável por motivos de desempenho. O gerenciador de filas não armazena os índices das propriedades de mensagem, portanto, a procura por uma mensagem deve ser uma procura linear. Quanto mais profunda a fila, mais complexo o seletor e menor a probabilidade de que o seletor correspondente a uma mensagem afetará adversamente o desempenho.

Se a seleção complexa for necessária, sugere-se filtrar as mensagens usando qualquer mecanismo de aplicativo ou de processamento, como IBM Integration Bus, para destinos diferentes. Como alternativa, o uso de uma hierarquia de tópicos pode ser útil.

Nota: O IBM MQ classes for Java não suporta o uso de seletores, se você deseja usar seletores, isso deve ser feito por meio da API do JMS.

Considerações de design e desempenho do aplicativo

Há várias maneiras em que um design ruim de programa pode afetar o desempenho. Elas podem ser difíceis de detectar porque o programa pode parecer executar bem, mas afetar o desempenho de outras tarefas. Vários problemas específicos de programas que estão fazendo chamadas do IBM MQ são explicados neste tópico.

Aqui estão algumas ideias para ajudar a projetar os aplicativos eficientes:

- Projete seu aplicativo de forma que o processamento ocorra em paralelo ao tempo de reflexão do usuário:
  - Exiba um painel e permita que o usuário comece a digitar enquanto o aplicativo ainda está inicializando.
  - Obtenha os dados que precisa em paralelo a partir de diferentes servidores.
- Mantenha conexões e filas abertas se for reutilizá-las, em vez de abrir e fechar, conectar e desconectar repetidamente.
- No entanto, um aplicativo do servidor que está efetuando put somente de uma mensagem deve usar MQPUT1.

- Os gerenciadores de filas são otimizados para mensagens com tamanho entre 4 KB e 100 KB. Mensagens muito grandes são ineficientes; é provavelmente melhor enviar 100 mensagens de 1 MB cada do que uma única mensagem de 100 MB. Mensagens muito pequenas também são ineficientes. O gerenciador de filas executa a mesma quantia de trabalho para uma mensagem de byte único que para uma mensagem de 4 KB.
- Mantenha suas mensagens em uma unidade de trabalho, para que possam ser confirmadas ou voltadas simultaneamente.
- Use a opção não persistente para mensagens que não precisam ser recuperáveis.
- Se precisar enviar uma mensagem para diversas filas de destino, considere usar uma lista de distribuição.

Efeito do comprimento da mensagem

A quantia de dados em uma mensagem pode afetar o desempenho do aplicativo que processa a mensagem. Para obter o melhor desempenho de seu aplicativo, envie somente os dados essenciais em uma mensagem. Por exemplo, em uma solicitação para debitar uma conta bancária, as únicas informações que podem precisar ser passadas do cliente para o aplicativo do servidor são o número da conta e o valor do débito.

Efeito de persistência de mensagem

Mensagens persistentes são geralmente registradas. Registrar mensagens reduz o desempenho de seu aplicativo, portanto, use mensagens persistentes somente para dados essenciais. Se os dados de uma mensagem puderem ser descartados se o gerenciador de filas parar ou falhar, use uma mensagem não persistente.

z/OS As operações MQPUT e MQGET para mensagens persistentes serão bloqueadas quando houver espaço do log de recuperação insuficiente para registrar as operações. Tal condição é indicada no log da tarefa do gerenciador de filas por mensagens CSQJ110E e CSQJ111A. Assegure-se de que os processos de monitoramento estejam no local para que tais condições sejam gerenciadas e evitadas.

Procurando uma mensagem específica

A chamada MQGET, geralmente, recupera a primeira mensagem de uma fila. Se você usar a mensagem e os identificadores de correlação (*MsgId* e *CorrelId*) no descritor de mensagens para especificar uma mensagem específica, o gerenciador de filas precisa procurar na fila até localizar essa mensagem. Usar a chamada MQGET dessa forma afeta o desempenho de seu aplicativo.

Filas que contêm mensagens de comprimentos diferentes

Se seu aplicativo não puder usar mensagens de um comprimento fixo, aumente e diminua os buffers dinamicamente para ajustar o tamanho típico de mensagem. Se o aplicativo emitir uma chamada MQGET que falha porque o buffer é muito pequeno, o tamanho dos dados da mensagem é retornado. Inclua o código para seu aplicativo para que o buffer seja redimensionado apropriadamente e a chamada MQGET seja emitida novamente.

Nota: Se você não configurar o atributo **MaxMsgLength** explicitamente, ele será padronizado para 4 MB, que poderá ser muito ineficiente se for usado para influenciar o tamanho do buffer do aplicativo.

Frequência de pontos de sincronização

Os programas que emitem um número muito grande de chamadas MQPUT ou MQGET no ponto de sincronização, sem confirmá-las, podem causar problemas de desempenho. As filas afetadas podem se ficar cheias de mensagens atualmente inacessíveis, enquanto que outras tarefas podem estar esperando para obter essas mensagens. Isso tem implicações em termos de armazenamento e em termos de encadeamentos que estão ligados a tarefas que estão tentando obter mensagens.

Uso da chamada MQPUT1

Use a chamada MQPUT1 somente se tiver uma única mensagem para colocar em uma fila. Se desejar colocar mais de uma mensagem, use a chamada MQOPEN, seguida por uma série de chamadas MQPUT e uma única chamada MQCLOSE.

Número de encadeamentos em uso

Windows Para o IBM MQ for Windows, um aplicativo pode requerer um grande número de encadeamentos. Cada processo do gerenciador de filas tem alocado um número máximo permitido de encadeamentos do aplicativo.

Os aplicativos podem usar encadeamentos em excesso. Considere se o aplicativo leva em consideração essa possibilidade e se executa ações para parar ou relatar esse tipo de ocorrência.

Colocar mensagens persistentes sob o ponto de sincronização

As mensagens persistentes devem ser colocadas e obtidas no ponto de sincronização. Isso acontece porque, ao se obter uma mensagem persistente fora do ponto de sincronização, se a obtenção falhar, não haverá uma maneira de o aplicativo saber se a mensagem foi obtida da fila ou não e, se a mensagem tiver sido obtida da fila, então ela também foi perdida. Ao obter mensagens persistentes no ponto de sincronização, se alguma coisa falhar, a transação será retrocedida e a mensagem persistente não estará perdida porque ela ainda estará na fila.

Da mesma forma, ao colocar mensagens persistentes, coloque-as no ponto de sincronização. Outra razão para colocar e obter mensagens persistentes no ponto de sincronização é que o código da mensagem persistente no IBM MQ é altamente otimizado para o ponto de sincronização. Então, colocar e obter mensagens persistentes no ponto de sincronização é mais rápido do que colocar e obter mensagens persistentes fora do ponto de sincronização.

Se o seu aplicativo coloca mensagens persistentes fora do ponto de sincronização, o gerenciador de filas verifica se ele pode criar um ponto de sincronização implícito em nome do aplicativo. Se o gerenciador de filas pode fazê-lo, ele inclui a colocação dentro desse syncpoint e o confirma automaticamente. Veja “Ponto de sincronização implícito no Multiplatforms” na página 873 para uma descrição mais detalhada.

No entanto, é mais rápido colocar e obter mensagens não persistentes fora do ponto de sincronização, porque o código não persistente no IBM MQ é otimizado para ficar fora do ponto de sincronização. Colocar e obter mensagens persistentes envolve velocidade do disco porque a mensagem persistente é persistida para o disco. Porém, colocar e obter mensagens não persistentes exige velocidades de CPU, porque não há gravação de disco envolvida, nem mesmo ao usar o ponto de sincronização.

Se um aplicativo está recebendo mensagens e não sabe com antecedência se elas são persistentes ou não, a opção GMO MQGMO_SYNCPOINT_IF_PERSISTENT pode ser usada.

Técnicas de design para aplicativos avançados

Ao projetar aplicativos mais avançados, existem algumas técnicas que você pode querer considerar, como esperar mensagens, correlacionar respostas, configurar e usar informações de contexto, iniciar aplicativos automaticamente, gerar relatórios e remover afinidades de mensagens ao usar o armazenamento em cluster.

Para um aplicativo simples IBM MQ, é necessário decidir quais objetos do IBM MQ usar em seu aplicativo e quais tipos de mensagens você deseja usar. Para um aplicativo mais avançado, pode desejar usar algumas das técnicas introduzidas nas seções a seguir.

Esperando mensagens

Um programa que está atendendo a uma fila pode esperar mensagens da seguinte forma:

- Esperando até uma mensagem chegar ou um intervalo de tempo especificado expirar (consulte “Esperando mensagens” na página 809).

- z/OS Somente no IBM MQ for z/OS, configurando um sinal para que o programa seja informado quando uma mensagem chegar. Para obter mais informações, consulte “Signaling” na página 810.
- Estabelecendo uma saída de retorno de chamada para ser acionada quando uma mensagem chegar; consulte “Consumo assíncrono de mensagens do IBM MQ” na página 43.
- Fazendo chamadas periódicas na fila para ver se uma mensagem chegou (*pesquisa*). Isso geralmente não é aconselhável pois pode ter implicações no desempenho.

Correlacionando respostas

Em aplicativos IBM MQ, quando um programa recebe uma mensagem que solicita que ela faça algum trabalho, o programa geralmente envia uma ou mais mensagens de resposta ao solicitante.

Para ajudar o solicitante a associar essas respostas à sua solicitação original, um aplicativo pode configurar um campo *correlation identifier* no descritor de cada mensagem. Os programas copiam, então, o identificador de mensagem da mensagem de solicitação para o campo do identificador de correlação de suas mensagens de resposta.

Configurando e usando informações de contexto

Informações de contexto são usadas para associar mensagens ao usuário que as gerou e para identificar o aplicativo que gerou a mensagem. Essas informações são úteis para segurança, contabilidade, auditoria e determinação de problemas.

Ao criar uma mensagem, é possível especificar uma opção que solicita que o gerenciador de filas associa informações de contexto padrão à sua mensagem.

Para obter mais informações sobre como usar e configurar informações de contexto, consulte “Contexto da mensagem” na página 48.

Iniciando programas IBM MQ automaticamente

Use o IBM MQ *acionamento* para iniciar um programa automaticamente quando as mensagens chegarem em uma fila

É possível configurar condições de acionamento em uma fila para que um programa comece a processar essa fila:

- Toda vez que uma mensagem chegar na fila
- Quando a primeira mensagem chegar na fila
- Quando o número de mensagens na fila atingir um número predefinido

Para obter mais informações sobre acionamento, consulte “Iniciando aplicativos IBM MQ usando acionadores” na página 877. O acionamento é apenas uma da maneira de iniciar um programa automaticamente. Por exemplo, é possível iniciar um programa automaticamente em um cronômetro usando recursos não IBM MQ.

Multi Em Multiplataformas, o IBM MQ pode definir objetos de serviço para iniciar os programas IBM MQ quando o gerenciador de filas é iniciado; consulte Objetos de serviço.

Gerando relatórios do IBM MQ

É possível solicitar relatórios a seguir em um aplicativo:

- Relatórios de exceções
- Relatórios de expiração
- Relatórios de confirmação de chegada (COA)
- Relatórios de confirmação de entrega (COD)
- Relatórios de positive action notification (PAN)

- Relatórios de negative action notification (NAN)

Eles são descritos em “Mensagens de relatório” na página 21.

Clusters e afinidades de mensagens

Antes de começar a usar clusters com diversas definições para a mesma fila, examine seus aplicativos para ver se há algum que requeira uma troca de mensagens relacionadas.

Em um cluster, uma mensagem pode ser roteada para qualquer gerenciador de filas que hospede uma instância da fila apropriada. Portanto, a lógica dos aplicativos com afinidades de mensagens pode ser afetada.

Por exemplo, você pode ter dois aplicativos que dependem de uma série de mensagens que fluem entre eles na forma de perguntas e respostas. Pode ser importante que todas as perguntas sejam enviadas ao mesmo gerenciador de filas e que todas as respostas sejam enviadas de volta ao outro gerenciador de filas. Nessa situação, é importante que a rotina de gerenciamento de carga de trabalho não envie as mensagens para qualquer gerenciador de filas que simplesmente hospeda uma instância da fila adequada.

Sempre que possível, remova as afinidades. Remover as afinidades de mensagens melhora a disponibilidade e escalabilidade de aplicativos.

Para obter mais informações, consulte Manipulando afinidades de mensagens.

Considerações de design e desempenho de aplicativos IBM i

Use essas informações para entender como design do aplicativo, encadeamentos e armazenamento podem afetar o desempenho.

Essas informações são divididas em duas seções:

- “Considerações de design do aplicativo” na página 65
- “Problemas de desempenho específicos” na página 66

Considerações de design do aplicativo

Há várias maneiras em que um design ruim de programa pode afetar o desempenho. Esses problemas podem ser difíceis de detectar porque o programa poderá aparecer ter um bom desempenho, enquanto afeta o desempenho de outras tarefas. Vários problemas específicos dos programas que fazem chamadas do IBM MQ for IBM i são explicados nas seções a seguir.

Para obter mais informações sobre design do aplicativo, consulte “Considerações de design para aplicativos IBM MQ” na página 50.

Efeito do comprimento da mensagem
Embora o IBM MQ for IBM i permita que as mensagens retenham até 100 MB de dados, a quantia de dados em uma mensagem afeta o desempenho do aplicativo que processa a mensagem. Para obter o melhor desempenho de seu aplicativo, enviar somente os dados essenciais em uma mensagem; por exemplo, em uma solicitação para debitar uma conta bancária, as únicas informações que podem precisar ser passadas do cliente para o aplicativo do servidor são o número da conta e o valor do débito.

Efeito de persistência de mensagem
Mensagens persistentes são registradas no diário. Registrar mensagens no diário reduz o desempenho de seu aplicativo, portanto, use mensagens persistentes somente para dados essenciais. Se os dados de uma mensagem puderem ser descartados se o gerenciador de filas parar ou falhar, use uma mensagem não persistente.

Procurando uma mensagem específica
A chamada MQGET, geralmente, recupera a primeira mensagem de uma fila. Se você usar a mensagem e os identificadores de correlação (*MsgId* e *CorrelId*) no descritor de mensagens para

especificar uma determinada mensagem, o gerenciador de filas deverá procurar a fila até encontrar essa mensagem. O uso da chamada MQGET desta maneira afeta o desempenho de seu aplicativo.

Filas que contêm mensagens de comprimentos diferentes
Se as mensagens em uma fila tiverem comprimentos diferentes, para determinar o tamanho de uma mensagem, seu aplicativo pode usar a chamada de MQGET com o campo *BufferLength* configurado como zero para que, mesmo que a chamada falhe, ele retorna o tamanho dos dados da mensagem. O aplicativo poderá, então, repetir a chamada, especificando o identificador da mensagem que mediu em sua primeira chamada e um buffer do tamanho correto. No entanto, se houver outros aplicativos atendendo a mesma fila, poderá perceber que o desempenho do seu aplicativo é reduzido porque sua segunda chamada MQGET gasta tempo procurando uma mensagem que outro aplicativo recuperou no tempo entre suas duas chamadas.

Se o seu aplicativo não puder usar mensagens de um comprimento fixo, outra solução para esse problema é usar a chamada MQINQ para localizar o tamanho máximo de mensagens que a fila pode aceitar, em seguida, use esse valor em sua chamada MQGET. O tamanho máximo de mensagens para uma fila é armazenado no atributo **MaxMsgLen** da fila. Esse método pode usar grandes quantias de armazenamento, no entanto, pois o valor desse atributo da fila pode ser o máximo permitido pelo IBM MQ for IBM i, que pode ser maior que 2 GB.

Frequência de pontos de sincronização
Os programas que emitem várias chamadas MQPUT no ponto de sincronização, sem confirmá-las, podem causar problemas de desempenho. As filas afetadas podem ficar cheias de mensagens atualmente inutilizadas, enquanto outras tarefas podem estar esperando para obter essas mensagens. Esse problema tem implicações em termos de armazenamento e em termos de encadeamentos ligados a tarefas que estão tentando obter mensagens.

Uso da chamada MQPUT1
Use a chamada MQPUT1 somente se tiver uma única mensagem para colocar em uma fila. Se desejar colocar mais de uma mensagem, use a chamada MQOPEN, seguida por uma série de chamadas MQPUT e uma única chamada MQCLOSE.

Número de encadeamentos em uso
Um aplicativo pode requerer muitos encadeamentos. Cada processo do gerenciador de filas tem um número máximo permitido de encadeamentos alocado. Se alguns aplicativos forem problemáticos, isso pode ser devido a seu design usar encadeamentos em excesso. Considere se o aplicativo leva em consideração essa possibilidade e se executa ações para parar ou relatar esse tipo de ocorrência. O número máximo de encadeamentos que o IBM i permite é 4.095. No entanto, o padrão é 64. OIBM MQ disponibiliza até 63 encadeamentos para seus processos.

Problemas de desempenho específicos

Esta seção explica os problemas de armazenamento e desempenho precário.

Problemas de armazenamento
Se você receber a mensagem do sistema `CPF0907. Serious storage condition may exist.` é possível que você esteja preenchendo o espaço associado aos gerenciadores de fila do IBM MQ for IBM i

O seu aplicativo ou o IBM MQ for IBM i está executando com lentidão?
Se o seu aplicativo estiver executando lentamente, isso pode indicar que está em um loop ou esperando um recurso que não está disponível. Essa execução lenta também pode ser causada por um problema de desempenho. Talvez seja porque o sistema está operando perto dos limites da sua capacidade. Esse tipo de problema é provavelmente pior nos horários de pico de carga do sistema, geralmente no meio da manhã e da tarde. (Se a sua rede se estender por mais de um fuso horário, o carregamento do sistema de pico poderá parecer ocorrer em algum outro momento.)

Se achar que a degradação do desempenho não depende do carregamento do sistema, mas ocorre às vezes quando o sistema está levemente carregado, provavelmente o culpado é um programa de aplicativo mal projetado. Esse problema pode ser manifestar como um problema que ocorre somente quando determinadas filas são acessadas.

QTOTJOB e QADLTOTJ são valores do sistema que vale a pena investigar.

Os sintomas a seguir podem indicar que o IBM MQ for IBM i esteja executando lentamente:

- Se o seu sistema estiver lento para responder a comandos MQSC.
- Se exibições repetidas da profundidade da fila indicarem que a fila está sendo processada lentamente para um aplicativo com o qual você esperaria uma grande quantia de atividade da fila.
- O rastreio do IBM MQ está em execução?

Linux Considerações de design para aplicativos Linux on Power Systems - Little Endian

Como o Linux on Power Systems - Little Endian suporta apenas aplicativos de 64 bits, não há suporte fornecido em IBM MQ para aplicativos de 32 bits.

Conceitos relacionados

"Considerações de design para aplicativos IBM MQ" na página 50

Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

z/OS Design and performance considerations for z/OS applications

Application design is one of the most important factors affecting performance. Use this topic to understand some of the design factors involved in performance.

There are a number of ways in which poor program design can affect performance. These problems can be difficult to detect because the program can appear to perform well, while affecting the performance of other tasks. Several problems specific to programs making MQI calls are demonstrated in the following sections.

For more information about application design, see "Considerações de design para aplicativos IBM MQ" on page 50.

Effect of message length

Although IBM MQ for z/OS allows messages to hold up to 100 MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, send only the essential data in a message. For example, in a request to debit a bank account, the only information that might need to be passed from the client to the server application is the account number and the amount to debit.

Effect of message persistence

Persistent messages are logged. Logging messages reduces the performance of your application, so use persistent messages for essential data only. If the data in a message can be discarded if the queue manager stops or fails, use a nonpersistent message.

Data for persistent messages is written to log buffers. These buffers are written to the log data sets when:

- A commit occurs
- A message is got or put out of syncpoint
- WRTHRSH buffers are filled

Processing many messages in one unit of work can cause less input/output than if the messages were processed one for each unit of work, or out of syncpoint.

Searching for a particular message

The MQGET call typically retrieves the first message from a queue. If you use the message and correlation identifiers (**MsgId** and **CorrelId**) in the message descriptor to specify a particular message, the queue manager searches the queue until it finds that message. Using MQGET in this way affects the performance of your application because, to find a particular message, IBM MQ might have to scan the entire queue.

You can use the **IndexType** queue attribute to specify that you want the queue manager to maintain an index that can be used to increase the speed of MQGET operations on the queue. However, there is a small performance reduction for maintaining an index, so only generate one if you need to use it. You can choose to build an index of message identifiers or of correlation identifiers, or you can choose not to build an index for queues where messages are retrieved sequentially. Try to have many different key values, not lots with the same value. For example Balance1, Balance2, and Balance3, not three with Balance. For shared queues, you must have the correct **IndexType**. For details of the **IndexType** queue attribute, see IndexType.

To avoid affecting queue manager restart time by using indexed queues, use the QINDXBLD(NOWAIT) parameter in the CSQ6SYSP macro. This allows the queue manager restart to complete without waiting for queue index building to complete.

For a full description of the **IndexType** attribute, and other object attributes see Attributes of objects.

Queues that contain messages of different lengths

Get a message, using a buffer size matching the expected size of the message. If you receive the return code indicating that the message is too long, get a bigger buffer. When the get fails in this way, the data length returned is the size of the unconverted message data. If you specify MQGMO_CONVERT on the MQGET call, and the data expands during conversion, it still might not fit in the buffer, in which case you need to further increase the size of the buffer.

If you issue the MQGET with a buffer length of zero, it returns the size of the message and the application can then get a buffer of this size and reissue the get. If you have multiple applications processing the queue, another application might have already processed the message when the original application reissued the get. If you occasionally have large messages, you might need to get a large buffer just for these messages, and release it after the message has been processed. This should help reduce virtual storage problems if all applications have large buffers.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the **MaxMsgL** attribute of the queue. This method could use large amounts of storage, however, because the value of **MaxMsgL** could be as high as 100 MB, the maximum allowed by IBM MQ for z/OS.

Note: You can lower the **MaxMsgL** parameter after large messages have been put to the queue. For example you can put a 100 MB message, then set **MaxMsgL** to 50 bytes. This means that it is still possible to get bigger messages than the application expected.

Frequency of syncpoints

Programs that issue many MQPUT calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently unusable, while other tasks might be waiting to get these messages. This has implications in terms of storage, and in terms of threads tied up with tasks that are attempting to get messages.

As a rule if you have multiple applications processing a queue you typically get the best performance when you have either

- 100 short messages (less than 1 KB), or
- One message for larger messages (100 KB)

for each syncpoint. If there is only one application processing the queue, you must have more messages for each unit of work.

You can limit the number of messages that a task can get or put within a single unit of recovery with the **MAXUMSGS** queue manager attribute. For information about this attribute, see the **ALTER QMGR** command in MQSC commands.

Advantages of the MQPUT1 call

Use the MQPUT1 call only if you have a single message to put on a queue. If you want to put more than one message, use the MQOPEN call, followed by a series of MQPUT calls and a single MQCLOSE call.

How many messages can a queue manager contain

Local Queues

The number of local messages a queue manager can hold is basically the size of the page sets. You can have up to 100 page sets (though it is recommended page set 0 and page set 1 are for system related objects and queues). You can use a page set with extended format and increase the capacity of a page set.

Shared Queues

The capacity for shared queues depends on the size of the coupling facility (CF). IBM MQ uses CF list structures where fundamental storage units are entries and elements. Each message is stored as 1 entry and multiple elements containing the associated MQMD and other message data. The number of elements consumed by a single message depends on the size of the message and, for CFLEVEL(5), the offload rules in effect at MQPUT time. Fewer elements are needed when message data is offloaded to either Db2 or SMDS. Message data access is slower when the message has been offloaded. See Performance Supportpac MP1H for further comparison of performance and CPU overhead associated with message offload.

What affects performance

Performance can mean how fast messages can be processed, and it can also mean how much CPU is needed per message.

What affects how fast messages can be processed

For persistent messages the biggest impact is the speed of the log data sets. The speed of the log data sets depends on the DASD they are on. Therefore care should be taken to put log data set on low used volumes to reduce contention. Striping the MQ logs improves the log performance when there are multiple pages written per I/O. Z High Performance Fibre connection (zHPF) also has a significant performance to I/O response time when the I/O subsystem is busy.

When there is a request to get and put a message, access to the queue is locked during the request to preserve integrity of the queue. For planning purposes consider the queue locked for the whole request. So if the time for a put is 100 microseconds, and you have more than 10,000 requests a second you might experience delays. You might achieve better than this in practice, but it is a good general rule. You can use different queues to improve performance.

Possible reasons for this can be:

- use a common reply queue which every CICS transaction uses
- each CICS transaction is given a unique reply to queue
- a reply to a queue for CICS region and all transactions in the CICS region use this queue.

The answer depends on the number of requests a second, and the response time of the requests.

If messages have to be read from a page set, they will be slower compared to when the messages are in the buffer pool. If you have more messages than fit into a buffer pool, then they will spill to disk. So you need to ensure that the buffer pool is big enough for your short lived messages. If you have

messages that you process many hours later, these are likely to spill to disk, so you should expect a get for these messages to be slower than if they were in the buffer pool.

For a shared queue, the speed of the messages depends on the speed of the Coupling Facility. A CF within the physical processor is likely to be faster than an external CF. The CF response time depends on how busy the CF is. For example on the Hursley systems, when the CF was 17% busy the response time was 14 microseconds. When the CF was 95% busy the response time was 45 microseconds.

If your MQ requests use a lot of CPU, this can affect how fast messages are processed. Because if the Logical Partition (LPAR) is constrained for CPU, applications will be delayed waiting for CPU.

How much CPU per message

In general bigger messages use more CPU, so avoid large (x MB) messages if possible.

When getting specific messages from queues, the queue should be indexed so the queue manager can go directly to the message (and so avoids potentially an entire scan of the queue). If the queue is not indexed then the queue is scanned from the beginning looking for the message. If there are 1000 messages on the queue, it may have to scan all 1000 messages. The result is a lot of unnecessary CPU usage.

Channels using TLS have an additional cost due to the encryption of the message.

In MQ V7 you can select messages by a selector string in addition to the **CORRELID** or **MSGID**. Every message has to be looked in, so if there are many messages on the queue this is expensive.

It is more efficient for an application to do OPEN PUT PUT CLOSE than PUT1 PUT1.

Triggering in CICS

When the message arrival rate of messages for a triggered queue is low, it is efficient to use trigger first. When the message arrival rate is more than 10 messages a second, it is more efficient to trigger the first transaction, then have the transaction process a message and get the next message, and so on. If a message has not arrived in a short period (say between 0.1 and 1 second) the transaction ends. At high throughput you might need multiple transactions running to process the messages and to prevent a build up of messages. For every trigger message produced, this requires a put and a get of a trigger message, which in effect doubles the cost of the message.

How many connections or concurrent users are supported

Each connection uses virtual storage within the queue manager, so the more concurrent users the more storage used. If you need a very large buffer pool and large number of users, then you might be constrained for virtual storage, and you might need to reduce the size of your buffer pools.

If security is being used, the queue manager caches information within the queue manager for a long period. The amount of virtual storage that is used within the queue manager is affected.

The **CHINIT** can support up to about 10,000 connections. This is limited by virtual storage. If a connection uses more storage, for example using by TLS, the storage per connection increases, which therefore means the **CHINIT** can support less connections. If you are processing large messages, these will require more storage for buffers in the **CHINIT**, so the **CHINIT** can support less messages.

Connections to a remote queue manager are more efficient than client connections. For example, every MQ client requests requires two network flows (one for the request and one for the response). With a channel to a remote queue manager, there may be 50 sends over the network before a response comes back. If you are considering a large client network, it may be more efficient to use a concentrator queue manager on a distributed box, and have one channel coming in and out of the concentrator.

Other things affecting performance

Log data set should be at least 1000 cylinders in size. If the logs are smaller than this, checkpoint activity may be too frequent. On a busy system a checkpoint typically should be every 15 minutes or longer, at very high throughputs it may less than this. When a checkpoint occurs the buffer pools are scanned

and 'old' messages and changed pages are written to disk. If checkpoints are too frequent, this can impact performance. The value of LOGLOAD can also affect checkpoint frequency. If the queue manager abnormally ends, then at restart it may have to read back to 3 checkpoints. The best checkpoint interval is a balance between the activity when a checkpoint is taken, and the amount of log data that may need to be read when the queue manager restarts.

There is a significant overhead incurred when starting a channel. It is usually better to start a channel and leave it connected, rather than frequent starts and stops of the channel.

Related information

MP1K: IBM MQ for z/OS 9.0 Performance Report

z/OS IMS and IMS bridge applications on IBM MQ for z/OS

This information helps you to write IMS applications using IBM MQ.

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using IBM MQ” on page 71.
- To write applications that use the IBM MQ - IMS bridge, see “Writing IMS bridge applications” on page 75.

Use the following links to find out more about IMS and IMS bridge applications on IBM MQ for z/OS:

- “Writing IMS applications using IBM MQ” on page 71
- “Writing IMS bridge applications” on page 75

Related concepts

“Visão geral da Message Queue Interface” on page 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” on page 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Abrindo e fechando objetos” on page 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” on page 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” on page 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” on page 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” on page 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” on page 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” on page 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” on page 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

z/OS Writing IMS applications using IBM MQ

There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

Use the following links to find out more about writing IMS applications on IBM MQ for z/OS:

- “Syncpoints in IMS applications” on page 72
- “MQI calls in IMS applications” on page 72

Restrictions

There are restrictions on which IBM MQ API calls can used by an application using the IMS adapter.

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

Related concepts

“Writing IMS bridge applications” on page 75
This topic contains information about writing applications to use the IBM MQ - IMS bridge.

z/OS *Syncpoints in IMS applications*

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint).

To back out all changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see ROLB call in the IMS documentation.

The queue manager is a participant in a two-phase commit protocol; the IMS syncpoint manager is the coordinator.

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify IBM MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason, applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application ends normally, any open queues are closed and an implicit commit occurs. If the application ends abnormally, any open queues are closed and an implicit backout occurs.

z/OS *MQI calls in IMS applications*

Use this information to learn about the use of MQI calls on Server applications and Enquiry applications.

This section covers the use of MQI calls in the following types of IMS applications:

- “Server applications” on page 72
- “Inquiry applications” on page 75

Server applications

Here is an outline of the MQI server application model:

```
Initialize/Connect
.
Open queue for input shared
.
```

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

Sample program CSQ4ICB3 shows the implementation, in C/370, of a BMP using this model. The program establishes communication with IMS first, and then with IBM MQ:

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

Return
```

The IMS initialization determines whether the program has been called as a message-driven or a batch-oriented BMP and controls IBM MQ queue manager connection and queue handles accordingly:

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

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

The IBM MQ initialization connects to the queue manager and opens the queues. In a message-driven BMP this is called after each IMS syncpoint is taken; in a batch-oriented BMP, this is called only during program startup:

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

Return to calling function
```

The implementation of the server model in an MPP is influenced by the fact that the MPP processes a single unit of work per invocation. This is because, when a syncpoint (GU) is taken, the connection and queue handles are closed and the next IMS message is delivered. This limitation can be partially overcome by one of the following:

- **Processing many messages within a single unit-of-work**

  This involves:

  – Reading a message
  – Processing the required updates
  – Putting the reply

  in a loop until all messages have been processed or until a set maximum number of messages has been processed, at which time a syncpoint is taken.

  Only certain types of application (for example, a simple database update or inquiry) can be approached in this way. Although the MQI reply messages can be put with the authority of the originator of the MQI message being handled, the security implications of any IMS resource updates need to be addressed carefully.

- **Processing one message per invocation of the MPP and ensuring multiple scheduling of the MPP to process all available messages.**

  Use the IBM MQ IMS trigger monitor program (CSQQTRMN) to schedule the MPP transaction when there are messages on the IBM MQ queue and no applications serving it.

  If trigger monitor starts the MPP, the queue manager name and queue name are passed to the program, as shown in the following COBOL code extract:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
```

```
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

The server model, which is expected to be a long running task, is better supported in a batch processing region, although the BMP cannot be triggered using CSQQTRMN.

Inquiry applications

A typical IBM MQ application initiating an inquiry or update works as follows:

- Gather data from the user
- Put one or more IBM MQ messages
- Get the reply messages (you might have to wait for them)
- Provide a response to the user

Because messages put on to IBM MQ queues do not become available to other IBM MQ applications until they are committed, they must either be put out of syncpoint, or the IMS application must be split into two transactions.

If the inquiry involves putting a single message, you can use the *no syncpoint* option; however, if the inquiry is more complex, or resource updates are involved, you might get consistency problems if failure occurs and you do not use syncpointing.

To overcome this, you can split IMS MPP transactions using MQI calls using a program-to-program message switch; see *IMS Intersystem Communication (ISC)* for information about this. This allows an inquiry program to be implemented in an MPP:

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

z/OS Writing IMS bridge applications

This topic contains information about writing applications to use the IBM MQ - IMS bridge.

For information about the IBM MQ - IMS bridge, see The IMS bridge.

Use the following links to find out more about writing IMS bridge applications on IBM MQ for z/OS:

- “How the IMS bridge deals with messages” on page 76
- “Writing IMS transaction programs through IBM MQ” on page 923

Related concepts

“Writing IMS applications using IBM MQ” on page 71
There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

z/OS *How the IMS bridge deals with messages*

When you use the IBM MQ - IMS bridge to send messages to an IMS application, you need to construct your messages in a special format.

You must also put your messages on IBM MQ queues that have been defined with a storage class that specifies the XCF group and member name of the target IMS system. These are known as MQ-IMS bridge queues, or simply **bridge** queues.

The IBM MQ-IMS bridge requires exclusive input access (MQOO_INPUT_EXCLUSIVE) to the bridge queue if it is defined with QSGDISP(QMGR), or if it is defined with QSGDISP(SHARED) together with the NOSHARE option.

A user does not need to sign on to IMS before sending messages to an IMS application. The user ID in the *UserIdentifier* field of the MQMD structure is used for security checking. The level of checking is determined when IBM MQ connects to IMS, and is described in Application access control for the IMS bridge. This enables a pseudo signon to be implemented.

The IBM MQ - IMS bridge accepts the following types of message:

- Messages containing IMS transaction data and an MQIIH structure (described in MQIIH):

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

  Note:

  1. The square brackets, [ ], represent optional multi-segments.
  2. Set the *Format* field of the MQMD structure to MQFMT_IMS to use the MQIIH structure.

- Messages containing IMS transaction data but no MQIIH structure:

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

IBM MQ validates the message data to ensure that the sum of the LL bytes plus the length of the MQIIH (if it is present) is equal to the message length.

When the IBM MQ - IMS bridge gets messages from the bridge queues, it processes them as follows:

- If the message contains an MQIIH structure, the bridge verifies the MQIIH (see MQIIH), builds the OTMA headers, and sends the message to IMS. The transaction code is specified in the input message. If this is an LTERM, IMS replies with a DFS1288E message. If the transaction code represents a command, IMS executes the command; otherwise the message is queued in IMS for the transaction.
- If the message contains IMS transaction data, but no MQIIH structure, the IMS bridge makes the following assumptions:
  - The transaction code is in bytes 5 through 12 of the user data
  - The transaction is in nonconversational mode
  - The transaction is in commit mode 0 (commit-then-send)
  - The *Format* in the MQMD is used as the *MFSMapName* (on input)
  - The security mode is MQISS_CHECK

  The reply message is also built without an MQIIH structure, taking the *Format* for the MQMD from the *MFSMapName* of the IMS output.

The IBM MQ - IMS bridge uses one or two Tpipes for each IBM MQ queue:

- A synchronized Tpipe is used for all messages using Commit mode 0 (COMMIT_THEN_SEND) (these show with SYN in the status field of the IMS /DIS TMEMBER client TPIPE xxxx command)
- A non-synchronized Tpipe is used for all messages using Commit mode 1 (SEND_THEN_COMMIT)

The Tpipes are created by IBM MQ when they are first used. A non-synchronized Tpipe exists until IMS is restarted. Synchronized Tpipes exist until IMS is cold started. You cannot delete these Tpipes yourself.

See the following topics for more information about how the IBM MQ - IMS bridge deals with messages:

- "Mapping IBM MQ messages to IMS transaction types" on page 77
- "If the message cannot be put to the IMS queue" on page 77
- "IMS bridge feedback codes" on page 78
- "The MQMD fields in messages from the IMS bridge" on page 78
- "The MQIIH fields in messages from the IMS bridge" on page 79
- "Reply messages from IMS" on page 80
- "Using alternate response PCBs in IMS transactions" on page 81
- "Sending unsolicited messages from IMS" on page 81
- "Message segmentation" on page 81
- "Data conversion for messages to and from the IMS bridge" on page 81

Related concepts

"Writing IMS transaction programs through IBM MQ" on page 923

The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

z/OS *Mapping IBM MQ messages to IMS transaction types*

A table describing the mapping of IBM MQ messages to IMS transaction types.

Table 4. How IBM MQ messages map to IMS transaction types

IBM MQ message type	Commit-then-send (mode 0) - uses synchronized IMS Tpipes	Send-then-commit (mode 1) - uses non-synchronized IMS Tpipes
Persistent IBM MQ messages	• Recoverable full function transactions • Unrecoverable transactions are rejected by IMS	• Fastpath transactions • Conversational transactions • Full function transactions
Nonpersistent IBM MQ messages	• Unrecoverable full function transactions • Recoverable transactions are permitted with IMS V8 and APAR PQ61404 and all later versions of IMS	• Fastpath transactions • Conversational transactions • Full function transactions

Note: IMS commands cannot use persistent IBM MQ messages with commit mode 0. See Commit mode (commitMode) for more information.

z/OS *If the message cannot be put to the IMS queue*

Learn about actions to take if the message cannot be put to the IMS queue.

If the message cannot be put to the IMS queue, the following action is taken by IBM MQ:

- If a message cannot be put to IMS because the message is invalid, the message is put to the dead-letter queue, and a message is sent to the system console.
- If the message is valid, but is rejected by IMS, IBM MQ sends an error message to the system console, the message includes the IMS sense code, and the IBM MQ message is put to the dead-letter queue. If the IMS sense code is 001A, IMS sends an IBM MQ message containing the reason for the failure to the reply-to queue.

  Note: In the circumstances listed previously, if IBM MQ cannot put the message to the dead-letter queue for any reason, the message is returned to the originating IBM MQ queue. An error message is sent to the system console, and no further messages are sent from that queue.

  To resend the messages, do ***one*** of the following:

  – Stop and restart the Tpipes in IMS corresponding to the queue
  – Alter the queue to GET(DISABLED), and again to GET(ENABLED)
  – Stop and restart IMS or the OTMA
  – Stop and restart your IBM MQ subsystem
- If the message is rejected by IMS for anything other than a message error, the IBM MQ message is returned to the originating queue, IBM MQ stops processing the queue, and an error message is sent to the system console.

  If an exception report message is required, the bridge puts it to the reply-to queue with the authority of the originator. If the message cannot be put to the queue, the report message is put to the dead-letter queue with the authority of the bridge. If it cannot be put to the DLQ, it is discarded.

z/OS *IMS bridge feedback codes*

IMS sense codes are typically output in hexadecimal format in IBM MQ console messages such as CSQ2001I (for example, sense code 0x001F). IBM MQ feedback codes as seen in the dead-letter header of messages put to the dead-letter queue are decimal numbers.

The IMS bridge feedback codes are in the range 301 through 399, or 600 through 855 for NACK sense code 0x001A. They are mapped from the IMS-OTMA sense codes as follows:

1. The IMS-OTMA sense code is converted from a hexadecimal number to a decimal number.
2. 300 is added to the number resulting from the calculation in 1, giving the IBM MQ *Feedback* code.
3. The IMS-OTMA sense code 0x001A, decimal 26 is a special case. A *Feedback* code in the range 600-855 is generated.
   a. The IMS-OTMA reason code is converted from a hexadecimal number to a decimal number.
   b. 600 is added to the number resulting from the calculation in a, giving the IBM MQ *Feedback* code.

For information about IMS-OTMA sense codes, see OTMA sense codes for NAK messages.

z/OS *The MQMD fields in messages from the IMS bridge*

Learn about the MQMD fields in messages from the IMS bridge.

The MQMD of the originating message is carried by IMS in the User Data section of the OTMA headers. If the message originates in IMS, this is built by the IMS Destination Resolution Exit. The MQMD of a message received from IMS is built as follows:

StrucID
"MD "

Version
MQMD_VERSION_1

Report
MQRO_NONE

MsgType
MQMT_REPLY

Expiry
If MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH, this field contains the remaining expiry time, else it is set to MQEI_UNLIMITED

Feedback
MQFB_NONE

Encoding
MQENC.Native (the encoding of the z/OS system)

CodedCharSetId
MQCCSI_Q_MGR (the CodedCharSetID of the z/OS system)

Format
MQFMT_IMS if the MQMD.Format of the input message is MQFMT_IMS, otherwise IOPCB.MODNAME

Priority
MQMD.Priority of the input message

Persistence
Depends on commit mode: MQMD.Persistence of the input message if CM-1; persistence matches recoverability of the IMS message if CM-0

MsgId
MQMD.MsgId if MQRO_PASS_MSG_ID, otherwise New MsgId (the default)

CorrelId
MQMD.CorrelId from the input message if MQRO_PASS_CORREL_ID, otherwise MQMD.MsgId from the input message (the default)

BackoutCount
0

ReplyToQ
Blanks

ReplyToQMgr
Blanks (set to local qmgr name by the queue manager during the MQPUT)

UserIdentifier
MQMD.UserIdentifier of the input message

AccountingToken
MQMD.AccountingToken of the input message

ApplIdentityData
MQMD.ApplIdentityData of the input message

PutApplType
MQAT_XCF if no error, otherwise MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
Date when message was put

PutTime
Time when message was put

ApplOriginData
Blanks

z/OS *The MQIIH fields in messages from the IMS bridge*

Learn about the MQIIH fields in messages from the IMS bridge.

The MQIIH of a message received from IMS is built as follows:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat of the input message if MQIIH.ReplyToFormat is not blank, otherwise IOPCB.MODNAME

Flags
0

LTermOverride
LTERM name (Tpipe) from OTMA header

MFSMapName
Map name from OTMA header

ReplyToFormat
Blanks

Authenticator
MQIIH.Authenticator of the input message if the reply message is being put to an MQ-IMS bridge queue, otherwise blanks.

TranInstanceId
Conversation ID / Server Token from OTMA header if in conversation. In versions of IMS prior to V14, this field is always nulls if not in conversation. From IMS V14 onwards, this field may be set by IMS even if not in conversation.

TranState
"C" if in conversation, otherwise blank

CommitMode
Commit mode from OTMA header ("0" or "1")

SecurityScope
Blank

Reserved
Blank

z/OS *Reply messages from IMS*

When an IMS transaction ISRTs to its IOPCB, the message is routed back to the originating LTERM or TPIPE.

These are seen in IBM MQ as reply messages. Reply messages from IMS are put onto the reply-to queue specified in the original message. If the message cannot be put onto the reply-to queue, it is put onto the dead-letter queue using the authority of the bridge. If the message cannot be put onto the dead-letter queue, a negative acknowledgment is sent to IMS to say that the message cannot be received. Responsibility for the message is then returned to IMS. If you are using commit mode 0, messages from that Tpipe are not sent to the bridge, and remain on the IMS queue; that is, no further messages are sent until restart. If you are using commit mode 1, other work can continue.

If the reply has an MQIIH structure, its format type is MQFMT_IMS; if not, its format type is specified by the IMS MOD name used when inserting the message.

z/OS *Using alternate response PCBs in IMS transactions*

When an IMS transaction uses alternate response PCBs (ISRTs to the ALTPCB, or issues a CHNG call to a modifiable PCB), the pre-routing exit (DFSYPRX0) is invoked to determine if the message should be rerouted.

If the message is to be rerouted, the destination resolution exit (DFSYDRU0) is invoked to confirm the destination and prepare the header information See Using OTMA exits in IMS and The pre-routing exit DFSYPRX0 for information about these exit programs.

Unless action is taken in the exits, all output from IMS transactions initiated from an IBM MQ queue manager, whether to the IOPCB or to an ALTPCB, will be returned to the same queue manager.

z/OS *Sending unsolicited messages from IMS*

To send messages from IMS to an IBM MQ queue, you need to invoke an IMS transaction that ISRTs to an ALTPCB.

You need to write pre-routing and destination resolution exits to route unsolicited messages from IMS and build the OTMA user data, so that the MQMD of the message can be built correctly. See The pre-routing exit DFSYPRX0 and The destination resolution user exit for information about these exit programs.

Note: The IBM MQ - IMS bridge does not know whether a message that it receives is a reply or an unsolicited message. It handles the message the same way in each case, building the MQMD and MQIIH of the reply based on the OTMA UserData that arrived with the message

Unsolicited messages can create new Tpipes. For example, if an existing IMS transaction switched to a new LTERM (for example PRINT01), but the implementation requires that the output be delivered through OTMA, a new Tpipe (called PRINT01 in this example) is created. By default, this is a non-synchronized Tpipe. If the implementation requires the message to be recoverable, set the destination resolution exit output flag. See the *IMS Customization Guide* for more information.

z/OS *Message segmentation*

You can define IMS transactions as expecting single- or multi-segment input.

The originating IBM MQ application must construct the user input following the MQIIH structure as one or more LLZZ-data segments. All segments of an IMS message must be contained in a single IBM MQ message sent with a single MQPUT.

The maximum length of an LLZZ-data segment is defined by IMS/OTMA (32767 bytes). The total IBM MQ message length is the sum of the LL bytes, plus the length of the MQIIH structure.

All the segments of the reply are contained in a single IBM MQ message.

There is a further restriction on the 32 KB limitation on messages with format MQFMT_IMS_VAR_STRING. When the data in an ASCII-mixed CCSID message is converted to an EBCDIC-mixed CCSID message, a shift-in byte or a shift-out byte is added every time that there is a transition between SBCS and DBCS characters. The 32 KB restriction applies to the maximum size of the message. That is, because the LL field in the message cannot exceed 32 KB, the message must not exceed 32 KB including all shift-in and shift-out characters. The application building the message must allow for this.

z/OS *Data conversion for messages to and from the IMS bridge*

The data conversion is performed by either the distributed queuing facility (which may call any necessary exits) or by the intra group queuing agent (which does not support the use of exits) when it puts a message to a destination queue that has XCF information defined for its storage class. The data conversion does not occur when a message is delivered to a queue by publish/subscribe.

Any exits needed must be available to the distributed queuing facility in the data set referenced by the CSQXLIB DD statement. This means that you can send messages to an IMS application using the IBM MQ - IMS bridge from any IBM MQ platform.

If there are conversion errors, the message is put to the queue unconverted; this results eventually in it being treated as an error by the IBM MQ - IMS bridge, because the bridge cannot recognize the header format. If a conversion error occurs, an error message is sent to the z/OS console.

See “Escrevendo saídas de conversão de dados” on page 995 for detailed information about data conversion in general.

Sending messages to the IBM MQ - IMS bridge

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIIH structure, the *Format* in the MQMD must be set to the built-in format MQFMT_IMS, and the *Format* in the MQIIH must be set to the name of the format that describes your message data. If there is no MQIIH, set the *Format* in the MQMD to your format name.

If your data (other than the LLZZs) is all character data (MQCHAR), use as your format name (in the MQIIH or MQMD, as appropriate) the built-in format MQFMT_IMS_VAR_STRING. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any MQIIH at the start of the message).

If your application uses *MFSMapName*, you can use messages with the MQFMT_IMS instead, and define the map name passed to the IMS transaction in the MFSMapName field of the MQIIH.

Receiving messages from the IBM MQ - IMS bridge

If an MQIIH structure is present on the original message that you are sending to IMS, one is also present on the reply message.

To ensure that your reply is converted correctly:

- If you have an MQIIH structure on your original message, specify the format that you want for your reply message in the MQIIH *ReplytoFormat* field of the original message. This value is placed in the MQIIH *Format* field of the reply message. This is particularly useful if all your output data is of the form LLZZ<character data>.
- If you do not have an MQIIH structure on your original message, specify the format that you want for the reply message as the MFS MOD name in the IMS application's ISRT to the IOPCB.

Desenvolvendo aplicativos JMS/Jakarta Messaging e Java

O IBM MQ fornece três interfaces de linguagem Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSe IBM MQ classes for Java.

Sobre esta tarefa

JM 3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging é um provedor Jakarta Messaging que implementa a interface Jakarta Messaging para IBM MQ como o sistema de mensagens. O Jakarta Connectors Architecture fornece uma maneira padrão de conectar aplicativos em execução em um ambiente do Jakarta EE a um Enterprise Information System (EIS), como IBM MQ ou Db2.

Para obter mais informações, veja “Por que devo usar o IBM MQ classes for Jakarta Messaging?” na página 84 e “Acessando IBM MQ de Java -Opção de API” na página 87.

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS é um provedor JMS que implementa a interface JMS para IBM MQ como o sistema de mensagens. O Java Platform, Enterprise Edition Connector Architecture (JCA) fornece uma maneira padrão de conectar aplicativos em execução em um ambiente Java EE a um Enterprise Information System (EIS) como IBM MQ ou Db2.

Para obter mais informações, veja “Por que devo usar o IBM MQ classes for JMS?” na página 86 e “Acessando IBM MQ de Java -Opção de API” na página 87.

IBM MQ classes for Java

O IBM MQ classes for Java permite usar o IBM MQ em um ambiente Java. O IBM MQ classes for Java permite que um aplicativo Java se conecte ao IBM MQ como um cliente IBM MQ ou se conecte diretamente a um gerenciador de filas do IBM MQ.

O IBM MQ classes for Java encapsula a Message Queue Interface (MQI), a API nativa do IBM MQ e usa o mesmo modelo de objeto que outras interfaces orientadas a objetos, enquanto IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging implementam Java interfaces do sistema de mensagens do Oracle e do Java Community Process , respectivamente.

Para obter mais informações, veja “Por que devo usar o IBM MQ classes for Java?” na página 354 e “Acessando IBM MQ de Java -Opção de API” na página 87.

Nota:

Stabilized A IBM não fará aprimoramentos adicionais no IBM MQ classes for Java e ele ficará funcionalmente estabilizado no nível enviado no IBM MQ 8.0. Os aplicativos existentes que usam o IBM MQ classes for Java continuam sendo totalmente suportados, mas novos recursos não serão incluídos e as solicitações de aprimoramentos serão rejeitadas. "Totalmente suportado" significa que os defeitos serão corrigidos, além de quaisquer mudanças necessárias, por meio de mudanças nos requisitos do sistema IBM MQ.

O IBM MQ classes for Java não é suportado no IMS.

O IBM MQ classes for Java não é suportado no WebSphere Liberty. Eles não devem ser usados com o recurso do sistema de mensagens do IBM MQ Liberty ou com o suporte genérico do JCA. Para obter mais informações, consulte Usando Interfaces do WebSphere MQ Java em J2EE/JEE .

Usando IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

JM 3.0 A partir do IBM MQ 9.3.0, o Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. O IBM MQ 9.3.0 continua a suportar JMS 2.0 para aplicativos existentes. Não é suportado usar a API JMS 2.0 e a API Jakarta Messaging 3.0 no mesmo aplicativo.

Nota: Para Jakarta Messaging 3.0, o controle da especificação JMS move de Oracle para o Java Community Process. No entanto, o Oracle retém o controle do nome "javax", que é usado em outras tecnologias Java que não foram movidas para o Java Community Process. Portanto, enquanto Jakarta Messaging 3.0 é funcionalmente equivalente a JMS 2.0 , há algumas diferenças na nomenclatura:

- O nome oficial da versão 3.0 é Jakarta Messaging em vez de Java Message Service.
- Os nomes de pacotes e constantes são prefixados com `jakarta` em vez de `javax`. Por exemplo, no JMS 2.0 , a conexão inicial com um provedor de sistemas de mensagens é um objeto `javax.jms.Connection` e no Jakarta Messaging 3.0 é um objeto `jakarta.jms.Connection` .

JMS 2.0 Os pacotes javax.jms definem as interfaces JMS e um provedor JMS implementa essas interfaces para um produto de sistema de mensagens específico. O IBM MQ classes for JMS é um provedor do JMS que implementa as interfaces do JMS para o IBM MQ.

JM 3.0 Os pacotes jakarta.jms definem as interfaces Jakarta Messaging e um provedor Jakarta Messaging implementa essas interfaces para um produto de sistema de mensagens específico. O IBM MQ classes for Jakarta Messaging é um provedor do Jakarta Messaging que implementa as interfaces do Jakarta Messaging para o IBM MQ.

As especificações JMS e Jakarta Messaging esperam que ConnectionFactory e objetos de Destino sejam objetos administrados. Um administrador cria e mantém objetos administrados em um repositório central e um aplicativo JMS ou Jakarta Messaging recupera esses objetos usando Java Naming Directory Interface (JNDI).

JMS 2.0 Para JMS 2.0, um administrador pode usar a IBM MQ JMS ferramenta de administração **JMSAdmin**ou IBM MQ Explorerpara criar e manter objetos administrados em um repositório central.

JM 3.0 Para Jakarta Messaging 3.0, não é possível administrar o JNDI usando IBM MQ Explorer A administração de JNDI é suportada pela variante Jakarta Messaging 3.0 de **JMSAdmin**, que é **JMS30Admin**

Como JMS e Jakarta Messaging compartilham muito em comum, referências adicionais a JMS neste tópico podem ser consideradas referentes a ambos. Quaisquer diferenças são destacadas conforme necessário.

O IBM MQ classes for JMS também fornece dois conjuntos de extensões para a API do JMS. O principal foco dessas extensões refere-se à criação e configuração de connection factories e destinos dinamicamente no tempo de execução, mas as extensões também fornecem a função que não está diretamente relacionada ao sistema de mensagens, como a função para determinação de problema.

As extensões do IBM MQ JMS

IBM MQ classes for JMS contém extensões que são implementadas em objetos como MQConnectionFactory, MQQueue e MQTopic. Esses objetos possuem propriedades e métodos que são específicos para IBM MQ. Os objetos podem ser objetos administrados ou um aplicativo pode criar os objetos dinamicamente no tempo de execução. Essas extensões são denominadas extensões IBM MQ JMS .

As extensões do IBM JMS

IBM MQ classes for JMS também fornece um conjunto mais genérico de extensões para a API JMS , que não são específicas do IBM MQ como o sistema de mensagens ou do Java como a linguagem de programação utilizada Essas extensões são denominadas extensões do IBM JMS e têm os objetivos gerais a seguir:

- Para fornecer um nível maior de consistência entre os provedores do IBM JMS
- Para facilitar a gravação de um aplicativo de ponte entre dois sistemas de mensagens do IBM .
- Para facilitar a porta de um aplicativo de um provedor do IBM JMS para outro.

As extensões fornecem a função que é semelhante àquela fornecida em IBM MQ Message Service Client (XMS) for C/C++ e IBM MQ Message Service Client (XMS) for .NET.

Conceitos relacionados

Interfaces de linguagem do IBM MQ Java

Tarefas relacionadas

“Gravando aplicativos IBM MQ classes for JMS/Jakarta Messaging” na página 142
Após uma breve introdução ao modelo do JMS , esta seção fornece orientação detalhada sobre como gravar aplicativos IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

JM 3.0 Por que devo usar o IBM MQ classes for Jakarta Messaging?

Ao usar o IBM MQ classes for Jakarta Messaging haverá um número de vantagens incluindo ser possível reutilizar quaisquer habilidades existentes do Jakarta Messaging em sua organização e os aplicativos serem mais independentes do provedor Jakarta Messaging e a configuração subjacente do IBM MQ.

Resumo das vantagens de usar o IBM MQ classes for Jakarta Messaging

Ao usar o IBM MQ classes for Jakarta Messaging permite que você reutilize habilidades existentes do Jakarta Messaging e forneça independência do aplicativo.

- É possível reutilizar habilidades do Jakarta Messaging.

  IBM MQ classes for Jakarta Messaging é um provedor Jakarta Messaging que implementa a interface Jakarta Messaging para IBM MQ como o sistema de mensagens. Se sua organização for nova no IBM MQ, mas já tiver qualificações de desenvolvimento de aplicativo Jakarta Messaging (ou JMS), você poderá achar mais fácil usar a API do Jakarta Messaging familiar para acessar recursos IBM MQ em vez de uma das outras APIs fornecidas com o IBM MQ.

- Jakarta Messaging é uma parte integral de Jakarta EE

  O Jakarta Messaging é a API natural a ser usada para o sistema de mensagens na plataforma Jakarta EE. Cada servidor de aplicativos compatível com Jakarta EE deve incluir um provedor Jakarta Messaging. É possível usar o Jakarta Messaging em aplicativos clientes, servlets, o Java Server Pages (JSPs), enterprise Java beans (EJBs) e beans acionados por mensagens (MDBs). Observe, em particular que os aplicativos Jakarta EE usam MDBs para processar mensagens de forma assíncrona e todas as mensagens são entregues aos MDBs como mensagens do Jakarta Messaging.
- Connection factories e destinos podem ser armazenados como objetos administrados do Jakarta Messaging em um repositório central em vez de ser codificado permanentemente em um aplicativo.

  Um administrador pode criar e manter objetos administrados Jakarta Messaging em um repositório central, e os aplicativos IBM MQ classes for Jakarta Messaging podem recuperar esses objetos usando o Java Naming Directory Interface (JNDI). As fábricas de conexão Jakarta Messaging e destinos encapsulam informações específicas de IBM MQ, como nomes do gerenciador de filas, nomes de canais, opções de conexão, nomes de filas e nomes de tópicos. Se connection factories e destinos forem armazenados como objetos administrados, essas informações não serão codificadas permanentemente em um aplicativo. Este acordo, portanto, fornece ao aplicativo um grau de independência da configuração subjacente do IBM MQ.
- Jakarta Messaging é uma API padrão de mercado que pode fornecer a portabilidade do aplicativo.

  Um aplicativo Jakarta Messaging pode usar JNDI para recuperar connection factories e destinos que são armazenados como objetos administrados e usar apenas as interfaces que são definidas no pacote `jakarta.jms` (Jakarta Messaging 3.0) para executar operações do sistema de mensagens. O aplicativo então é completamente independente de qualquer provedor Jakarta Messaging, como IBM MQ classes for Jakarta Messaging e pode ser transportado a partir de um provedor Jakarta Messaging para outro sem qualquer mudança ao aplicativo.

  Se JNDI não estiver disponível em um determinado ambiente de aplicativos, um aplicativo IBM MQ classes for Jakarta Messaging poderá usar extensões para a API Jakarta Messaging para criar e configurar connection factories e destinos dinamicamente no tempo de execução. O aplicativo então é totalmente autocontido, mas é ligado ao IBM MQ classes for Jakarta Messaging como o provedor Jakarta Messaging.
- Os aplicativos de ligação podem ser mais fácil de gravar usando Jakarta Messaging.

  Um aplicativo de ligação é um aplicativo que recebe as mensagens a partir de um sistema de mensagens e as envia para outro sistema de mensagens. A gravação de um aplicativo de ligação pode ser complicada usando as APIs específicas do produto e formatos de mensagens. Em vez disso, é possível gravar um aplicativo de ligação usando dois provedores Jakarta Messaging, um para cada sistema de mensagens. O aplicativo então usa somente uma API, a API do Jakarta Messaging e processa somente mensagens do Jakarta Messaging.

Ambientes implementáveis

Para fornecer integração com um servidor de aplicativo Jakarta EE, as normas do Jakarta EE requerem que provedores de sistema de mensagens forneçam um adaptador de recursos. Seguindo a especificação Jakarta Connectors Architecture , IBM MQ fornece um adaptador de recursos que usa Jakarta Messaging para fornecer funções do sistema de mensagens em qualquer ambiente certificado do Jakarta EE . Para mais informações, consulte “Liberty e o adaptador de recursos do IBM MQ” na página 450.

Nota: O WebSphere Application Server traditional não suporta atualmente Jakarta EE.

Fora do ambiente do Jakarta EE, os arquivos OSGi e JAR são fornecidos, tornando-o mais fácil para obter apenas o IBM MQ classes for Jakarta Messaging. Esses arquivos JAR são mais prontamente implementáveis independente ou dentro de estruturas de gerenciamento de software, como Maven. Para obter mais informações, consulte “Obtendo o IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging separadamente” na página 128.

Conceitos relacionados

Classes do IBM MQ para Jakarta Messaging: uma visão geral

“Acessando IBM MQ de Java -Opção de API” na página 87
IBM MQ fornece três interfaces de linguagem Java .

JMS 2.0 Por que devo usar o IBM MQ classes for JMS?

Ao usar o IBM MQ classes for JMS haverá um número de vantagens incluindo ser possível reutilizar quaisquer habilidades existentes do JMS em sua organização e os aplicativos serem mais independentes do provedor JMS e a configuração subjacente do IBM MQ.

Resumo das vantagens de usar o IBM MQ classes for JMS

Ao usar o IBM MQ classes for JMS permite que você reutilize habilidades existentes do JMS e forneça independência do aplicativo.

Nota: O JMS 2.0 foi substituído por Jakarta Messaging O IBM MQ classes for JMS continua a suportar o padrão JMS 2.0 , mas aprimoramentos futuros no sistema de mensagens do Java surgirão apenas no Jakarta Messaging, portanto, no IBM MQ classes for Jakarta Messaging. IBM MQ classes for JMS são recomendados apenas para manter e estender aplicativos JMS 2.0 existentes. IBM MQ classes for Jakarta Messaging deve ser a tecnologia preferencial para novo desenvolvimento.

- É possível reutilizar habilidades do JMS.

  IBM MQ classes for JMS é um provedor JMS que implementa a interface JMS para IBM MQ como o sistema de mensagens. Se sua organização for nova para IBM MQ, mas já tiver habilidades de desenvolvimento do aplicativo JMS, você poderá achar mais fácil usar a API familiar do JMS para acessar os recursos do IBM MQ em vez de uma das outras APIs fornecida com o IBM MQ.
- JMS é uma parte integral do Java Platform, Enterprise Edition (Java EE).

  O JMS é a API natural a ser usada para o sistema de mensagens na plataforma Java EE. Cada servidor de aplicativos compatível com Java EE deve incluir um provedor JMS. É possível usar o JMS em aplicativos clientes, servlets, o Java Server Pages (JSPs), enterprise Java beans (EJBs) e beans acionados por mensagens (MDBs). Observe, em particular que os aplicativos Java EE usam MDBs para processar mensagens de forma assíncrona e todas as mensagens são entregues aos MDBs como mensagens do JMS.
- Connection factories e destinos podem ser armazenados como objetos administrados do JMS em um repositório central em vez de ser codificado permanentemente em um aplicativo.

  Um administrador pode criar e manter objetos administrados JMS em um repositório central, e os aplicativos IBM MQ classes for JMS podem recuperar esses objetos usando o Java Naming Directory Interface (JNDI). As fábricas de conexão JMS e destinos encapsulam informações específicas de IBM MQ, como nomes do gerenciador de filas, nomes de canais, opções de conexão, nomes de filas e nomes de tópicos. Se connection factories e destinos forem armazenados como objetos administrados, essas informações não serão codificadas permanentemente em um aplicativo. Este acordo, portanto, fornece ao aplicativo um grau de independência da configuração subjacente do IBM MQ.
- JMS é uma API padrão de mercado que pode fornecer a portabilidade do aplicativo.

  Um aplicativo JMS pode usar JNDI para recuperar connection factories e destinos que são armazenados como objetos administrados e usar apenas as interfaces definidas no pacote `javax.jms` para executar operações de sistema de mensagens. O aplicativo então é completamente independente de qualquer provedor JMS, como IBM MQ classes for JMS e pode ser transportado a partir de um provedor JMS para outro sem qualquer mudança ao aplicativo.

  Se o JNDI não estiver disponível em um ambiente de aplicativos específico, um aplicativo IBM MQ classes for JMS poderá usar extensões para a API do JMS para criar e configurar connection factories e destinos dinamicamente no tempo de execução. O aplicativo então é totalmente autocontido, mas é ligado ao IBM MQ classes for JMS como o provedor JMS.
- Os aplicativos de ligação podem ser mais fácil de gravar usando JMS.

  Um aplicativo de ligação é um aplicativo que recebe as mensagens a partir de um sistema de mensagens e as envia para outro sistema de mensagens. A gravação de um aplicativo de ligação

pode ser complicada usando as APIs específicas do produto e formatos de mensagens. Em vez disso, é possível gravar um aplicativo de ligação usando dois provedores JMS, um para cada sistema de mensagens. O aplicativo então usa somente uma API, a API do JMS e processa somente mensagens do JMS.

Ambientes implementáveis

Para fornecer integração com um servidor de aplicativo Java EE, as normas do Java EE requerem que provedores de sistema de mensagens forneçam um adaptador de recursos. Após a especificação do Java EE Connector Architecture (JCA), o IBM MQ fornece um adaptador de recursos que usa o JMS para fornecer funções do sistema de mensagens dentro de qualquer ambiente certificado do Java EE.

Enquanto foi possível usar o IBM MQ classes for Java dentro do Java EE, esta API não será projetada ou otimizada para esta finalidade. Para obter mais informações sobre IBM MQ classes for Java considerações dentro de Java EE, consulte “Executando aplicativos IBM MQ classes for Java no Java EE” na página 355.

Fora do ambiente do Java EE, os arquivos OSGi e JAR são fornecidos, tornando-o mais fácil para obter apenas o IBM MQ classes for JMS. Esses arquivos JAR são mais prontamente implementáveis independente ou dentro de estruturas de gerenciamento de software, como Maven. Para obter mais informações, consulte “Obtendo o IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging separadamente” na página 128.

Conceitos relacionados

Classes do IBM MQ para Jakarta Messaging: uma visão geral

“Por que devo usar o IBM MQ classes for Jakarta Messaging?” na página 84
Ao usar o IBM MQ classes for Jakarta Messaging haverá um número de vantagens incluindo ser possível reutilizar quaisquer habilidades existentes do Jakarta Messaging em sua organização e os aplicativos serem mais independentes do provedor Jakarta Messaging e a configuração subjacente do IBM MQ.

“Acessando IBM MQ de Java -Opção de API” na página 87
IBM MQ fornece três interfaces de linguagem Java .

Acessando IBM MQ de Java -Opção de API

IBM MQ fornece três interfaces de linguagem Java .

- IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

O IBM MQ classes for Jakarta Messaging permite que aplicativos gravados usando as APIs Jakarta Messaging 3.0 utilizem IBM MQ como um provedor de sistemas de mensagens.

Jakarta Messaging é a direção estratégica para o sistema de mensagens em aplicativos Java

Jakarta Messaging 3.0 é funcionalmente equivalente a JMS 2.0, portanto, para obter mais informações, consulte “Usando IBM MQ classes for JMS/Jakarta Messaging” na página 83.

IBM MQ classes for JMS

O IBM MQ classes for JMS permite que aplicativos gravados usando as APIs JMS 2.0 utilizem IBM MQ como um provedor de sistemas de mensagens.

Como Jakarta Messaging substitui JMS, IBM MQ classes for JMS é recomendado para uso em aplicativos existentes ou em ambientes (por exemplo, WebSphere Application Server) que não suportam Jakarta Messaging.

Não é suportado usar IBM MQ classes for Jakarta Messaging e IBM MQ classes for JMS no mesmo aplicativo.

Para obter mais informações, consulte “Usando IBM MQ classes for JMS/Jakarta Messaging” na página 83.

IBM MQ classes for Java

Stabilized A outra API que os aplicativos Java podem usar para acessar IBM MQ recursos é IBM MQ classes for Java, que fornece uma API orientada a IBM MQpara programas usarem IBM MQ como um provedor de sistemas de mensagens. No entanto, o IBM MQ classes for Java é estabilizado funcionalmente no nível enviado em IBM MQ 8.0 Para obter informações adicionais, consulte “Por que devo usar o IBM MQ classes for Java?” na página 354. Embora os aplicativos existentes que usam o IBM MQ classes for Java continuem sendo totalmente suportados, novos aplicativos devem usar o IBM MQ classes for Jakarta Messaging.

Recursos comuns do IBM MQ classes for JMS e do IBM MQ classes for Jakarta Messaging

O IBM MQ classes for JMS e o IBM MQ classes for Jakarta Messaging fornecem acesso aos recursos do sistema de mensagens ponto a ponto e de publicação / assinatura do IBM MQ. Assim como o envio de mensagens do JMS que fornecem suporte para o modelo do sistema de mensagens padrão do JMS, os aplicativos também podem enviar e receber mensagens sem cabeçalhos adicionais e, portanto, podem interoperar com outros aplicativos IBM MQ, por exemplo, aplicativos C MQI. O controle total das cargas úteis de mensagens MQMD e MQ está disponível.

Outros recursos do IBM MQ, como o fluxo de mensagens, put assíncrono e mensagens de relatório também estão disponíveis.

O uso das classes auxiliares PCF fornecidas, mensagens de administração PCF do IBM MQ pode ser enviado e recebido através da API do JMS e pode ser usada para administrar gerenciadores de filas.

Os recursos que foram recentemente incluídos no IBM MQ, como consumo assíncrono e reconexão automática, não estão disponíveis no IBM MQ classes for Java, mas estão disponíveis no IBM MQ classes for JMS e no IBM MQ classes for Jakarta Messaging

Solicitando aprimoramentos

Se você precisar de acesso aos recursos do IBM MQ que não estão disponíveis por IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, será possível criar uma ideia.

IBM pode então informar se a implementação é possível na implementação IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging ou se há uma melhor prática que pode ser seguida.

Para os recursos adicionais do sistema de mensagens, como IBM é um contribuidor para o padrão aberto, esses recursos podem ser gerados como parte do processo de JCP. Isso se aplicaria apenas ao Jakarta Messaging.

Informações relacionadas

Bem-vindo ao IBM Ideas Portal

Processo de Revisão de Especificação JMS Java

Usando JMS para enviar mensagens PCF

JM 3.0 Pré-requisitos para o IBM MQ classes for Jakarta Messaging

Este tópico informa o que você precisa saber antes de usar o IBM MQ classes for Jakarta Messaging. Para desenvolver e executar aplicativos IBM MQ classes for Jakarta Messaging, você precisará de determinados componentes de software como pré-requisitos.

Para obter informações sobre os pré-requisitos para o IBM MQ classes for Jakarta Messaging, veja Requisitos do sistema para IBM MQ.

Para desenvolver aplicativos do IBM MQ classes for Jakarta Messaging, é necessário um Kit de desenvolvimento de software (SDK) do Java SE. Para obter detalhes dos JDKs suportados por seu sistema operacional, veja Requisitos do sistema para IBM MQ.

Para executar aplicativos do IBM MQ classes for Jakarta Messaging, é necessário ter os componentes de software a seguir:

- Um gerenciador de filas do IBM MQ.
- Um Java runtime environment (JRE), para cada sistema no qual você executa os aplicativos.
- IBM i Para IBM i, Qshell, que á opção 30 do sistema operacional.
- z/OS Para z/OS, z/OS UNIX System Services (z/OS UNIX).

O provedor JSSE do IBM inclui um provedor de criptografia certificado por FIPS, portanto, pode ser configurado programaticamente para conformidade do FIPS 140-2 pronta para uso imediato. Portanto, a conformidade FIPS 140-2 pode ser suportada diretamente a partir do IBM MQ classes for Jakarta Messaging.

O provedor JSSE da Oracle pode ter um provedor criptográfico certificado por FIPS que esteja configurado para ele, mas isso não está pronto para uso imediato e não está disponível para configuração programática. Portanto, neste caso, o IBM MQ classes for Jakarta Messaging não pode ativar a conformidade FIPS 140-2 diretamente. Talvez seja possível ativar manualmente tal conformidade, mas o IBM não pode fornecer atualmente orientação sobre isso.

É possível usar Internet Protocol versão 6 (IPv6) endereços em seus aplicativos IBM MQ classes for Jakarta Messaging se IPv6 endereços forem suportados por sua Java máquina virtual (JVM) e a implementação TCP/IP em seu sistema operacional. A ferramenta de administração IBM MQ Jakarta Messaging , **JMS30Admin**, também aceita endereços IPv6 . Para obter mais informações sobre essa ferramenta, consulte Configurando JMS e Jakarta Messaging objetos que usam as ferramentas de administração

A ferramenta de administração do IBM MQ JMS e o IBM MQ Explorer usam o Java Naming Directory Interface (JNDI) para acessar um serviço de diretório, que armazena objetos administrados. Os aplicativos IBM MQ classes for Jakarta Messaging também podem usar o JNDI para recuperar objetos administrados de um serviço de diretório.

Nota: Para Jakarta Messaging 3.0, não é possível administrar o JNDI usando IBM MQ Explorer A administração de JNDI é suportada pela variante Jakarta Messaging 3.0 de **JMSAdmin**, que é **JMS30Admin**

Um provedor de serviços é o código que fornece acesso a um serviço de diretório mapeando chamadas do JNDI para o serviço de diretório. Um provedor do serviço de sistema de arquivos nos arquivos `fscontext.jar` e `providerutil.jar` é fornecido com o IBM MQ classes for Jakarta Messaging. O provedor de serviços do sistema de arquivos fornece acesso a um serviço de diretório baseado no sistema de arquivos local.

Se você pretende usar um serviço de diretório baseado em um servidor LDAP, deve-se instalar e configurar um servidor LDAP ou ter acesso a um servidor LDAP existente. Em particular, deve-se configurar o servidor LDAP para armazenar os objetos do Java. Para obter informações sobre como instalar e configurar seu servidor LDAP, consulte a documentação que é fornecida com o servidor.

JMS 2.0 Pré-requisitos para o IBM MQ classes for JMS

Este tópico informa o que você precisa saber antes de usar o IBM MQ classes for JMS. Para desenvolver e executar aplicativos IBM MQ classes for JMS, você precisará de determinados componentes de software como pré-requisitos.

Para obter informações sobre os pré-requisitos para o IBM MQ classes for JMS, veja Requisitos do sistema para IBM MQ.

Para desenvolver aplicativos do IBM MQ classes for JMS, é necessário um Kit de desenvolvimento de software (SDK) do Java SE. Para obter detalhes dos JDKs suportados por seu sistema operacional, veja Requisitos do sistema para IBM MQ.

Para executar aplicativos do IBM MQ classes for JMS, é necessário ter os componentes de software a seguir:

- Um gerenciador de filas do IBM MQ.
- Um Java runtime environment (JRE), para cada sistema no qual você executa os aplicativos.
- IBM i Para IBM i, Qshell, que á opção 30 do sistema operacional.
- z/OS Para z/OS, z/OS UNIX System Services (z/OS UNIX).

O provedor JSSE do IBM inclui um provedor de criptografia certificado por FIPS, portanto, pode ser configurado programaticamente para conformidade do FIPS 140-2 pronta para uso imediato. Portanto, a conformidade do FIPS 140-2 pode ser suportada diretamente do IBM MQ classes for Java e IBM MQ classes for JMS.

O provedor JSSE da Oracle pode ter um provedor criptográfico certificado por FIPS que esteja configurado para ele, mas isso não está pronto para uso imediato e não está disponível para configuração programática. Portanto, neste caso, IBM MQ classes for Java e IBM MQ classes for JMS não podem ativar a conformidade do FIPS 140-2 diretamente. Talvez seja possível ativar manualmente tal conformidade, mas o IBM não pode fornecer atualmente orientação sobre isso.

É possível usar Internet Protocol versão 6 (IPv6) endereços em seus aplicativos IBM MQ classes for JMS se IPv6 endereços forem suportados por sua Java máquina virtual (JVM) e a implementação TCP/IP em seu sistema operacional. A ferramenta de administração do IBM MQ JMS (consulte Configurando objetos do JMS usando a ferramenta de administração) também aceita endereços IPv6.

A ferramenta de administração do IBM MQ JMS e o IBM MQ Explorer usam o Java Naming Directory Interface (JNDI) para acessar um serviço de diretório, que armazena objetos administrados. Os aplicativos IBM MQ classes for JMS também podem usar o JNDI para recuperar objetos administrados de um serviço de diretório. Um provedor de serviços é o código que fornece acesso a um serviço de diretório mapeando chamadas do JNDI para o serviço de diretório. Um provedor de serviços de sistema de arquivos nos arquivos `fscontext.jar` e `providerutil.jar` é fornecido com IBM MQ classes for JMS. O provedor de serviços do sistema de arquivos fornece acesso a um serviço de diretório baseado no sistema de arquivos local.

Se você pretende usar um serviço de diretório baseado em um servidor LDAP, deve-se instalar e configurar um servidor LDAP ou ter acesso a um servidor LDAP existente. Em particular, deve-se configurar o servidor LDAP para armazenar os objetos do Java. Para obter informações sobre como instalar e configurar seu servidor LDAP, consulte a documentação que é fornecida com o servidor.

Instalando e configurando IBM MQ classes for JMS/Jakarta Messaging

Esta seção descreve os diretórios e arquivos que são criados quando você instala IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaginge informa como configurar IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging após a instalação.

Conceitos relacionados

"Usando o adaptador de recursos do IBM MQ" na página 444
O adaptador de recursos permite que aplicativos que estão em execução em um servidor de aplicativos acesse recursos do IBM MQ. Ele suporta a comunicação de entrada e de saída.

O que é instalado para IBM MQ classes for JMS

Vários arquivos e diretórios são criados ao instalar o IBM MQ classes for JMS. No Windows, algumas configurações são executadas durante a instalação, configurando automaticamente variáveis de ambiente. Em outras plataformas e em determinados ambientes de Windows, deve-se configurar as variáveis de ambiente antes de poder executar aplicativos IBM MQ classes for JMS.

Para a maioria dos sistemas operacionais, os IBM MQ classes for JMS são instalados como um componente opcional ao instalar o IBM MQ.

Para obter informações adicionais sobre como instalar o IBM MQ, consulte:

Multi Instalando o IBM MQ

z/OS Instalando o IBM MQ for z/OS

Importante: Além dos arquivos JAR realocáveis descritos em "IBM MQ classes for JMS/Jakarta Messaging arquivos JAR relocalizáveis" na página 92, não é suportada a cópia dos arquivos JAR ou das bibliotecas nativas do IBM MQ classes for JMS em outras máquinas ou em um local diferente de uma máquina na qual o IBM MQ classes for JMS foi instalado.

Diretórios de instalação

O Tabela 5 na página 91 mostra onde os arquivos IBM MQ classes for JMS são instalados em cada plataforma.

Tabela 5. Diretórios de instalação do IBM MQ classes for JMS

Plataforma	Diretório
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*`/java`
Windows Windows	*MQ_INSTALLATION_PATH*`\java`
IBM i IBM i	`/QIBM/ProdData/mqm/java`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java`

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

O diretório de instalação inclui o seguinte conteúdo:

- Os arquivos JAR IBM MQ classes for JMS, incluindo os arquivos JAR realocáveis, que estão no diretório *MQ_INSTALLATION_PATH*`\java\lib`.
- As bibliotecas nativas do IBM MQ, que são usadas por aplicativos que usam a interface nativa do Java.

  As bibliotecas nativas de 32 bits são instaladas no diretório *MQ_INSTALLATION_PATH*`\java\lib` e as bibliotecas nativas de 64 bits podem ser encontradas no diretório *MQ_INSTALLATION_PATH*\java\lib64.

  Para obter informações adicionais sobre as bibliotecas nativas do IBM MQ, consulte "Configurando as bibliotecas do Java Native Interface (JNI)" na página 97.
- Os scripts adicionais que são descritos em "Scripts fornecidos com IBM MQ classes for JMS/Jakarta Messaging" na página 125. Esses scripts estão no diretório *MQ_INSTALLATION_PATH*`\java\bin`.
- As especificações do IBM MQ classes for JMS API. A ferramenta Javadoc foi usada para gerar as páginas HTML que contêm as especificações da API.

  As páginas HTML estão no diretório *MQ_INSTALLATION_PATH*`\java\doc\WMQJMSClasses`:

  – ALW No AIX, Linux, and Windows, esse subdiretório contém as páginas HTML individuais.

  – IBM i No IBM i, as páginas HTML estão em um arquivo chamado `wmqjms_javadoc.jar`.

  – z/OS No z/OS, as páginas HTML estão em um arquivo chamado `wmqjms_javadoc.jar`.
- Suporte para OGSi. Os pacotes configuráveis OSGi são instalados no diretório `java\lib\OSGi` e descritos em "Suporte para OSGi com o IBM MQ classes for JMS" na página 126.
- O adaptador de recursos IBM MQ , que pode ser implementado em qualquer servidor de aplicativos compatível com Java Platform, Enterprise Edition 7 (Java EE 7) ou Jakarta EE .

  O adaptador de recursos IBM MQ está no diretório *MQ_INSTALLATION_PATH*`\java\lib\jca`; para obter mais informações, consulte "Usando o adaptador de recursos do IBM MQ" na página 444

- Windows No Windows, os símbolos que podem ser usados para depuração são instalados no diretório *MQ_INSTALLATION_PATH*\java\lib\symbols.

O diretório de instalação também inclui alguns arquivos que pertencem a outros componentes do IBM MQ.

Aplicativos de amostra

JMS 2.0 Alguns aplicativos de amostra são fornecidos com o IBM MQ classes for JMS. O Tabela 6 na página 92 mostra onde os aplicativos de amostra são instalados em cada plataforma.

JM 3.0 Para IBM MQ classes for Jakarta Messaging, novas amostras estão sendo preparadas..

JMS 2.0

Tabela 6. Diretórios de amostras para IBM MQ classes for JMS	
Plataforma	**Diretório**
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/samp/jms
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms
IBM i IBM i	/QIBM/ProdData/mqm/java/samples/jms
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

Nesta tabela, *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Após a instalação, pode ser necessário executar algumas tarefas de configuração para compilar e executar aplicativos.

"Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging" na página 95 descreve o caminho de classe necessário para executar aplicativos de amostra do IBM MQ classes for JMS . Este tópico também descreve arquivos JAR adicionais que precisam ser referidos em circunstâncias especiais e as variáveis de ambiente que devem ser configuradas para executarem os scripts fornecidos com o IBM MQ classes for JMS.

Para controlar propriedades, como o rastreio e criação de log de um aplicativo, será necessário fornecer um arquivo de propriedades de configuração. O arquivo de propriedades de configuração do IBM MQ classes for JMS está descrito em "O arquivo de configuração IBM MQ classes for JMS/Jakarta Messaging" na página 100.

Conceitos relacionados

Problemas na implementação do adaptador de recursos

Tarefas relacionadas

"Usando os aplicativos de amostra IBM MQ classes for JMS" na página 121
Os aplicativos de amostra do IBM MQ classes for JMS fornecem uma visão geral dos recursos comuns da API do JMS. É possível usá-los para verificar a sua instalação e o servidor de sistema de mensagens configurado e para ajudar a construir os seus próprios aplicativos.

IBM MQ classes for JMS/Jakarta Messaging arquivos JAR relocalizáveis
Os arquivos JAR relocalizáveis podem ser movidos para sistemas que precisam executar IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

Importante:

- Além dos arquivos JAR realocáveis descritos em Arquivos JAR Relocáveis, copiar os arquivos JAR IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging ou bibliotecas nativas para outras

máquinas ou para um local diferente em uma máquina em que o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging foram instalados não é suportado.

- Não inclua os arquivos JAR realocáveis dentro de aplicativos implementados em servidores de aplicativos Java EE, como o WebSphere Application Server ou o WebSphere Liberty. Nesses ambientes, o adaptador de recursos do IBM MQ deve ser implementado e usado como alternativa. Observe que o WebSphere Application Server integra o adaptador de recursos do IBM MQ, portanto, não há necessidade de implementá-lo manualmente nesse ambiente.
- Para evitar conflitos do carregador de classes, não é recomendado empacotar os arquivos JAR realocáveis dentro de vários aplicativos dentro do mesmo tempo de execução de Java. Neste cenário, disponibilize os arquivos JAR relocalizáveis do IBM MQ no caminho de classe do tempo de execução do Java
- Se você estiver empacotando os arquivos JAR realocáveis dentro de seus aplicativos, assegure-se de incluir todos os arquivos JAR de pré-requisito, conforme descrito em Arquivos JAR realocáveis. Você também deve assegurar que tenha procedimentos apropriados para atualizar os arquivos JAR empacotados como parte da manutenção do aplicativo, para assegurar que o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging ainda sejam problemas atuais e conhecidos sejam remediados.

Arquivos JAR relocalizáveis

Em uma empresa, os arquivos a seguir podem ser movidos para sistemas que precisam executar IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging:

- `bcpkix-jdk15to18.jar` "4" na página 93
- V 9.4.0 `bcpkix-jdk18on.jar` "3" na página 93
- `bcprov-jdk15to18.jar` "4" na página 93
- V 9.4.0 `bcprov-jdk18on.jar` "3" na página 93
- `bcutil-jdk15to18.jar` "4" na página 93
- V 9.4.0 `bcutil-jdk18on.jar` "3" na página 93
- JMS 2.0 `com.ibm.mq.allclient.jar` "1" na página 93
- JM 3.0 `com.ibm.mq.jakarta.client.jar` "2" na página 93
- `fscontext.jar`
- `jakarta.jms-api.jar`
- `jms.jar`
- `org.json.jar`
- `providerutil.jar`

Notas:

1. JMS 2.0 e JMS 1.1
2. Jakarta Messaging 3.0
3. De IBM MQ 9.4.0
4. AntesIBM MQ 9.4.0

JMS Arquivos JAR

`jms.jar` contém as interfaces JMS 1.1 e JMS 2.0 -elas são denominadas `javax.jms.*`

JM 3.0 `jakarta.jms-api.jar` contém as interfaces Jakarta Messaging 3.0 -elas são denominadas `jakarta.jms.*`

fscontext.jar e providerutil.jar

Os arquivos `fscontext.jar` e `providerutil.jar` são necessários se o seu aplicativo executa consultas de JNDI usando um contexto de sistema de arquivos.

O provedor de segurança Bouncy Castle e o CMS suportam arquivos JAR

O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS são necessários. Para mais informações, consulte o Suporte para JREs não-IBM com AMS.

V 9.4.0 São necessários os seguintes arquivos JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

O arquivo `org.json.jar` é necessário se o seu aplicativo IBM MQ classes for JMS usa um CCDT em formato JSON.

com.ibm.mq.allclient.jar e com.ibm.mq.jakarta.client.jar

Os arquivos `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar` contêm o IBM MQ classes for JMS, o IBM MQ classes for Jakarta Messaging, o IBM MQ classes for Javae as classes PCF e de cabeçalhos. Se você mover esse arquivo JAR para um novo local, certifique-se de executar etapas para manter esse novo local mantido com novos Fix Packs do IBM MQ . Além disso, certifique-se de que o uso dos arquivos seja conhecido para o Suporte IBM se você estiver obtendo uma correção temporária

Para determinar a versão dos arquivos `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar`, use o comando a seguir:

JM 3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

O exemplo a seguir mostra uma saída de amostra desse comando:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging
Antes de poder compilar e executar aplicativos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , a configuração para sua variável de ambiente **CLASSPATH** deve incluir o archive (JAR) IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging Java . Dependendo de seus requisitos, pode ser necessário incluir outros arquivos JAR no caminho de classe. Para executar os scripts fornecidos com IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, outras variáveis de ambiente devem ser configuradas.

Antes de começar

JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando IBM MQ classes para JMS/Jakarta Messaging.

Importante: A configuração da Java opção *-Xbootclasspath* para incluir o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging não é suportada

Sobre esta tarefa

Para compilar e executar aplicativos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , use a configuração **CLASSPATH** para sua plataforma e versão do sistema de mensagens do Java , conforme mostrado nas tabelas a seguir. Como alternativa, é possível especificar o caminho de classe no comando **`java`** em vez de usar a variável de ambiente.

JMS 2.0 Para IBM MQ classes for JMS, a configuração inclui o diretório de amostras para que seja possível compilar e executar os aplicativos de amostra IBM MQ classes for JMS .

JM 3.0 Para IBM MQ classes for Jakarta Messaging, novas amostras estão sendo preparadas..

JM 3.0

*Tabela 7. **CLASSPATH** configurações para Jakarta Messaging 3.0 para compilar e executar aplicativos IBM MQ classes for Jakarta Messaging*

Plataforma	CLASSPATH como definir
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

*Tabela 8. **CLASSPATH** configurações para JMS 2.0 compilar e executar aplicativos IBM MQ classes for JMS , incluindo os aplicativos de amostra*	
Plataforma	**Configuração de CLASSPATH**
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

Nessas tabelas, *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual IBM MQ está instalado.

O manifesto do arquivo JAR `com.ibm.mq.jakarta.client.jar` ou `com.ibm.mq.allclient.jar` contém referências para a maioria dos outros arquivos JAR requeridos por aplicativos IBM MQ classes for JMS e, portanto, não é necessário incluir esses arquivos JAR em seu caminho de classe Esses arquivos JAR incluem aqueles requeridos por aplicativos que usam o Java Naming Directory Interface (JNDI) para recuperar objetos administrados de um serviço de diretório e por aplicativos que usam o Java Transaction API (JTA).

No entanto, deve-se incluir arquivos JAR adicionais em seu caminho de classe nas circunstâncias a seguir:

- Se estiver usando classes de saída de canal que implementam as interfaces de saída de canal definidas no pacote `com.ibm.mq` em vez de as definidas no pacote `com.ibm.mq.exits`, você deverá incluir o arquivo JAR IBM MQ classes for Java, `com.ibm.mq.jar`, no caminho de classe.
- Se seu aplicativo usa a JNDI para recuperar objetos administrados de um serviço de diretório, também deve-se incluir os arquivos JAR a seguir em seu caminho de classe:
  - `fscontext.jar`
  - `providerutil.jar`
- Se o aplicativo usar o JTA, você também deverá incluir `jta.jar` no caminho de classe.

Nota: Esses arquivos JAR adicionais são necessários somente para compilar seus aplicativos, não para executá-los.

Os scripts fornecidos com IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging usam as seguintes variáveis de ambiente:

MQ_JAVA_DATA_PATH
Essa variável de ambiente especifica o diretório para log e saída de rastreio.

MQ_JAVA_INSTALL_PATH
Essa variável de ambiente especifica o diretório no qual o IBM MQ classes for JMS está instalado.

MQ_JAVA_LIB_PATH
Essa variável de ambiente especifica o diretório no qual as bibliotecas IBM MQ classes for JMS são armazenadas, conforme mostrado nas tabelas anteriores.

Procedimento

- Windows

  No Windows, após a instalação do IBM MQ, execute o comando **setmqenv**

  Se você não executar este comando primeiro, a mensagem de erro a seguir poderá aparecer quando você estiver emitindo um comando **dspmqver** :

  ```
  AMQ8351: o ambiente IBM MQ Java não foi configurado
  corretamente ou o recurso IBM MQ JRE não foi instalado.
  ```

  Nota: Essa mensagem será esperada se você não tiver instalado o IBM MQ Java runtime environment (JRE) (veja Verificação de pré-requisito de recursos adicionais do Windows.

- Linux AIX

  Em sistemas AIX and Linux , configure você mesmo as variáveis de ambiente:

  JMS 2.0 Para JMS 2.0, use um dos seguintes scripts para configurar as variáveis de ambiente:

  – Se você estiver usando uma JVM de 32 bits, use o script `setjmsenv`
  – Se você estiver usando uma JVM de 64 bits em um sistema AIX ou Linux , use o script `setjmsenv64`

  JM 3.0 Para Jakarta Messaging 3.0, use um dos seguintes scripts para configurar as variáveis de ambiente:

  – Se você estiver usando uma JVM de 32 bits, use o script `setjms30env`
  – Se você estiver usando uma JVM de 64 bits, use o script `setjms30env64`

  Estes scripts estão no diretório *MQ_INSTALLATION_PATH*`/java/bin`, em que *MQ_INSTALLATION_PATH* representa o diretório de alto nível em que o IBM MQ está instalado.

  É possível usar esses scripts de várias maneiras. É possível usar o script como base para configurar as variáveis de ambiente necessárias, conforme mostrado na tabela ou incluí-las em `.profile` usando um editor de texto. Se você tiver uma configuração atípica, edite o conteúdo do script conforme necessário. Como alternativa, é possível executar o script em cada sessão a partir da qual os scripts de inicialização do JMS devem ser executados. Se você escolher essa opção, será necessário executar o script em cada janela shell que você iniciar, durante o processo de verificação JMS :

  – JMS 2.0 Para JMS 2.0, digite `. ./setjmsenv` ou `../setjmsenv64`..
  – JM 3.0 Para Jakarta Messaging 3.0, digite `. ./setjms30env` ou `. ./setjms30env64`..

  IBM i No IBM i, você deve configurar a variável de ambiente **QIBM_MULTI_THREADED** como Y. Em seguida, é possível executar aplicativos multiencadeados da mesma maneira que você executa aplicativos de encadeamento único. Para obter mais informações, consulte Configurando IBM MQ com Java e JMS.

Tarefas relacionadas

“Usando os aplicativos de amostra IBM MQ classes for JMS” na página 121
Os aplicativos de amostra do IBM MQ classes for JMS fornecem uma visão geral dos recursos comuns da API do JMS. É possível usá-los para verificar a sua instalação e o servidor de sistema de mensagens configurado e para ajudar a construir os seus próprios aplicativos.

Referências relacionadas

“Scripts fornecidos com IBM MQ classes for JMS/Jakarta Messaging” na página 125
Vários scripts são fornecidos para ajudar com tarefas comuns que precisam ser executadas ao usar IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Configurando as bibliotecas do Java Native Interface (JNI)
Aplicativos IBM MQ classes for JMS, que se conectam com um gerenciador de filas usando o transporte de ligações ou que se conectam a um gerenciador de filas usando o transporte de cliente e usam

programas de saída de canal gravados em linguagens diferentes de Java, precisam ser executados em um ambiente que permite acesso às bibliotecas Java Native Interface (JNI).

Antes de começar

Consulte Configurando o provedor de sistemas de mensagens IBM MQ com informações de bibliotecas nativas para obter mais informações sobre o uso do ambiente WebSphere Application Server.

Sobre esta tarefa

Para configurar esse ambiente, deve-se configurar o caminho da biblioteca do ambiente para que o Java Virtual Machine (JVM) possa carregar a biblioteca mqjbnd antes de iniciar o aplicativo IBM MQ classes for JMS.

O IBM MQ fornece duas bibliotecas Java Native Interface (JNI):

mqjbnd
Esta biblioteca é usada pelos aplicativos que se conectam a um gerenciador de filas que usam o transporte de ligações. Ela fornece a interface entre o IBM MQ classes for JMS e o gerenciador de filas. A biblioteca mqjbnd instalada com o IBM MQ 9.4 pode ser usada para se conectar a qualquer gerenciador de filas do IBM MQ 9.4 (ou anterior).

mqjexitstub02
A biblioteca mqjexitstub02 é carregada pelo IBM MQ classes for JMS quando um aplicativo se conecta a um gerenciador de filas usando o transporte de cliente e usa um programa de saída do canal gravado em um idioma diferente de Java.

Em algumas plataformas, o IBM MQ instala as versões de 32 bist e 64 bits destas bibliotecas JNI. O local das bibliotecas para cada plataforma é mostrado na Tabela 1.

Tabela 9. O local das bibliotecas do IBM MQ classes for JMS para cada plataforma

Plataforma	**Diretório que contém as bibliotecas do IBM MQ classes for JMS**
AIX AIX Linux Linux (plataformas POWER, x86-64 e zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\java\lib64 (bibliotecas de 64 bits)
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 31 bits e 64 bits)

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Nota: z/OS No z/OS, é possível usar uma Java Virtual Machine (JVM) de 31 bits ou 64 bits. Você não precisa especificar quais as bibliotecas JNI usar; IBM MQ classes for JMS pode determinar sozinho quais bibliotecas JNI carregar.

Procedimento

1. Configure a propriedade **`java.library.path`** da JVM, que pode ser feito de duas maneiras:

- Especificando o argumento JVM conforme mostrado no exemplo a seguir:

  ```
  -Djava.library.path=path_to_library_directory
  ```

  Linux Por exemplo, para uma JVM de 64 bits no Linux para uma instalação de local padrão, especifique:

  ```
  -Djava.library.path=/opt/mqm/java/lib64
  ```

- A configuração do ambiente do shell de modo que a JVM irá configurar seu próprio `java.library.path`. Esse caminho varia por plataforma e pelo local no qual você instalou o IBM MQ. Por exemplo, para uma JVM de 64 bits e um local de instalação padrão do IBM MQ, é possível usar as configurações a seguir:

  AIX
  ```
  export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
  ```

  Linux
  ```
  export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
  ```

  Windows
  ```
  set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
  ```

  Um exemplo da pilha de exceções que você vê quando o ambiente não foi configurado corretamente é o seguinte:

```
Causado por: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Falha ao carregar a biblioteca JNI nativa do WebSphere MQ: 'mqjbnd'.
    em com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    em com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    em java.security.AccessController.doPrivileged(AccessController.java:400)
    em com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    em com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    em sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    em
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    em
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a:58)
    em java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    em com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    em com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    em
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 mais
Causado por: java.lang.UnsatisfiedLinkError: mqjbnd (não localizado em java.library.path)
    em java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    em java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    em java.lang.System.loadLibrary(System.java:534)
    em com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 mais
```

2. Depois que o ambiente do 32 bits ou 64 bits tiver sido configurado, inicie o aplicativo IBM MQ classes for JMS usando o comando:

   ```
   java application-name
   ```

   em que *application-name* é o nome do aplicativo IBM MQ classes for JMS a ser executado.

   Uma exceção contendo o Código de razão 2495 (MQRC_MODULE_NOT_FOUND) do IBM MQ será lançada pelo IBM MQ classes for JMS se:

   - O aplicativo IBM MQ classes for JMS é executado em um Java runtime environment de 32 bits e um ambiente de 64 bits foi configurado para o IBM MQ classes for JMS, pois o Java runtime environment de 32 bits não é capaz de carregar a biblioteca nativa do Java de 64 bits.

- O aplicativo IBM MQ classes for JMS é executado em um Java runtime environment de 64 bits e um ambiente de 32 bits foi configurado para o IBM MQ classes for JMS, pois o Java runtime environment de 64 bits não é capaz de carregar a biblioteca nativa do Java de 32 bits.

O arquivo de configuração IBM MQ classes for JMS/Jakarta Messaging
Os arquivos de configuração IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging especificam as propriedades que são usadas para configurar IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Nota: As propriedades definidas no arquivo de configuração também podem ser configuradas como propriedades de sistema da JVM. Se uma propriedade for configurada no arquivo de configuração e como uma propriedade de sistema, a propriedade de sistema terá precedência. Portanto, se necessário, será possível substituir qualquer propriedade no arquivo de configuração especificando-a como uma propriedade de sistema no comando **java**.

O formato de um arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging é o formato de um arquivo de propriedades Java padrão Um arquivo de configuração de amostra chamado `jms.config` é fornecido no subdiretório `bin` do diretório de instalação do IBM MQ classes for JMS. Este arquivo documenta todas as propriedades suportadas e seus valores padrão.

É possível escolher o nome e local de um arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . Ao iniciar o seu aplicativo, use um comando **java** com o seguinte formato:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

No comando, *config_file_url* é um localizador uniforme de recursos (URL) que especifica o nome e o local do arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . URLs dos seguintes tipos são suportadas: http, arquivo, ftp e jar.

A seguir está um exemplo de um comando **java**:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

Esse comando identifica o arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging como o arquivo `D:\mydir\mjms.config` no sistema local Windows .

Quando um aplicativo é iniciado, o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging lê o conteúdo do arquivo de configuração e armazena as propriedades especificadas em um armazenamento de propriedade interno Se o comando **java** não identificar um arquivo de configuração ou se o arquivo de configuração não puder ser localizado, IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging usará os valores padrão para todas as propriedades.

Um arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging pode ser usado com qualquer um dos transportes suportados entre um aplicativo e um gerenciador de fila ou broker

Substituindo propriedades especificadas em um arquivo de configuração do IBM MQ MQI client

Um arquivo de configuração IBM MQ MQI client também pode especificar as propriedades que são utilizadas para configurar IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging No entanto, as propriedades especificadas em um arquivo de configuração do IBM MQ MQI client se aplicam apenas quando um aplicativo se conecta a um gerenciador de filas no modo cliente.

Se necessário, é possível substituir qualquer atributo em um arquivo de configuração IBM MQ MQI client , especificando-o como uma propriedade em um arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . Para substituir um atributo em um arquivo de configuração IBM MQ

MQI client , use uma entrada com o seguinte formato no arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging :

```
com.ibm.mq.cfg. stanza. propName = propValue
```

As variáveis na entrada possuem os seguintes significados:

stanza
O nome da sub-rotina no arquivo de configuração do IBM MQ MQI client que contém o atributo

propName
O nome do atributo, conforme especificado no arquivo de configuração do IBM MQ MQI client

propValue
O valor da propriedade que substitui o valor do atributo especificado no arquivo de configuração do IBM MQ MQI client

Como alternativa, é possível substituir um atributo em um arquivo de configuração do IBM MQ MQI client, especificando a propriedade como uma propriedade de sistema no comando **java**. Use o formato anterior para especificar a propriedade como uma propriedade de sistema.

Somente os seguintes atributos em um arquivo de configuração IBM MQ MQI client são relevantes para IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging. Se você especificar ou substituir outros atributos, isso não terá efeito. Especificamente, observe que o `ChannelDefinitionFile` e o `ChannelDefinitionDirectory` na sub-rotina CHANNELS do arquivo de configuração do cliente não são usados. Consulte "Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS" na página 289 para obter detalhes de como usar a CCDT com o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

Tabela 10. Qual sub-rotina do arquivo de configuração do cliente contém qual atributo

Sub-rotina	**Atributo**
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente	Put1DefaultAlwaysSync
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente	DefRecon
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente	ReconDelay
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente	PasswordProtection
Sub-rotina ClientExitPath do arquivo de configuração do cliente	Caminho Padrão das Saídas
Sub-rotina ClientExitPath do arquivo de configuração do cliente	ExitsDefaultPath64
Sub-rotina ClientExitPath do arquivo de configuração do cliente	JavaExitsClasspath
Sub-rotina JMQI do arquivo de configuração do cliente	useMQCSPauthentication
Sub-rotina MessageBuffer do arquivo de configuração do cliente	MaximumSize
Sub-rotina MessageBuffer do arquivo de configuração do cliente	PurgeTime
Sub-rotina MessageBuffer do arquivo de configuração do cliente	UpdatePercentage

Tabela 10. Qual sub-rotina do arquivo de configuração do cliente contém qual atributo (continuação)	
Sub-rotina	**Atributo**
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntRcvBufSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntSndBufSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	Connect_Timeout
Sub-rotina TCP do Arquivo de Configuração do Cliente	KeepAlive

Para obter detalhes adicionais sobre a configuração IBM MQ MQI client , consulte o arquivo de configuração IBM MQ MQI client , `mqclient.ini`

Usando o rastreio do Java Standard Environment Trace para configurar JMS
Use a sub-rotina Configurações de Rastreio do Ambiente Padrão Java para configurar o recurso de rastreio IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging .

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputName*
traceOutputName é o nome do diretório e do arquivo ao qual a saída de rastreio é enviada.

Por padrão, as informações de rastreio são gravadas em um arquivo de rastreio no diretório atualmente em funcionamento do aplicativo. O nome do arquivo de rastreio depende do ambiente no qual o aplicativo está em execução:

- JM 3.0 Em IBM MQ 9.3.0, se o aplicativo tiver carregado o IBM MQ classes for Jakarta Messaging do arquivo JAR realocável `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) ou o IBM MQ classes for JMS do arquivo JAR realocável `com.ibm.mq.allclient.jar` (JMS 2.0), o rastreio será gravado em um arquivo chamado `mqjavaclient_%PID%.cl%u.trc`.
- Se o aplicativo tiver carregado o IBM MQ classes for JMS do arquivo JAR relocalizável `com.ibm.mq.allclient.jar`, o rastreio será gravado em um arquivo chamado `mqjavaclient_%PID%.cl%u.trc`.
- Se o aplicativo tiver carregado o IBM MQ classes for JMS do arquivo JAR `com.ibm.mqjms.jar`, o rastreio será gravado em um arquivo chamado `mqjava_%PID%.cl%u.trc`.

em que *%PID%* é o identificador de processo do aplicativo que está sendo rastreado e *%u* é um número exclusivo para diferenciar arquivos entre os encadeamentos que executam rastreio sob diferentes carregadores de classes Java.

Se um ID de processo estiver indisponível, um número aleatório será gerado e prefixado com a letra `f`. Para incluir o ID do processo em um nome de arquivo você especificar, use a sequência `%PID%`.

Se você especificar um diretório alternativo, ele deve existir e você deve ter permissão de gravação para esse diretório. Se você não tiver permissão de gravação, a saída de rastreio será gravada para `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList é uma lista de pacotes e classes que são rastreados ou os valores especiais `ALL` ou `NONE`.

Nomes de classe ou pacote separados com um ponto e vírgula, `;`. O *includeList* é padronizado como `ALL`e rastreia todos os pacotes e classes no IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

Nota: É possível incluir um pacote, mas depois excluir os subpacotes desse pacote. Por exemplo, se você incluir o pacote `a.b` e excluir o pacote `a.b.x`, o rastreio inclui tudo no `a.b.y` e `a.b.z`, mas não `a.b.x` ou `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList é uma lista de pacotes e classes que não são rastreados ou os valores especiais ALL ou NONE.

Nomes de classe ou pacote separados com um ponto e vírgula, ;. *excludeList* é padronizado como NONEe, portanto, não exclui nenhum pacote e classe no IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging do rastreio.

Nota: É possível excluir um pacote, mas depois incluir os subpacotes desse pacote. Por exemplo, se você excluir o pacote a.b e incluir o pacote a.b.x, o rastreio incluirá tudo em a.b.x e a.b.x.1, mas não a.b.y ou a.b.z.

Qualquer pacote ou classe que estiver especificado, no mesmo nível, como ambos incluídos e excluídos, será incluído.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes é o número máximo de bytes que são rastreados a partir de qualquer matriz de byte.

Se *maxArrayBytes* for configurado como um número inteiro positivo, ele limitará o número de bytes em uma matriz de bytes que é gravada no arquivo de rastreio. Ela trunca a matriz de bytes depois da gravação de *maxArrayBytes*. Configurar *maxArrayBytes* reduz o tamanho do arquivo de rastreio resultante e reduz o efeito de rastreio no desempenho do aplicativo.

Um valor 0 para esta propriedade significa que nenhum conteúdo de qualquer matriz de bytes é enviado para o arquivo de rastreio.

O valor padrão é -1, o que remove qualquer limite no número de bytes em uma matriz de bytes que são enviados para o arquivo de rastreio.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes é o número máximo de bytes que são gravados em um arquivo de saída de rastreio.

maxTraceBytes trabalha com *traceCycles*. Se o número de bytes de rastreio gravado estiver perto do limite, o arquivo será fechado e um novo arquivo de saída de rastreio será iniciado.

Um valor 0 significa que um arquivo de saída de rastreio tem o comprimento zero. O valor padrão é -1, o que significa que a quantidade de dados a serem gravados em um arquivo de saída de rastreio é ilimitada.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles é o número de arquivos de saída de rastreio para percorrer.

Se o arquivo de saída de rastreio atual atingir o limite especificado por *maxTraceBytes*, o arquivo será fechado. A saída de rastreio adicional é gravada para o próximo arquivo de saída de rastreio na sequência. Cada arquivo de saída de rastreio é distinto por um sufixo numérico anexado ao nome do arquivo. O arquivo de saída de rastreio atual ou mais recente é mqjms.trc.0, o próximo arquivo de saída de rastreio mais recente é mqjms.trc.1. Os arquivos de rastreio seguem o mesmo padrão de numeração até o limite.

O valor padrão de *traceCycles* é 1. Se *traceCycles* for 1, quando o arquivo de saída de rastreio atual atingir seu tamanho máximo, o arquivo será fechado e excluído. Um novo arquivo de saída de rastreio com o mesmo nome é iniciado. Portanto, existe só um arquivo de saída de rastreio por vez.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters controla se os parâmetros de método e valores de retorno são incluídos no rastreio.

traceParameters é padronizado para TRUE. Se *traceParameters* for configurado como FALSE, somente as assinaturas de método serão rastreadas.

com.ibm.msg.client.commonservices.trace.startup = *startup*
Há uma fase de inicialização de IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging durante a qual recursos são alocados. O recurso de rastreio principal é inicializado durante a fase de alocação de recurso.

Se *startup* estiver configurado como TRUE, o rastreio de inicialização será usado. As informações de rastreio são produzidas imediatamente e incluem a configuração de todos os componentes, incluindo o próprio recurso de rastreio. As informações de rastreio de inicialização podem ser usadas para diagnosticar os problemas de configuração. As informações de rastreio de inicialização são sempre gravadas em `System.err`.

startup é padronizado para FALSE.

startup é verificado antes da inicialização ser concluída. Por essa razão, especifique apenas a propriedade na linha de comandos como uma propriedade do sistema Java. Não especifique no arquivo de configuração IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Configure *compressedTrace* para TRUE para compactar a saída de rastreio.

O valor padrão de *compressedTrace* é FALSE.

Se *compressedTrace* for configurado como TRUE, a saída de rastreio será compactada. O nome do arquivo de saída de rastreio padrão tem a extensão `.trz`. Se a compactação estiver configurada como FALSE, o valor padrão, o arquivo terá a extensão `.trc` para indicar que está descompactado. No entanto, se o nome do arquivo para a saída de rastreio tiver sido especificado em *traceOutputName*, esse nome será usado no lugar; nenhum sufixo será aplicado ao arquivo.

A saída de rastreio compactada é menor do que a descompactada. Como há menos E/S, isso pode ser gravado mais rápido do que o rastreio descompactado. O rastreio compactado tem menos efeito no desempenho de IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging que o rastreio descompactado.

Se *maxTraceBytes* e *traceCycles* estiverem configurados, vários arquivos de rastreio compactados serão criados no lugar de vários arquivos simples.

Se IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging terminar de uma maneira não controlada, um arquivo de rastreio compactado pode não ser válido Por essa razão, a compactação de rastreio deve ser usada apenas quando IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging for fechado de uma maneira controlada Use apenas a compactação de rastreio se os problemas que estão sendo investigados não fizerem com que a própria JVM pare inesperadamente. Não use a compactação de rastreio ao diagnosticar problemas que possam resultar em encerramentos `System.Halt()` ou términos de JVM não controlados e anormais.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel especifica um nível de filtragem para o rastreio. Os níveis de rastreio definidos são os seguintes:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Cada nível de rastreio inclui todos os níveis inferiores. Por exemplo, se o nível de rastreio for configurado em `TRACE_INFO`, qualquer ponto de rastreio com um nível definido de `TRACE_EXCEPTION`, `TRACE_WARNING`ou `TRACE_INFO` será gravado no rastreio. Todos os outros pontos de rastreio são excluídos.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace controla se o serviço de rastreio do cliente IBM MQ JMS é usado em um ambiente do WebSphere Application Server.

Se *standaloneTrace* for configurado como TRUE, as propriedades de rastreio do cliente IBM MQ JMS são usadas para determinar a configuração de rastreio.

Se *standaloneTrace* for configurado como FALSE e o cliente IBM MQ JMS estiver em execução em um contêiner do WebSphere Application Server, o serviço de rastreio do WebSphere Application Server será usado. As informações de rastreio que são geradas dependem das configurações de rastreio do servidor de aplicativos.

O valor padrão de *standaloneTrace* é FALSE.

Sub-rotina de criação de log
Use a Sub-rotina de criação de log para configurar o recurso de log do IBM MQ classes for JMS.

As propriedades a seguir podem ser incluídas na Sub-rotina de criação de log:

com.ibm.msg.client.commonservices.log.outputName = path
O nome do arquivo de log que é usado pelo recurso de log do IBM MQ classes for JMS. O valor padrão é `mqjms.log`, que é gravado no diretório de trabalho atual para o Java Runtime Environment no qual o IBM MQ classes for JMS está em execução.

A propriedade pode ter um dos valores a seguir:

- um único nome de caminho
- uma lista separada por vírgula de nomes de caminho (todos os dados são registrados em todos os arquivos)

Cada nome de caminho pode ser um nome de caminho absoluto ou relativo ou:

"stderr" ou "System.err"
Representa o fluxo de erro padrão.

"stdout" ou "System.out"
Representa o fluxo de saída padrão.

com.ibm.msg.client.commonservices.log.maxBytes
O número máximo de bytes registrados de qualquer chamada para registrar dados da mensagem.

Número inteiro positivo
Os dados são gravados até esse valor de bytes por chamada de log.

0
Nenhum dado é gravado.

-1
Dados ilimitados são gravados (padrão).

com.ibm.msg.client.commonservices.log.limit
O número máximo de bytes que são gravados em qualquer arquivo de log (o padrão é 262144).

Número inteiro positivo
Os dados são gravados até esse valor de bytes por arquivo de log.

0
Nenhum dado é gravado.

-1
Dados ilimitados são gravados.

com.ibm.msg.client.commonservices.log.count
O número de arquivos de log para percorrer. À medida que cada arquivo atingir o `com.ibm.msg.client.commonservices.trace.limit`, o rastreio começará no próximo arquivo. O padrão é 3.

Número inteiro positivo
Número de arquivos para percorrer.

0
Um único arquivo.

Sub-rotina de Java SE Specifics
Use a sub-rotina de Java SE Specifics para configurar propriedades que são usadas quando IBM MQ classes for JMS estão sendo usados em um ambiente Java Standard Edition.

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE | FALSE
Determina se um arquivo JavaCore é gravado imediatamente após o IBM MQ classes for JMS ter gerado um arquivo FDC. Se for configurado para TRUE um arquivo JavaCore será produzido no diretório ativo do Java Runtime Environment no qual IBM MQ classes for JMS estão em execução.

true
Gere o JavaCore, sujeito a capacidade do Java Runtime Environment para fazer isso.

falso
Não gere JavaCore; esse é o valor padrão.

Sub-rotina propriedades doIBM MQ
Use a sub-rotina de propriedades do IBM MQ para configurar propriedades que afetam como o IBM MQ classes for JMS interage com o IBM MQ.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
Quando um aplicativo que usa o IBM MQ classes for JMS está se conectando a um gerenciador de filas do IBM MQ usando o modo de migração do provedor de sistemas de mensagens do IBM MQ, o IBM MQ classes for JMS usa um tamanho de buffer padrão de 4 KB ao receber mensagens. Se a mensagem que o aplicativo está tentando obter for maior que 4 KB, o IBM MQ classes for JMS redimensiona o buffer para ser grande o suficiente para acomodar a mensagem. O tamanho do buffer maior é, então, usado quando as mensagens subsequentes são recebidas.

Essa propriedade controla quando o tamanho do buffer é reduzido de volta para 4 KB. Por padrão, quando dez mensagens consecutivas, que são menores que o tamanho de buffer maior, são recebidas, o tamanho do buffer é reduzido de volta para 4 KB. Para reconfigurar o tamanho do buffer de volta para 4 KB toda vez que uma mensagem for recebida, configure a propriedade com o valor 0.

0

O buffer sempre será reconfigurado para o tamanho padrão.

10

Esse é o valor-padrão. O buffer será redimensionado após a décima mensagem.

com.ibm.msg.client.wmq.receiveConversionCCSID
Quando um aplicativo que está usando o IBM MQ classes for JMS está se conectando a um gerenciador de filas do IBM MQ usando o modo normal do provedor de mensagens do IBM MQ, a propriedade `receiveConversionCCSID` pode ser configurada para substituir o valor padrão do CCSID na estrutura MQMD que é usada para receber mensagens do gerenciador de filas. Por padrão, o MQMD contém um campo CCSID configurado para 1208, mas isso pode ser mudado se, por exemplo, o gerenciador de filas for incapaz de converter mensagens para essa página de códigos.

Os valores válidos são qualquer número CCSID válido ou um dos seguintes valores:

-1
Use o padrão de plataforma.

1208
Esse é o valor-padrão.

Sub-rotina de especificações de modo do cliente
Use a sub-rotina de especificações de modo do cliente para especificar as propriedades usadas quando o IBM MQ classes for JMS se conectar a um gerenciador de filas que está usando o transporte CLIENT.

com.ibm.mq.polling.RemoteRequestEntry
Especifica o intervalo de pesquisa que o IBM MQ classes for JMS usa para verificar as conexões interrompidas quando está aguardando por uma resposta a partir de um gerenciador de filas.

Número inteiro positivo
O número de milissegundos para aguardar antes da verificação. O valor padrão é 10000 ou 10 segundos. O valor mínimo é 3000 e valores inferiores são tratados da mesma maneira que esse valor mínimo.

Propriedades usadas para configurar o comportamento do cliente JMS
Use essas propriedades para configurar o comportamento do cliente JMS.

com.ibm.mq.jms.SupportMQExtensions VERDADEIRO|FALSO
A especificação JMS 2.0 introduz mudanças na forma como certos comportamentos funcionam. O IBM MQ 8.0 inclui a propriedade `com.ibm.mq.jms.SupportMQExtensions`, que pode ser configurada como *TRUE* para reverter a implementações anteriores esses comportamentos mudados. Reverter os comportamentos mudados pode ser necessário para alguns aplicativos JMS 2.0 e também para alguns aplicativos que usam a API do JMS 1.1, mas são executados no IBM MQ 8.0 IBM MQ classes for JMS.

TRUE
As três áreas de funcionalidade a seguir são revertidas ao configurar o SupportMQExtensions para TRUE:

Prioridade da mensagem
Uma prioridade por ser designada às mensagens, *0 – 9*. Antes do JMS 2.0, as mensagens também podiam usar o valor *-1*, indicando que a prioridade padrão de uma fila é usada. O JMS 2.0 não permite que uma prioridade de mensagem de *-1* seja configurada. Ativar `SupportMQExtensions` permite que o valor de *-1* seja usado.

Identificador de cliente
A especificação JMS 2.0 requer que os identificadores de cliente não nulos sejam verificados quanto à exclusividade quando fizerem uma conexão. Ativar o `SupportMQExtensions` significa que esse requisito é desconsiderado e que um ID de cliente pode ser reutilizado.

`NoLocal`
A especificação JMS 2.0 requer que, quando essa constante estiver ativada, um consumidor não poderá receber mensagens publicadas pelo mesmo ID de cliente. Antes do JMS 2.0, esse atributo era configurado em um assinante para evitar que recebesse mensagens publicadas por sua própria conexão. Ativar `SupportMQExtensions` reverterá esse comportamento para sua implementação anterior.

FALSE
As mudanças de comportamento são retidas.

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= VERDADEIRO|FALSO
No IBM MQ 8.0.0 Fix Pack 2, após um aplicativo ter enviado uma mensagem de Bytes ou Fluxo, o IBM MQ classes for JMS pode configurar o estado da mensagem que acabou de ser enviada para somente leitura ou somente gravação.

TRUE
Os objetos são configurados para somente leitura depois de serem enviados. Definir esse valor mantém a compatibilidade com a especificação JMS 2.0

FALSE
Os objetos são configurados para somente gravação depois de serem enviados. Esse é o valor-padrão.

Conceitos relacionados
“Propriedade SupportMQExtensions” na página 334
A especificação JMS 2.0 introduziu mudanças na maneira como determinados comportamentos funcionam.. IBM MQ 8.0 e posterior inclui a propriedade **com.ibm.mq.jms.SupportMQExtensions**, que pode ser configurada como TRUE para reverter esses comportamentos alterados de volta para as implementações anteriores

STEPLIB configuration for IBM MQ classes for JMS on z/OS
On z/OS, the STEPLIB used at run time must contain the IBM MQ SCSQAUTH and SCSQANLE libraries. Specify these libraries in the startup JCL or using the `.profile` file.

From z/OS UNIX System Services, you can add these using a line in your `.profile` as shown in the following code snippet, replacing `thlqual` with the high-level data set qualifier that you chose when installing IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In other environments, you typically need to edit the startup JCL to include SCSQAUTH and SCSQANLE on the STEPLIB concatenation:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS e ferramentas de gerenciamento de software
As ferramentas de gerenciamento de software, como o Apache Maven, podem ser usadas com o IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Muitas das grandes organizações de desenvolvimento usam essas ferramentas para gerenciar centralmente os repositórios de bibliotecas de terceiros.

O IBM MQ classes for JMS e o IBM MQ classes for Jakarta Messaging são compostos de vários arquivos JAR. Quando você estiver desenvolvendo aplicativos de linguagem do Java usando essa API, uma instalação de um IBM MQ Server, IBM MQ Client ou IBM MQ Client SupportPac será requerida na máquina em que o aplicativo está sendo desenvolvido.

Se desejar usar essa ferramenta e incluir os arquivos JAR que formam o IBM MQ classes for JMS em um repositório gerenciado centralmente, os pontos a seguir deverão ser observados:

- Um repositório ou contêiner deve ser disponibilizado somente para os desenvolvedores em sua organização. Qualquer distribuição fora da organização não é permitida.
- O repositório precisa conter um conjunto completo e consistente de arquivos JAR a partir de uma liberação única ou fix pack do IBM MQ.
- Você é responsável por atualizar o repositório com qualquer manutenção fornecida pelo suporte IBM.

Os seguintes arquivos JAR precisam ser instalados no repositório:

- JMS 2.0 `com.ibm.mq.allclient.jar` e `jms.jar` serão necessários se você estiver usando o IBM MQ classes for JMS
- JM 3.0 `com.ibm.mq.jakarta.client.jar` e `jakarta.jms-api.jar` são necessários se estiver usando IBM MQ classes for Jakarta Messaging.
- `fscontext.jar` será necessário se você estiver usando IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging e estiver acessando JMS objetos administrados armazenados em um contexto JNDI do sistema de arquivos.
- `providerutil.jar` será necessário se você estiver usando o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging e estiver acessando JMS objetos administrados que são armazenados em um contexto JNDI do sistema de arquivos.
- O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS são necessários para suporte para JREs nãoIBM . Para obter mais informações, consulte Suporte para JREs não IBM.

Executando aplicativos IBM MQ classes for JMS no Java security manager

O IBM MQ classes for JMS pode executar com o Java security manager ativado. Para executar aplicativos com sucesso com o Java security manager ativado, deve-se configurar o Java Virtual Machine (JVM) com um arquivo de configuração de política adequado.

A maneira mais simples de criar um arquivo de definição de política adequado é mudar o arquivo de configuração de política fornecido com o Java runtime environment (JRE). Na maioria dos sistemas, esse arquivo está no diretório `lib/security/java.policy` em relação ao seu diretório JRE. É possível editar o arquivo de configuração de política usando seu editor preferencial ou usando o programa policy tool fornecido com o JRE.

Exemplo de arquivo de configuração de política

Aqui está um exemplo de um arquivo de configuração de política que permite que o IBM MQ classes for JMS seja executado com sucesso no gerenciador de segurança padrão. Este arquivo precisará ser customizado para especificar os locais de determinados arquivos e diretórios: *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado, *MQ_DATA_DIRECTORY* representa o local do diretório de dados do MQ e *QM_NAME* é o nome do gerenciador de filas para o qual o acesso está sendo configurado.

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
  permission javax.management.MBeanPermission "*","*";

  // We need to be able to read manifests etc from the jar files in the installation directory
  permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

  //For the client transport type.
  permission java.net.SocketPermission "*","connect,resolve";

  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";

  //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

  //For applications that use User Exits
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
  permission java.lang.RuntimePermission "createClassLoader";

  //Required for the z/OS platform
  permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

  // Used by the internal ConnectionFactory implementation
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

  // Used for controlled class loading
  permission java.lang.RuntimePermission "setContextClassLoader";

  // Used to default the Application name in Client mode connections
  permission java.util.PropertyPermission "sun.java.command","read";

  // Used by the IBM JSSE classes
  permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

  //Required to determine if an IBM Java Runtime is running in FIPS mode,
  //and to modify the property values status as required.
  permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
  permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
  //Required if an IBM FIPS provider is to be used for SSL communication.
  permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

  // Required for non-IBM Java Runtimes that establish secure client
```

```
  // transport mode connections using mutual TLS authentication
  permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
  permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

No exemplo, a instrução `grant` contém as permissões requeridas pelo IBM MQ classes for JMS. Para usar essas instruções de concessão no arquivo de configuração de política, pode ser necessário modificar os nomes do caminho, dependendo do local em que você instalou o IBM MQ classes for JMS e o local em que você armazena seus aplicativos.

Os aplicativos de amostra fornecidos com o IBM MQ classes for JMS e scripts a executá-los, não ativam o gerenciador de segurança.

Importante:

O utilitário de rastreio do IBM MQ classes for JMS requer permissões adicionais, já que ele executa consulta adicional das propriedades do sistema e também operações adicionais do sistema de arquivos.

Um arquivo de política de segurança de template adequado para execução sob um gerenciador de segurança com rastreio ativado é fornecido no diretório `samples/wmqjava` da instalação IBM MQ como `example.security.policy`.

Configuração de pós-instalação para os aplicativos IBM MQ classes for JMS

Este tópico informa quais autoridades do aplicativos IBM MQ classes for JMS precisam para acessar os recursos de um gerenciador de filas. Ele também introduz os modos de conexão e descreve como configurar um gerenciador de filas para que os aplicativos possam se conectar no modo cliente.

Lembre-se de verificar o arquivo leia-me do IBM MQ. Ele pode conter informações que suplantam as informações neste tópico.

Objetos usados pelo JMS que requerem autorização para usuários não privilegiados

Os usuários não privilegiados precisam de autorização concedida para acessar as filas usadas pelo JMS. Cada aplicativo JMS precisa de autorização para o gerenciador de filas com o qual trabalha.

Para obter detalhes sobre o controle de acesso no IBM MQ, consulte Configurando a segurança.

Os aplicativos IBM MQ classes for JMS precisam da autoridade `connect` e `inq` para o gerenciador de filas. É possível configurar as autorizações apropriadas usando o comando de controle **setmqaut**, por exemplo:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Para o domínio ponto a ponto, as seguintes autoridades são necessárias:

- As filas usadas pelos objetos MessageProducer precisam da autoridade `put`.
- As filas usadas por objetos MessageConsumer e QueueBrowser precisam das autoridades `get`, `inq` e `browse`.
- O método QueueSession.createTemporaryQueue() precisa de acesso à fila de modelo especificada pela propriedade TEMPMODEL do objeto QueueConnectionFactory. Por padrão, esta fila de modelo é SYSTEM.TEMP.MODEL.QUEUE.

Se qualquer uma destas filas for filas de alias, suas filas de destino irão requerer a autoridade de consulta. Se a fila de destino for uma fila de cluster, ela também irá requerer a autoridade de procura.

Para o domínio de publicação/assinatura, as seguintes filas serão usadas se o IBM MQ classes for JMS estiver se conectando a um gerenciador de filas do IBM MQ no modo de migração do provedor com sistema de mensagens do IBM MQ:

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE

- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Para obter informações adicionais sobre o modo de migração do provedor de sistemas de mensagens do IBM MQ , consulte Configurando a propriedade JMS **PROVIDERVERSION**

Além disso, se o IBM MQ classes for JMS estiver se conectando a um gerenciador de filas neste modo, qualquer aplicativo que publicar mensagens precisará de acesso à fila de fluxo especificada pelo objeto do tópico ou TopicConnectionFactory. Por padrão, esta fila é SYSTEM.BROKER.DEFAULT.STREAM.

Se você usar ConnectionConsumer, o IBM MQ Resource Adapter ou o provedor de sistema de mensagens do WebSphere Application Server IBM MQ, a autorização adicional poderá ser necessária.

As filas a serem lidas pelo ConnectionConsumer deve ter as autoridades `get`, `inq` e `browse`. A fila de mensagens não entregues do sistema e qualquer fila de backout-requeue ou fila de relatório usada pelo ConnectionConsumer deve ter as autoridades `put` e `passall`.

Quando um aplicativo usar o modo normal do provedor com sistema de mensagens do IBM MQ para executar mensagens de publicação/assinatura, o aplicativo fará uso da funcionalidade de publicação/assinatura integrada fornecida pelo gerenciador de filas. Consulte Segurança de publicação/assinatura para obter informações sobre como proteger os tópicos e as filas usados.

Modos de conexão para IBM MQ classes for JMS
Um aplicativo IBM MQ classes for JMS pode se conectar a um gerenciador de filas no modo de cliente ou de ligações. No modo cliente, o IBM MQ classes for JMS se conecta ao gerenciador de filas através de TCP/IP. No modo de ligações, o IBM MQ classes for JMS se conecta diretamente ao gerenciador de filas usando o Java Native Interface (JNI).

No z/OS, o modo de ligações pode ser usado em qualquer ambiente, mas o modo de cliente pode ser usado apenas nos seguintes ambientes:

- No WebSphere Application Server ou no WebSphere Liberty Profile conectando-se a qualquer gerenciador de filas, em qualquer plataforma, incluindo z/OS
- Em ambientes em lote ao conectar a um gerenciador de filas do IBM MQ for z/OS , em execução em qualquer LPAR.

Um aplicativo em execução em qualquer outra plataforma pode se conectar a um gerenciador de filas tanto no modo de ligações como de cliente.

É possível usar a versão atual ou qualquer anterior suportada do IBM MQ classes for JMS com um gerenciador de filas atual, e é possível usar uma versão do gerenciador de filas atual ou anterior suportada com a versão atual do IBM MQ classes for JMS. Se diferentes versões forem misturadas, a função será limitada ao nível da versão anterior.

As seções a seguir descrevem cada um dos modos de conexão em mais detalhes.

Modo do cliente

Para se conectar a um gerenciador de filas no modo cliente, um aplicativo IBM MQ classes for JMS pode ser executado no mesmo sistema no qual o gerenciador de filas está em execução ou em um sistema diferente. Em cada caso, o IBM MQ classes for JMS se conecta ao gerenciador de filas por meio de TCP/IP.

Modo de ligações

Para se conectar a um gerenciador de filas no modo de ligações, um aplicativo IBM MQ classes for JMS deve ser executado no mesmo sistema no qual o gerenciador de filas está em execução.

O IBM MQ classes for JMS se conecta diretamente ao gerenciador de filas usando o Java Native Interface (JNI). Para usar o transporte de ligações, o IBM MQ classes for JMS deve ser executado em um ambiente

que possui acesso às bibliotecas do IBM MQ Java Native Interface; consulte “Configurando as bibliotecas do Java Native Interface (JNI)” na página 97 para obter informações adicionais.

O IBM MQ classes for JMS suporta os seguintes valores de *ConnectOption*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Para mudar as opções de conexão usadas pelo IBM MQ classes for JMS, modifique a propriedade Connection Factory CONNOPT.

Para obter informações adicionais sobre opções de conexão, consulte “Conectando-se a um gerenciador de filas usando a chamada MQCONNX” na página 749

Para usar o transporte de ligações, o Java Runtime Environment que está sendo usado deve suportar o Identificador de conjunto de caracteres codificados (CCSID) do gerenciador de filas ao qual o IBM MQ classes for JMS está se conectando.

Os detalhes sobre como determinar quais CCSIDs são suportados por um Java Runtime Environment podem ser localizados no IBM MQ FDC com o ID de análise 21 gerado ao usar as classes do IBM MQ V7 para Java ou classes do IBM MQ V7 para JMS.

Configurando seu gerenciador de filas para que os aplicativos IBM MQ classes for JMS possam se conectar no modo cliente
Para configurar o seu gerenciador de filas para que os aplicativos IBM MQ classes for JMS possam se conectar no modo cliente, deve-se criar uma definição de canal de conexão do servidor e iniciar um listener.

Criando uma definição de canal de conexão do servidor

Em todas as plataformas, é possível usar o comando DEFINE CHANNEL do MQSC para criar uma definição de canal de conexão do servidor. Consulte o exemplo a seguir:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i No IBM i, é possível usar o comando de CL CRTMQMCHL em vez disso, como no exemplo a seguir:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

Nesse comando, *QMGRNAME* é o nome do gerenciador de filas.

Windows Linux No Linux e no Windows, também é possível criar uma definição de canal de conexão do servidor usando o IBM MQ Explorer.

z/OS No z/OS, é possível usar as operações e os painéis de controle para criar uma definição de canal de conexão do servidor.

O nome do canal (JAVA.CHANNEL nos exemplos anteriores) deve ser o mesmo que o nome do canal especificado pela propriedade CHANNEL do connection factory que seu aplicativo usa para se conectar ao gerenciador de filas. O valor padrão da propriedade CHANNEL é SYSTEM.DEF.SVRCONN.

Iniciando um listener

Deve-se iniciar um listener para o gerenciador de filas se um ainda não estiver iniciado.

Multi Em Multiplataformas, é possível usar o comando START LISTENER do MQSC para iniciar um listener após primeiramente criar um objeto de listener usando o comando DEFINE LISTENER do MQSC, conforme mostrado no exemplo a seguir:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS No z/OS, você usa somente o comando START LISTENER, como no exemplo a seguir, mas observe que o espaço de endereço do inicializador de canais deve ser iniciado antes que seja possível iniciar um listener:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i No IBM i, também é possível usar o comando STRMQMLSR de CL para iniciar um listener, como no exemplo a seguir:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

Nesse comando, *QMGRNAME* é o nome do gerenciador de filas.

ALW No AIX, Linux, and Windows, é possível também usar o comando de controle **runmqlsr** para iniciar um listener, como no exemplo a seguir:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

Nesse comando, *QMgrName* é o nome do gerenciador de filas.

Windows Linux No Linux e Windows, é possível também iniciar um listener usando o IBM MQ Explorer.

z/OS No z/OS, também é possível usar as operações e os painéis de controle para iniciar um listener.

O número da porta na qual o listener está atendendo deve ser o mesmo que o número da porta especificado pela propriedade PORT do connection factory que seu aplicativo usa para se conectar ao gerenciador de filas. O valor padrão da propriedade PORT é 1414.

O IVT ponto a ponto para IBM MQ classes for JMS

Um programa de teste de verificação de instalação ponto a ponto (IVT) é fornecido com IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging. O programa se conecta a um gerenciador de filas no modo de ligações ou de cliente, envia uma mensagem à fila chamada SYSTEM.DEFAULT.LOCAL.QUEUE e, em seguida, recebe a mensagem da fila. O programa pode criar e configurar todos os objetos que requer dinamicamente no tempo de execução ou pode usar JNDI para recuperar objetos administrados a partir de um serviço de diretório.

Execute o teste de verificação de instalação sem antes usar JNDI porque o teste é autocontido e não requer o uso de um serviço de diretório. Para obter uma descrição de objetos administrados, consulte Configurando objetos do JMS usando a ferramenta de administração.

O teste de verificação de instalação ponto a ponto sem usar JNDI

Neste teste, o programa IVT cria e configura todos os objetos que requer dinamicamente no tempo de execução e não usa JNDI.

Multi Em multiplataformas, um script é fornecido para executar o programa IVT O script é chamado de **IVTRun** em sistemas AIX and Linux e **IVTRun.bat** em Windows o script está localizado no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS . O caminho de classe deve conter `com.ibm.mqjms.jar`..

Para executar o teste no modo de ligações, insira o comando a seguir:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Para executar o teste no modo cliente, primeiro configure o gerenciador de filas, conforme descrito em "Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms" na página 1081 Observe que o canal a ser usado é padronizado como **SYSTEM.DEF.SVRCONN** e a fila a ser usada é **SYSTEM.DEFAULT.LOCAL.QUEUE**, em seguida, insira o comando a seguir:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

z/OS Nenhum script equivalente é fornecido nos sistemas z/OS .. Em vez disso, você executa o IVT no modo de ligações chamando a classe Java diretamente No z/OS, você escolhe entre duas instâncias funcionalmente idênticas do programa IVT:

- `com.ibm.mq.jms.MQJMSIVT`, que está disponível com IBM MQ classes for JMS (JMS 2.0). Para usar esse programa, o caminho de classe deve conter `com.ibm.mqjms.jar` ou `com.ibm.mq.allclient.jar`
- `com.ibm.mq.jakarta.jms.MQJMSIVT`, que está disponível com IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0). Para usar esse programa, o caminho de classe deve conter `com.ibm.mq.jakarta.client.jar`

Para executar o teste no modo de ligações em z/OS, insira o comando a seguir:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

O parâmetros nos comandos têm os significados a seguir:

-m *qmgr*
O nome do gerenciador de filas ao qual o programa IVT se conecta. Se você executar o teste no modo de ligações e omitir esse parâmetro, o programa IVT se conecta ao gerenciador de filas padrão.

-host *hostname*
O nome do host ou o endereço IP do sistema em que o gerenciador de filas está em execução.

-port *port*
O número da porta na qual o listener do gerenciador de filas está atendendo. O valor padrão é 1414.

-channel *channel*
O nome do canal MQI que o programa IVT usa para se conectar ao gerenciador de filas. O valor padrão é `SYSTEM.DEF.SVRCONN`.

-v *providerVersion*
O nível de liberação do gerenciador de filas ao qual o programa IVT espera se conectar.

Esse parâmetro é usado para configurar a propriedade PROVIDERVERSION de um objeto MQQueueConnectionFactory e tem os mesmos valores válidos que os da propriedade PROVIDERVERSION. Portanto, para obter mais informações sobre esse parâmetro, incluindo os valores válidos, consulte JMS: mudanças na propriedade PROVIDERVERSION e a descrição da propriedade PROVIDERVERSION em Propriedades de objetos do IBM MQ classes for JMS.

O valor padrão é `unspecified`.

-ccsid *ccsid*
O identificador de conjunto de caracteres codificados (CCSID) ou página de códigos a ser usado pela conexão. O valor padrão é 819.

-t
O rastreio está ativado. Por padrão, o rastreio está desativado.

Um teste bem-sucedido produz uma saída semelhante à saída de amostra a seguir:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

O teste de verificação de instalação ponto a ponto usando JNDI

Multi

Neste teste, o programa IVT usa JNDI para recuperar objetos administrados de um serviço de diretório.

Antes que seja possível executar o teste, deve-se configurar um serviço de diretório baseado em um servidor Lightweight Directory Access Protocol (LDAP) ou no sistema de arquivos local. Deve-se também configurar a ferramenta de administração do IBM MQ JMS para que possa usar o serviço de diretório para armazenar objetos administrados. Para obter mais informações sobre esses pré-requisitos, consulte “Pré-requisitos para o IBM MQ classes for JMS” na página 89. Para obter mais informações sobre como configurar a ferramenta de administração IBM MQ JMS, consulte Configurando a ferramenta de administração JMS.

O programa IVT deve ser capaz de usar JNDI para recuperar um objeto MQQueueConnectionFactory e um objeto MQQueue do serviço de diretório. Um script é fornecido para criar esses objetos administrados para você. O script é chamado IVTSetup em sistemas AIX and Linux e IVTSetup.bat no Windows, e está

no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS. Para executar o script, insira o comando a seguir:

```
IVTSetup
```

O script chama a ferramenta de administração do IBM MQ JMS para criar os objetos administrados.

O objeto de MQQueueConnectionFactory é ligado com o nome ivtQCF e é criado com os valores padrão para todas as suas propriedades, o que significa que o programa IVT executa no modo de ligações e se conecta ao gerenciador de filas padrão. Se desejar que o programa IVT seja executado no modo cliente ou se conecte a um gerenciador de filas diferente do gerenciador de filas padrão, deve-se usar a ferramenta de administração do IBM MQ JMS ou o IBM MQ Explorer para mudar as propriedades apropriadas do objeto MQQueueConnectionFactory. Para obter informações sobre como usar a ferramenta de administração do IBM MQ Explorer JMS, consulte Configurando objetos do JMS usando a ferramenta de administração. Para obter informações sobre como usar o IBM MQ Explorer, consulte Introdução ao IBM MQ Explorer ou a ajuda fornecida com o IBM MQ Explorer.

O objeto MQQueue é ligado com o nome ivtQ e é criado com os valores padrão para todas as suas propriedades, exceto para a propriedade QUEUE, que tem o valor SYSTEM.DEFAULT.LOCAL.QUEUE.

Quando tiver criado os objetos administrados, será possível executar o programa IVT. Para executar o teste usando JNDI, insira o comando a seguir:

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

Os parâmetros no comando têm os significados a seguir:

-url " *providerURL* "
O localizador uniforme de recursos (URL) do serviço de diretório. A URL pode ter um dos formatos a seguir:

- `ldap://hostname/contextName` , para um serviço de diretório com base em um servidor LDAP
- `file:/directoryPath` , para um serviço de diretório com base no sistema de arquivos local

Observe que a URL deve ser colocada entre aspas (").

-icf *initCtxFact*
O nome da classe do factory de contexto inicial, que deve ser um dos valores a seguir:

- `com.sun.jndi.ldap.LdapCtxFactory`, para um serviço de diretório com base em um servidor LDAP. Esse é o valor-padrão.
- `com.sun.jndi.fscontext.RefFSContextFactory`, para um serviço de diretório com base no sistema de arquivos local.

-t
O rastreio está ativado. Por padrão, o rastreio está desativado.

Um teste bem-sucedido produz uma saída semelhante àquela de um teste bem-sucedido sem usar JNDI. A diferença principal é que a saída indica que o teste está usando JNDI para recuperar um objeto MQQueueConnectionFactory e um objeto MQQueue.

Embora não seja estritamente necessário, uma boa prática é limpar após o teste excluindo os objetos administrados criados pelo script IVTSetup. Um script é fornecido para esse propósito. O script é chamado IVTTidy em sistemas AIX and Linux e IVTTTidy.bat no Windows, e está no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS.

Determinação de problema para o teste de verificação de instalação ponto a ponto

O teste de verificação de instalação pode falhar pelas razões a seguir:

- Se o programa IVT grava uma mensagem indicando que não pode localizar uma classe, verifique se o caminho de classe está configurado corretamente, conforme descrito em “Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging” na página 95.
- O teste pode falhar com a mensagem a seguir:

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

  e um código de razão associado de 2059. As variáveis na mensagem têm os significados a seguir:

  qmgr
  O nome do gerenciador de filas ao qual o programa IVT está tentando se conectar. Essa inserção de mensagem estará em branco se o programa IVT estiver tentando se conectar com o gerenciador de filas padrão no modo de ligações.

  connMode
  O modo de conexão, que é `Bindings` ou `Client`.

  HOSTNAME
  O nome do host ou o endereço IP do sistema em que o gerenciador de filas está em execução.

  Esta mensagem significa que o gerenciador de filas ao qual o programa IVT está tentando se conectar não está disponível. Verifique se o gerenciador de filas está em execução e, se o programa IVT estiver tentando se conectar com o gerenciador de filas padrão, certifique-se de que o gerenciador de filas esteja definido como o gerenciador de filas padrão para seu sistema.
- O teste pode falhar com a mensagem a seguir:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

  Esta mensagem significa que a fila SYSTEM.DEFAULT.LOCAL.QUEUE não existe no gerenciador de filas ao qual o programa IVT está conectado. Como alternativa, se a fila existir, o programa IVT não poderá abrir a fila porque não está ativado para efetuar put e get de mensagens. Verifique se a fila existe e se está ativada para efetuar out e get de mensagens.
- O teste pode falhar com a mensagem a seguir:

```
Unable to bind to object
```

  Essa mensagem significa que há uma conexão com o servidor LDAP, mas que o servidor LDAP não está configurado corretamente. O servidor LDAP não está configurado para armazenar objetos Java ou as permissões nos objetos ou o sufixo não estão corretos. Para obter mais ajuda nessa situação, consulte a documentação de seu servidor LDAP.
- O teste pode falhar com a mensagem a seguir:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

  Esta mensagem significa que o gerenciador de filas não é configurado corretamente para aceitar uma conexão do cliente a partir de seu sistema. Consulte “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081 para obter detalhes.

O IVT de publicar/assinar para IBM MQ classes for JMS

Um programa de teste de verificação de instalação (IVT) de publicar/assinar é fornecido com o IBM MQ classes for JMS. O programa se conecta a um gerenciador de filas no modo de ligações ou do cliente, assina um tópico, publica uma mensagem sobre o tópico e, em seguida, recebe a mensagem que acaba de publicar. O programa pode criar e configurar todos os objetos que requer dinamicamente no tempo de execução ou pode usar JNDI para recuperar objetos administrados a partir de um serviço de diretório.

Execute o teste de verificação de instalação sem antes usar JNDI porque o teste é autocontido e não requer o uso de um serviço de diretório. Para obter uma descrição de objetos administrados, consulte Configurando objetos do JMS usando a ferramenta de administração.

O teste de verificação da instalação de publicar/assinar sem usar JNDI

Neste teste, o programa IVT cria e configura todos os objetos que requer dinamicamente no tempo de execução e não usa JNDI.

Um script é fornecido para executar o programa IVT. O script é denominado PSIVTRun em sistemas AIX and Linux e PSIVTRun.bat no Windows, e está no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS.

Para executar o teste no modo de ligações, insira o comando a seguir:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

Para executar o teste no modo cliente, primeiro, configure o gerenciador de filas conforme descrito em "Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms" na página 1081, observando que o canal a ser usado é por padrão SYSTEM.DEF.SVRCONN, em seguida, insira o comando a seguir:

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

O parâmetros nos comandos têm os significados a seguir:

-m *qmgr*
O nome do gerenciador de filas ao qual o programa IVT se conecta. Se você executar o teste no modo de ligações e omitir esse parâmetro, o programa IVT se conecta ao gerenciador de filas padrão.

-host *hostname*
O nome do host ou o endereço IP do sistema em que o gerenciador de filas está em execução.

-port *port*
O número da porta na qual o listener do gerenciador de filas está atendendo. O valor padrão é 1414.

-channel *channel*
O nome do canal MQI que o programa IVT usa para se conectar ao gerenciador de filas. O valor padrão é `SYSTEM.DEF.SVRCONN`.

-bqm *brokerQmgr*
O nome do gerenciador de filas no qual o broker está em execução. O valor padrão é o nome do gerenciador de filas ao qual o programa IVT se conecta.

Esse parâmetro não é relevante para o gerenciador de filas versão v de 7, ou superior.

-v *providerVersion*
O nível de liberação do gerenciador de filas ao qual o programa IVT espera se conectar.

Esse parâmetro é usado para configurar a propriedade PROVIDERVERSION de um objeto MQTopicConnectionFactory e tem os mesmos valores válidos que os da propriedade PROVIDERVERSION. Para obter mais informações sobre esse parâmetro, incluindo os valores válidos, consulte a descrição da propriedade PROVIDERVERSION em Propriedades dos objetos IBM MQ classes for JMS.

O valor padrão é `unspecified`.

-ccsid *ccsid*
O identificador de conjunto de caracteres codificados (CCSID) ou página de códigos a ser usado pela conexão. O valor padrão é 819.

-t
O rastreio está ativado. Por padrão, o rastreio está desativado.

Um teste bem-sucedido produz uma saída semelhante à saída de amostra a seguir:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

O teste de verificação de instalação de publicar/assinar usando JNDI

Neste teste, o programa IVT usa JNDI para recuperar objetos administrados de um serviço de diretório.

Antes que seja possível executar o teste, deve-se configurar um serviço de diretório baseado em um servidor Lightweight Directory Access Protocol (LDAP) ou no sistema de arquivos local. Deve-se também configurar a ferramenta de administração do IBM MQ JMS para que possa usar o serviço de diretório para armazenar objetos administrados. Para obter mais informações sobre esses pré-requisitos, consulte “Pré-requisitos para o IBM MQ classes for JMS” na página 89. Para obter mais informações sobre como configurar a ferramenta de administração IBM MQ JMS, consulte Configurando a ferramenta de administração JMS.

O programa IVT deve ser capaz de usar JNDI para recuperar um objeto MQTopicConnectionFactory e um objeto MQTopic do serviço de diretório. Um script é fornecido para criar esses objetos administrados para você. O script é chamado IVTSetup em sistemas AIX and Linux e IVTSetup.bat no Windows, e está no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS. Para executar o script, insira o comando a seguir:

```
IVTSetup
```

O script chama a ferramenta de administração do IBM MQ JMS para criar os objetos administrados.

O objeto MQTopicConnectionFactory é ligado com o nome ivtTCF e é criado com os valores padrão para todas as suas propriedades, o que significa que o programa IVT é executado no modo de ligações, se conecta ao gerenciador de filas padrão e usa a função de publicar/assinar integrada. Se deseja que o programa IVT seja executado no modo cliente, conecte-se a um gerenciador de filas diferente do gerenciador de filas padrão ou use IBM Integration Bus em vez da função de publicar/assinar integrada. Deve-se usar a ferramenta de administração do IBM MQ JMS ou o IBM MQ Explorer para mudar as propriedades apropriadas do objeto MQTopicConnectionFactory. Para obter informações sobre como usar a ferramenta de administração do IBM MQ JMS, consulte Configurando objetos do JMS usando a ferramenta de administração. Para obter informações sobre como usar o IBM MQ Explorer, consulte a ajuda fornecida com o IBM MQ Explorer.

O objeto MQTopic é ligado ao nome ivtT e é criado com os valores padrão para todas as suas propriedades, exceto para a propriedade TOPIC, que tem o valor MQJMS/PSIVT/Information.

Quando tiver criado os objetos administrados, será possível executar o programa IVT. Para executar o teste usando JNDI, insira o comando a seguir:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Os parâmetros no comando têm os significados a seguir:

-url " *providerURL* "
O localizador uniforme de recursos (URL) do serviço de diretório. A URL pode ter um dos formatos a seguir:

- `ldap://hostname/contextName` , para um serviço de diretório com base em um servidor LDAP
- `file:/directoryPath` , para um serviço de diretório com base no sistema de arquivos local

Observe que a URL deve ser colocada entre aspas (").

-icf *initCtxFact*
O nome da classe do factory de contexto inicial, que deve ser um dos valores a seguir:

- `com.sun.jndi.ldap.LdapCtxFactory`, para um serviço de diretório com base em um servidor LDAP. Esse é o valor-padrão.
- `com.sun.jndi.fscontext.RefFSContextFactory`, para um serviço de diretório com base no sistema de arquivos local.

-t
O rastreio está ativado. Por padrão, o rastreio está desativado.

Um teste bem-sucedido produz uma saída semelhante àquela de um teste bem-sucedido sem usar JNDI. A diferença principal é que a saída indica que o teste está usando JNDI para recuperar um objeto MQTopicConnectionFactory e um objeto MQTopic.

Embora não seja estritamente necessário, uma boa prática é limpar após o teste excluindo os objetos administrados criados pelo script IVTSetup. Um script é fornecido para esse propósito. O script é chamado IVTTidy em sistemas AIX and Linux e IVTTTidy.bat no Windows, e está no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS.

Determinação de problema para o teste de verificação de instalação de publicar/assinar

O teste de verificação de instalação pode falhar pelas razões a seguir:

- Se o programa IVT grava uma mensagem indicando que não pode localizar uma classe, verifique se o caminho de classe está configurado corretamente, conforme descrito em “Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging” na página 95.
- O teste pode falhar com a mensagem a seguir:

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

e um código de razão associado de 2059. As variáveis na mensagem têm os significados a seguir:

qmgr
O nome do gerenciador de filas ao qual o programa IVT está tentando se conectar. Essa inserção de mensagem estará em branco se o programa IVT estiver tentando se conectar com o gerenciador de filas padrão no modo de ligações.

connMode
O modo de conexão, que é `Bindings` ou `Client`.

HOSTNAME
O nome do host ou o endereço IP do sistema em que o gerenciador de filas está em execução.

Esta mensagem significa que o gerenciador de filas ao qual o programa IVT está tentando se conectar não está disponível. Verifique se o gerenciador de filas está em execução e, se o programa IVT estiver tentando se conectar com o gerenciador de filas padrão, certifique-se de que o gerenciador de filas esteja definido como o gerenciador de filas padrão para seu sistema.

- O teste pode falhar com a mensagem a seguir:

```
Unable to bind to object
```

Essa mensagem significa que há uma conexão com o servidor LDAP, mas que o servidor LDAP não está configurado corretamente. O servidor LDAP não está configurado para armazenar objetos Java ou as permissões nos objetos ou o sufixo não estão corretos. Para obter mais ajuda nessa situação, consulte a documentação de seu servidor LDAP.

- O teste pode falhar com a mensagem a seguir:

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

Essa mensagem significa que o gerenciador de filas não está configurado corretamente para aceitar uma conexão do cliente do sistema. Para obter informações adicionais, consulte “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081.

JMS 2.0 *Usando os aplicativos de amostra IBM MQ classes for JMS*

Os aplicativos de amostra do IBM MQ classes for JMS fornecem uma visão geral dos recursos comuns da API do JMS. É possível usá-los para verificar a sua instalação e o servidor de sistema de mensagens configurado e para ajudar a construir os seus próprios aplicativos.

Sobre esta tarefa

Se você precisar de ajuda para criar seus próprios aplicativos, será possível usar os aplicativos de amostra como um ponto de início. Tanto a origem quanto uma versão compilada são fornecidas para cada aplicativo. Revise o código-fonte de amostra e identifique as etapas principais para criar cada objeto necessário para seu aplicativo (ConnectionFactory, Conexão, Sessão, Destino, e um Produtor, ou um Consumidor, ou ambos) e para configurar quaisquer propriedades específicas que sejam necessárias para especificar como você deseja que seu aplicativo funcione. Para obter informações adicionais, consulte “Gravando aplicativos IBM MQ classes for JMS/Jakarta Messaging” na página 142. As amostras podem estar sujeitas a mudanças em futuras liberações do IBM MQ.

Para JMS 2.0, Tabela 11 na página 122 mostra onde os aplicativos de amostra IBM MQ classes for JMS são instalados em cada plataforma.

Nota:

JM 3.0 Para IBM MQ classes for Jakarta Messaging, novas amostras estão sendo preparadas..

Tabela 11. Diretórios de instalação para os aplicativos de amostra do IBM MQ classes for JMS	
Plataforma	**Diretório**
AIX Linux	*MQ_INSTALLATION_PATH*/samp/jms/samples
Windows	*MQ_INSTALLATION_PATH*\tools\jms\samples
IBM i	/qibm/proddata/mqm/java/samples/jms/samples
z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

Nesse diretório, há subdiretórios que contêm um ou mais aplicativos de amostra, conforme mostrado em Tabela 12 na página 122.

Tabela 12. Aplicativos de Amostra do IBM MQ classes for JMS	
Nome da amostra	**Descrição**
JmsBrowser.java	Um aplicativo de navegador de fila do JMS que examina todas as mensagens disponíveis na fila nomeada, sem removê-las, na ordem na qual seriam recebidas por um aplicativo consumidor.
JmsConsumer.java	Um aplicativo de navegador de fila do JMS que examina todas as mensagens disponíveis na fila nomeada, sem removê-las, na ordem na qual seriam recebidas por um aplicativo consumidor, examinando a instância de connection factory e a instância de destino em um contexto inicial (essa amostra suporta apenas o contexto do sistema de arquivos).
JmsJndiConsumer.java	Um aplicativo JMS consumidor (receptor ou assinante) que recebe uma mensagem do destino nomeado (fila ou tópico) ao consultar a instância de connection factory e a instância de destino em um contexto inicial (Essa amostra suporta apenas o contexto do sistema de arquivos).
JmsJndiProducer.java	Um aplicativo JMS produtor (emissor ou publicador) que envia uma mensagem simples para o destino nomeado (fila ou tópico) ao consultar a instância de connection factory e a instância de destino em um contexto inicial (Essa amostra suporta apenas o contexto do sistema de arquivos).
JmsProducer.java	Um aplicativo JMS produtor (emissor ou publicador) que envia uma mensagem simples para o destino nomeado (fila ou tópico).
/interactive/	
SampleConsumerJava.java	Receber mensagens de um tópico/fila.
SampleProducerJava.java	Enviar mensagens para um tópico/fila.
/interactive/helper/	
BaseOptions.java	Uma classe abstrata que pode ser estendida para fornecer a funcionalidade de opções de usuário.
IsValidType.java	Classe abstrata para classes verificadoras de validade.

Tabela 12. Aplicativos de Amostra do IBM MQ classes for JMS (continuação)	
Nome da amostra	**Descrição**
JmsApp.java	Uma classe abstrata que pode ser estendida para fornecer funcionalidade de consumidor/produtor.
Keys.java	Um conjunto de chaves que definem opções para os aplicativos de amostra.
Literals.java	Um conjunto de literais constantes.
MyContext.java	O contexto no qual as opções são apresentadas.
Options.java	Fornece funcionalidade para opções de usuário.
OptionsPresenter.java	Contexto no qual as opções atuais são apresentadas.
/simple/	
SimpleAsyncPutPTP.java	Um aplicativo simples para o sistema de mensagens ponto a ponto; a mensagem é enviada de forma assíncrona (também conhecida como o sistema de mensagens *fire-and-forget*). Nenhuma mensagem é recebida.
SimpleDurableSub.java	Um aplicativo simples que demonstra o recurso de assinatura durável.
SimpleJNDILookup.java	Um aplicativo mínimo e simples que demonstra a consulta de objetos JMS usando o contexto inicial. Nenhuma conexão com o gerenciador de filas é feita e nenhuma mensagem é enviada ou recebida.
SimpleMQMDRead.java	Um aplicativo simples que demonstra como um aplicativo JMS pode aproveitar campos do MQ Message Descriptor (MQMD) como propriedades de mensagem JMS. Nenhuma mensagem é enviada; supõe-se que a fila em uso seja preenchida com algumas mensagens.
SimpleMQMDWrite.java	Um aplicativo simples que demonstra como um aplicativo JMS pode gravar campos do MQ Message Descriptor (MQMD). Nenhuma mensagem é recebida.
SimplePTP.java	Um aplicativo mínimo e simples para o sistema de mensagens ponto a ponto.
SimplePubSub.java	Um aplicativo mínimo e simples para o sistema de mensagens de publicação/assinatura.
SimpleReadAheadPTP.java	Um aplicativo simples para o sistema de mensagens ponto a ponto; as mensagens são transmitidas por meio do gerenciador de filas (também conhecido como recurso de leitura antecipada). Nenhuma mensagem é enviada; supõe-se que a fila em uso seja preenchida com algumas mensagens.
SimpleRequestor.java	Um aplicativo simples que usa um solicitante para enviar uma mensagem de solicitação e, em seguida, aguardar e receber a resposta. Nota: supõe-se que algum outro aplicativo processará a mensagem de solicitação e enviará a mensagem de resposta.
SimpleResponder.java	Um aplicativo simples que recebe uma mensagem em um destino e, em seguida, envia uma resposta para o destino replyTo da mensagem. O aplicativo é escrito para operar em conjunto com a amostra SimpleRequestor.
SimpleRetainedPub.java	Um aplicativo simples que demonstra uma publicação retida. Nenhuma mensagem é recebida.

Tabela 12. Aplicativos de Amostra do IBM MQ classes for JMS (continuação)	
Nome da amostra	**Descrição**
SimpleWMQJMSPTP.java	Um aplicativo mínimo e simples para o sistema de mensagens ponto a ponto.
SimpleWMQJMSPubSub.java	Um aplicativo mínimo e simples para o sistema de mensagens de publicação/assinatura.

O IBM MQ classes for JMS fornece um script chamado `runjms` que pode ser usado para executar os aplicativos de amostra. Esse script configura o ambiente do IBM MQ para permitir que você execute os aplicativos de amostra do IBM MQ classes for JMS.

A Tabela 13 na página 124 mostra o local do script em cada plataforma:

Tabela 13. Localização do script `runjms`	
Plataforma	**Diretório**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	ou
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

Para usar o script `runjms` para chamar um aplicativo de amostra, conclua as seguintes etapas:

Procedimento

1. Ative um prompt de comandos e navegue até o diretório que contém o aplicativo de amostra que você deseja executar.
2. Insira o seguinte comando:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   O aplicativo de amostra exibe uma lista dos parâmetros necessários.
3. Insira o comando a seguir para executar a amostra com estes parâmetros:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Exemplo

Linux Por exemplo, para executar a amostra `JmsBrowser` em Linux, insira os comandos a seguir:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

Conceitos relacionados

“O que é instalado para IBM MQ classes for JMS” na página 90

Vários arquivos e diretórios são criados ao instalar o IBM MQ classes for JMS. No Windows, algumas configurações são executadas durante a instalação, configurando automaticamente variáveis de ambiente. Em outras plataformas e em determinados ambientes de Windows, deve-se configurar as variáveis de ambiente antes de poder executar aplicativos IBM MQ classes for JMS.

Scripts fornecidos com IBM MQ classes for JMS/Jakarta Messaging

Vários scripts são fornecidos para ajudar com tarefas comuns que precisam ser executadas ao usar IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Tabela 14 na página 125 lista todos os scripts e seus usos. Os scripts estão no subdiretório `bin` do diretório de instalação IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging .

Tabela 14. Scripts fornecidos com IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Utilitário	**Uso**
Limpeza [1]	Esse script é mantido para compatibilidade com liberações anteriores, mas não executa nenhuma função. A limpeza manual das informações de assinatura não é mais necessária
DefaultConfiguration	Executa o aplicativo de configuração padrão em plataformas diferentes de Windows.
formatLog [1]	Esse script é mantido para compatibilidade com liberações anteriores, mas não executa nenhuma função. A saída de log é agora produzida em texto legível.
IVTRun [1] IVTSetup [1] IVTTidy [1]	Usado no teste de verificação de instalação ponto a ponto, conforme descrito em "O IVT ponto a ponto para IBM MQ classes for JMS" na página 113.
JM 3.0 JMS30Admin [1]	Executa a ferramenta de administração do IBM MQ Jakarta Messaging, conforme descrito em Iniciando a ferramenta de administração.
JM 3.0 JMS30Admin.config	O arquivo de configuração da ferramenta de administração do IBM MQ Jakarta Messaging, conforme descrito em Configurando a ferramenta de administração do JMS.
JMS 2.0 JMSAdmin [1]	Executa a ferramenta de administração do IBM MQ JMS, conforme descrito em Iniciando a ferramenta de administração.
JMS 2.0 JMSAdmin.config	O arquivo de configuração da ferramenta de administração do IBM MQ JMS, conforme descrito em Configurando a ferramenta de administração do JMS.
PSIVTRun [1]	Executa o programa de teste de verificação da instalação de publicação/assinatura, conforme descrito em "O IVT de publicar/assinar para IBM MQ classes for JMS" na página 117.
PSReportDump.class	Essa classe é mantida para compatibilidade com liberações anteriores, mas não executa nenhuma função.
JM 3.0 setjms30env "2" na página 126	Linux AIX Para Jakarta Messaging 3.0, configura as variáveis de ambiente para executar um aplicativo IBM MQ classes for JMS em uma máquina virtual (JVM) Java de 32 bits em sistemas AIX and Linux , conforme descrito em "Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging" na página 95.

Tabela 14. Scripts fornecidos com IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging (continuação)	
Utilitário	**Uso**
JMS 2.0 setjmsenv "2" na página 126	Linux AIX Para JMS 2.0, configura as variáveis de ambiente para executar um aplicativo IBM MQ classes for JMS em uma máquina virtual (JVM) Java de 32 bits em sistemas AIX and Linux , conforme descrito em "Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging" na página 95.
JM 3.0 setjms30env64 "2" na página 126	Linux AIX Para Jakarta Messaging 3.0, configura as variáveis de ambientes para executar um aplicativo IBM MQ classes for JMS em um JVM de 64 bits em sistemas AIX and Linux , conforme descrito em "Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging" na página 95.
JMS 2.0 setjmsenv64 "2" na página 126	Linux AIX Para JMS 2.0, configura as variáveis de ambientes para executar um aplicativo IBM MQ classes for JMS em um JVM de 64 bits em sistemas AIX and Linux , conforme descrito em "Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging" na página 95.

Nota:

1. No Windows, o nome do arquivo tem a extensão `.bat`
2. Esses scripts estão disponíveis apenas no AIX and Linux No Windows, após a instalação do IBM MQ, execute o comando **setmqenv** Para obter informações adicionais, consulte "Configurando variáveis de ambiente para IBM MQ classes for JMS/Jakarta Messaging" na página 95.

Suporte para OSGi com o IBM MQ classes for JMS

OSGi fornece uma estrutura que suporta a implementação de aplicativos como pacotes configuráveis. Os pacotes configuráveis do OSGi são fornecidos como parte do IBM MQ classes for JMS

O IBM MQ classes for JMS inclui os seguintes pacotes configuráveis do OSGi.

com.ibm.msg.client.osgi.jms*version_number*.jar
A camada comum de código no IBM MQ classes for JMS. Para obter informações sobre a arquitetura em camadas de classes IBM MQ para JMS, consulte as classes do IBM MQ para arquitetura JMS.

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
O pré-requisito do archive Java (JAR) para a camada comum.

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Serviços comuns para aplicativos Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_*version_number*.jar
Mensagens para a camada comum.

com.ibm.msg.client.osgi.wmq_*version_number*.jar
O provedor de sistemas de mensagens do IBM MQ no IBM MQ classes for JMS. Para obter informações sobre a arquitetura em camadas de IBM MQ classes for JMS, consulte as classes do IBM MQ para arquitetura JMS.

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
Os arquivos JAR de pré-requisito para o provedor de sistemas de mensagens do IBM MQ.

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
Mensagens para o provedor de sistemas de mensagens do IBM MQ.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar

JM 3.0 Para Jakarta Messaging 3.0, esse arquivo JAR permite que os aplicativos usem o IBM MQ classes for JMS e o IBM MQ classes for Javae também inclui o código para manipular mensagens PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar

JM 3.0 Para Jakarta Messaging 3.0, esse arquivo JAR fornece os pré-requisitos para `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclient_*version_number*.jar

JMS 2.0 Para JMS 2.0, esse arquivo JAR permite que aplicativos usem o IBM MQ classes for JMS e o IBM MQ classes for Javae também inclui o código para manipular mensagens PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar

JMS 2.0 Para JMS 2.0, esse arquivo JAR fornece os pré-requisitos para `com.ibm.mq.osgi.allclient_version_number.jar`.

em que *version_number* é o número da versão de IBM MQ que está instalado

Os bundles são instalados no subdiretório `java/lib/OSGi` da sua instalação IBM MQ, ou a pasta `java\lib\OSGi` no Windows.

A partir de IBM MQ 8.0, use os bundles `com.ibm.mq.osgi.allclient_8.0.0.0.jar`, e `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` para quaisquer novos aplicativos. O uso desses pacotes configuráveis remove a restrição de não ser capaz de executar ambos IBM MQ classes for JMS e o IBM MQ classes for Java dentro da mesma estrutura OSGi, no entanto, todas as outras restrições ainda se aplicam.

O bundle `com.ibm.mq.osgi.java`*version_number*`.jar`, que também é instalado no subdiretório `java/lib/OSGi` da sua instalação IBM MQ, ou a pasta `java\lib\OSGi` no Windows, faz parte do IBM MQ classes for Java. Este pacote configurável não deve ser carregado em um ambiente de tempo de execução do OSGi que tem o IBM MQ classes for JMS carregado.

Os pacotes configuráveis OSGi para o IBM MQ classes for JMS foram gravados na especificação da liberação 4 do OSGi. Eles não funcionam em um ambiente de liberação 3 do OSGi.

Deve-se configurar seu caminho de sistema ou caminho da biblioteca corretamente para que o ambiente de tempo de execução do OSGi possa localizar quaisquer arquivos da DLL ou bibliotecas compartilhadas requeridas.

Se você usar os pacotes configuráveis do OSGi para o IBM MQ classes for JMS, os tópicos temporários não funcionarão. Além disso, as classes de saída do canal gravadas em Java não são suportadas devido a um problema inerente em classes de carregamento em um ambiente do carregador de classes múltiplo como OSGi. Um pacote configurável do usuário pode ter conhecimento dos pacotes configuráveis do IBM MQ classes for JMS, mas os pacotes configuráveis do IBM MQ classes for JMS não têm conhecimento de qualquer pacote configurável do usuário. Como resultado, o carregador de classes usado em um pacote configurável do IBM MQ classes for JMS não pode carregar uma classe de saída do canal que está em um pacote configurável do usuário.

Para obter mais informações sobre o OSGi, consulte o website do OSGi Alliance.

z/OS MQ Adv. VUE JMS/Jakarta Messaging client connectivity to batch applications running on z/OS

Under certain conditions, an IBM MQ classes for JMS/Jakarta Messaging application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

By using a client connection, an IBM MQ classes for JMS/Jakarta Messaging application can connect to a remote z/OS queue manager if the following conditions apply:

- The application is running in a batch environment.

- O gerenciador de filas que está sendo conectado está em execução com a autorização IBM MQ Advanced for z/OS Value Unit Edition e, portanto, possui o parâmetro **ADVCAP** configurado como ENABLED.

Para obter mais informações sobre o IBM MQ Advanced for z/OS Value Unit Edition , consulte IBM MQ identificadores do produto e informações de exportação

Consulte DISPLAY QMGR para obter mais informações sobre **ADVCAP** e START QMGR para obter mais informações sobre **QMGRPROD**.

Note that batch is the only environment supported; there is no support for JMS/Jakarta Messaging for CICS or JMS/Jakarta Messaging for IMS.

An IBM MQ classes for JMS/Jakarta Messaging application on z/OS cannot use a client mode connection to connect to a queue manager that is not running on z/OS.

If an IBM MQ classes for JMS/Jakarta Messaging application on z/OS attempts to connect using client mode, and is not allowed to do so, exception message JMSFMQ0005 is issued.

Advanced Message Security (AMS) support

IBM MQ classes for JMS/Jakarta Messaging client applications can use AMS when connecting to remote z/OS queue managers, subject to the conditions previously described in this topic.

Para usar AMS dessa maneira, os aplicativos clientes devem usar um tipo de armazenamento de chave `jceracfks` em `keystore.conf`, em que:

- O prefixo do nome da propriedade é `jceracfks` e esse prefixo de nome não faz distinção entre maiúsculas e minúsculas.
- O armazenamento de chaves é um conjunto de chaves RACF.
- As senhas não são necessárias e serão ignoradas. Isso ocorre porque conjuntos de chaves RACF não usam senhas.
- Se você especificar o provedor, ele deverá ser `IBMJCE`.

Quando você usa `jceracfks` com o AMS, o armazenamento de chaves deve estar no formato: `safkeyring://`*`user`*`/`*`keyring`*, em que:

- `safkeyring` é um literal e esse nome não faz distinção entre maiúsculas e minúsculas
- *`user`* é o ID do usuário do RACF que possui o conjunto de chaves
- *`keyring`* é o nome do conjunto de chaves RACF e o nome do conjunto de chaves faz distinção entre maiúsculas e minúsculas

O exemplo a seguir usa o conjunto de chaves padrão AMS para o usuário `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Related concepts

“Java client connectivity to batch applications running on z/OS” on page 377
Under certain conditions, an IBM MQ classes for Java application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

Obtendo o IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging separadamente

As bibliotecas e os utilitários necessários para executar aplicativos usando IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging estão disponíveis em um arquivo JAR autoextrator que você faz download a partir do Fix Central. Faça isso se desejar obter apenas esses arquivos, por exemplo, para implementação em uma ferramenta de gerenciamento de software ou para uso com aplicativos clientes independentes.

Antes de começar

Antes de iniciar essa tarefa, certifique-se de ter um Java runtime environment (JRE) instalado em sua máquina e que o JRE foi incluído no caminho do sistema.

O instalador do Java que é usado nesse processo de instalação não requer que seja executado como raiz ou qualquer usuário específico. O único requisito é que o usuário no qual ele é executado tenha acesso de gravação ao diretório no qual você deseja que os arquivos sejam enviados.

JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando IBM MQ classes para JMS/Jakarta Messaging.

Sobre esta tarefa

Um arquivo JAR autoextrator é usado para minimizar o tamanho do download e o tempo necessário para executar a extração. O conteúdo exato desse arquivo JAR e os subdiretórios nos quais ele extrai os arquivos dependem da versão de IBM MQ.

Ao executar o arquivo JAR de extração automática, ele exibe o contrato de licença do IBM MQ , que deve ser aceito Ele também permite alterar o diretório-pai para a extração.

Para IBM MQ 9.3 e mais recente, o arquivo JAR autoextrator extrai os arquivos na estrutura de diretório a seguir:

wmq/JavaEE
Os arquivos EAR e RAR do adaptador de recursos IBM MQ .

JMS 2.0 Os seguintes arquivos são para uso com objetos JMS 2.0 e JMS 1.1 :

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 Há também um conjunto equivalente para uso com objetos Jakarta Messaging 3.0 :

- `wmq.jakarta.jmsra.ivt.ear`. Contém os arquivos de Teste de Verificação da instalação
- `wmq.jakarta.jmsra.rar`. Contém os arquivos do adaptador de recurso

wmq/JavaSE

wmq/JavaSE/bin
As ferramentas **JMSAdmin** e **JMS30Admin** . Usado para definir entidades do JNDI que representam objetos JMS ou Jakarta Messaging

JMS 2.0 Os seguintes arquivos são para uso com objetos JMS 2.0 e JMS 1.1 :

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 Há também um conjunto equivalente para uso com objetos Jakarta Messaging 3.0 :

- `JMS30Admin.bat`. Um arquivo usado para iniciar a ferramenta no Windows.
- `JMS30Admin`. Um script usado para iniciar a ferramenta nas plataformas Linux e UNIX .
- `JMS30Admin.config`. Um arquivo de configuração de amostra para a ferramenta

Nota:

- Antes da ferramenta **JMSAdmin** ser incluída no arquivo JAR autoextrator, os arquivos nesse diretório estavam no diretório-pai `wmq/JavaSE`.

- Um cliente que é instalado usando o arquivo JAR autoextrator pode usar a ferramenta **JMSAdmin** ou **JMS30Admin** para criar Java objetos administrados pelo sistema de mensagens dentro de um contexto do sistema de arquivos (arquivo `.bindings`). O cliente também pode consultar e usar esses objetos administrados.
- JMS 2.0 A ferramenta **JMSAdmin** para uso com os objetos JMS 2.0 e JMS 1.1 foi incluída no arquivo JAR autoextrator em IBM MQ 9.2.0 Fix Pack 2 e IBM MQ 9.2.2
- JM 3.0 A ferramenta **JMS30Admin** para uso com objetos Jakarta Messaging 3.0 foi incluída no arquivo JAR de autoextração em IBM MQ 9.3.0

wmq/JavaSE/lib

O Advanced Message Security usa os pacotes Bouncy Castle de software livre a seguir para suportar a sintaxe de mensagem criptográfica (CMS). Consulte o Suporte para JREs nãoIBM com AMS

V 9.4.0 De IBM MQ 9.4.0:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

Cada um dos seguintes arquivos contém as classes para seu nível JMS ou Jakarta Messaging específico:

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 e JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

Outros arquivos JAR de pré-requisito:

- `fscontext.jar`. Necessário se o seu aplicativo executar consultas JNDI usando um contexto do sistema de arquivos
- JM 3.0 `jakarta.jms-api.jar`. Contém a interface Jakarta Messaging 3.0 e definições de Exceção.
- JMS 2.0 `jms.jar`. Contém a interface JMS 2.0 e definições de Exceção.
- `org.json.jar`. Contém classes que permitem que o IBM MQ classes for JMS interprete os arquivos CCDT no formato JSON
- `providerutil.jar`. Necessário se o seu aplicativo executar consultas JNDI usando um contexto do sistema de arquivos

Nota: Stabilized `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar` contêm uma cópia do IBM MQ classes for Java. No entanto, em IBM MQ 9.0, essas classes são declaradas como estabilizadas funcionalmente no nível fornecido em IBM MQ 8.0 Consulte Descontinuações, estabilizações e remoções em IBM MQ 9.0..

wmq/OSGi

Os pacotes configuráveis do cliente OSGi IBM MQ :

- JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`
- JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

em que *V.R.M.F* é o número de Versão, Liberação, Modificação e Fix Pack.

Procedimento

1. Faça download do arquivo JAR do cliente IBM MQ Java/JMS da Fix Central.
   a) Clique neste link: IBM MQ Java / JMS cliente.
   b) Localize o cliente para a sua versão do IBM MQ na lista de correções disponíveis que é exibida.
      Por exemplo:

      ```
      release level: 9.3.0.0-IBM-MQ-Install-Java-All
      Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
      ```

      Em seguida, clique no nome do arquivo do cliente e siga o processo de download.
2. Inicie a extração do diretório para o qual você transferiu por download o arquivo.
   Para iniciar a extração, insira um comando no seguinte formato:

   ```
   java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
   ```

   em que *V.R.M.F* é o número da versão do produto, por exemplo, `9.3.0.0`e *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` é o nome do arquivo que foi transferido por download de Fix Central.
   Por exemplo, para extrair o cliente JMS para a liberação IBM MQ 9.4.0 , você usaria o comando a seguir:

   ```
   java -jar 9.4.0.0-IBM-MQ-Install-Java-All.jar
   ```

   Nota: Para realizar essa instalação, deve-se ter um JRE instalado em sua máquina e incluído no caminho do sistema.

   Ao inserir o comando, as seguintes informações são exibidas:

   ```
   Antes de poder usar, extrair ou instalar o IBM MQ V9.4, deve-se aceitar
   os termos de 1. IBM Contrato Internacional de Licença para Avaliação de
   Programas 2. Contrato Internacional de Licença do Programa IBM e adicional
   . Leia os contratos de licença a seguir com atenção.

   O contrato de licença é exibido separadamente usando a opção
   --viewLicenseAgreement.

   Pressione Enter para exibir os termos de licença agora, ou 'x' para ignorar.
   ```
3. Revise e aceite os termos de licença:
   a) Para exibir a licença, pressione Enter.
      Como alternativa, pressione x para pular a exibição da licença.
      Após a licença ser exibida, ou imediatamente se você pressionar x, a mensagem a seguir será exibida:

      ```
      As informações adicionais sobre licenças são exibidas separadamente usando a opção
      --viewLicenseInfo.

      Pressione Enter para exibir informações adicionais sobre licenças agora ou 'x' para ignorar.
      ```
   b) Para exibir os termos de licença adicionais, pressione Enter.
      Como alternativa, pressione x para ignorar a exibição dos termos de licença adicionais.
      Após os termos de licença adicionais serem exibidos, ou imediatamente se você pressionar x, a mensagem a seguir será exibida:

      ```
      Ao escolher a opção "Concordo" abaixo, você concorda com os termos do
      contrato de licença e com os termos não IBM, se aplicável. Se você não
      concordar, selecione "Eu não concordo".

      Selecione [1] Eu concordo ou [2] Eu não concordo:
      ```
   c) Para aceitar o contrato de licença e continuar selecionando o diretório de instalação, selecione 1.
      Como alternativa, selecione 2 para terminar a instalação imediatamente..
      Se você selecionar 1, uma mensagem semelhante à seguinte será exibida:

      ```
      Insira um diretório para arquivos do produto ou mantenha em branco para aceitar o valor
      padrão.
      ```

```
O diretório de destino padrão é H: \downloads

Diretório de destino para arquivos do produto?
```

4. Especifique o diretório-pai para a extração

   O local padrão é o diretório atual.

   - Se desejar extrair os arquivos do produto no local padrão, pressione Enter sem especificar um valor.
   - Se você desejar extrair os arquivos do produto para um local diferente, especifique o nome do diretório no qual deseja extrair os arquivos e, em seguida, pressione Enter para iniciar a extração

   O nome do diretório especificado ainda não deve existir, caso contrário, quando você iniciar a extração, um erro será relatado e nenhum arquivo será instalado.

   Desde que ele ainda não exista, o diretório especificado é criado e os arquivos de programa são extraídos para esse diretório Durante a instalação, um novo diretório com o nome `wmq` é criado dentro do diretório-pai especificado.

   Três sub-diretórios, `JavaEE`, `JavaSE` e `OSGi`, são criados no diretório `wmq` com os conteúdos a seguir:

   JavaEE

   JM 3.0 `wmq.jakarta.jmsra.ivt.ear`

   JM 3.0 `wmq.jakarta.jmsra.rar`

   JMS 2.0 `wmq.jmsra.ivt.ear`

   JMS 2.0 `wmq.jmsra.rar`

   JavaSE

   Esse diretório contém os seguintes subdiretórios e arquivos:

   JavaSE/lib

   V 9.4.0 `bcpkix-jdk18on.jar`

   V 9.4.0 `bcprov-jdk18on.jar`

   V 9.4.0 `bcutil-jdk18on.jar`

   JMS 2.0 `com.ibm.mq.allclient.jar`

   JM 3.0 `com.ibm.mq.jakarta.client.jar`

   `fscontext.jar`

   `jms.jar`

   `org.json.jar`

   `providerutil.jar`

   JavaSE/bin

   `JMSAdmin.bat`

   `JMSAdmin`

   `JMSAdmin.config`

   OSGi

   JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`

   JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`

   JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`

   JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

Quando a extração for concluída, uma mensagem de confirmação será exibida conforme mostrado no exemplo a seguir:

```
Extraindo arquivos para H: \downloads\wmq
Todos os arquivos do produto foram extraídos com sucesso.
```

Lista de permissões em IBM MQ classes for JMS/Jakarta Messaging

O mecanismo de serialização e desserialização do objeto Java foi identificado como um risco de segurança em potencial A lista de permissões em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging fornece alguma proteção contra alguns riscos de serialização.

Sobre esta tarefa

O mecanismo de serialização e desserialização de objeto do Java foi identificado como um risco de segurança em potencial porque a desserialização instancia objetos arbitrários do Java, em que há muita possibilidade de dados serem enviados intencionalmente para causar vários problemas. Uma aplicação notável de serialização está em Jakarta Messaging 3.0 e Java Message Service 2.0 ObjectMessages que usam a serialização para encapsular e transferir objetos arbitrários

A listagem de permissões de serialização é uma mitigação em potencial em relação a alguns dos riscos que a serialização representa. Ao especificar explicitamente quais classes podem ser encapsuladas no ObjectMessages e extraídas dele, a listagem de permissões fornece alguma proteção com relação a alguns riscos de serialização.

Conceitos relacionados

“Executando aplicativos IBM MQ classes for JMS no Java security manager” na página 108
O IBM MQ classes for JMS pode executar com o Java security manager ativado. Para executar aplicativos com sucesso com o Java security manager ativado, deve-se configurar o Java Virtual Machine (JVM) com um arquivo de configuração de política adequado.

Conceitos de listagem de permissões

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, há suporte para a lista de permissões de classes na implementação da interface JMS ObjectMessage . Isso fornece uma mitigação em potencial com relação a alguns dos riscos de segurança potencialmente relacionados ao mecanismo de serialização e desserialização do objeto Java .

Listagem de permissões em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Importante:

Sempre que possível, o termo *lista de permissões* substitui o termo *lista de desbloqueio*. Isso inclui alguns nomes de propriedade de sistema Java mencionados neste tópico Não é necessário mudar nenhuma configuração existente. Os nomes de propriedades do sistema anteriores também continuam funcionando.

IBM MQ classes for JMS (JMS 2.0) e IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) suportam a lista de permissões de classes na implementação da interface JMS ObjectMessage .

- JMS 2.0 Para IBM MQ classes for JMS, os nomes de propriedades relevantes são **com.ibm.mq.jms.allowlist.***..
- JM 3.0 Para IBM MQ classes for Jakarta Messaging, os nomes de propriedades relevantes são **com.ibm.mq.jakarta.jms.allowlist.***

A lista de permissões define quais classes do Java podem ser serializadas com ObjectMessage.setObject() e desserializadas com ObjectMessage.getObject().

- JMS 2.0 As tentativas de serializar ou desserializar uma instância de uma classe não incluída na lista de permissões com o ObjectMessage fazem com que um javax.jms.MessageFormatException seja lançado, com um java.io.InvalidClassException como sua causa.

- JM 3.0 Tentativas de serializar ou desserializar uma instância de uma classe não incluída na lista de permissões com ObjectMessage fazem com que um jakarta.jms.MessageFormatException seja lançado, com um java.io.InvalidClassException como sua causa.

Produzindo a lista de permissões

Importante: IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging não podem ser distribuídos com uma lista de permissões. A opção das classes a serem transferidas usando ObjectMessages é uma opção de design de aplicativo e o IBM MQ não pode priorizar isso.

Por essa razão, o mecanismo de listagem de permissões permite dois modos de operação:

DISCOVERY
Nesse modo, o mecanismo produz uma listagem de nomes completos de classe, relatando todas as classes que foram observadas para serem serializadas ou desserializadas em ObjectMessages.

ENFORCEMENT
Nesse modo, o mecanismo impõe a listagem de permissões, rejeitando as tentativas de serializar ou desserializar as classes que não estão na lista de permissões.

Para usar esse mecanismo, a execução deverá ser feita inicialmente no modo de DESCOBERTA para reunir a lista de classes atualmente serializadas e desserializadas, revisá-la e usá-la como base para a sua lista de permissões. Pode até ser apropriado usar a lista inalterada, mas a lista deve ser revisada primeiro, antes de decidir fazer isso.

Controlando o mecanismo de listagem de permissões

Três propriedades do sistema estão disponíveis para controlar o mecanismo de listagem de permissões:

com.ibm.mq.jms.allowlist (JMS 2.0) e com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
Essa propriedade pode ser especificada de uma das seguintes maneiras:

- O nome do caminho do arquivo que contém a lista de permissões em formato de URI de arquivo (ou seja, começando com `file:`). No modo de DESCOBERTA, esse arquivo é gravado pelo mecanismo de listagem de permissões. O arquivo não deve existir. Se o arquivo existir, o mecanismo lançará uma exceção, em vez de sobrescrevê-lo. No modo de CUMPRIMENTO, esse arquivo é lido pelo mecanismo de listagem de permissões.
- Uma lista separada por vírgula de nomes completos de classe que constituem a lista de permissões.

Se essa propriedade estiver desconfigurada, o mecanismo da lista de permissões estará inativo.

Se você estiver usando um Java security manager, deverá assegurar que os arquivos JAR do IBM MQ classes for JMS tenham acesso de leitura e gravação a esse arquivo.

com.ibm.mq.jms.allowlist.discover (JMS 2.0) e com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- Se essa propriedade for desconfigurada ou configurada como false, o mecanismo da lista de permissões será executado no modo de CUMPRIMENTO.
- Se essa propriedade for configurada como true e a lista de permissões tiver sido especificada como um URI de arquivo, o mecanismo da lista de permissões será executado no modo de DESCOBERTA.
- Se essa propriedade for configurada como true e a lista de permissões tiver sido especificada como uma lista de nomes de classes, o mecanismo da lista de permissões lançará uma exceção adequada.
- Se essa propriedade for configurada como true, e a lista de permissões não tiver sido especificada usando a propriedade com.ibm.mq.jms.allowlist ou com.ibm.mq.jakarta.jms.allowlist, o mecanismo da lista de permissões estará inativo
- Se essa propriedade for configurada como true e o arquivo da lista de permissões já existir, o mecanismo da lista de permissões lançará uma java.io.InvalidClassException e as entradas não serão incluídas no arquivo.

com.ibm.mq.jms.allowlist.mode (JMS 2.0) e com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)

Essa propriedade de sequência pode ser especificada em qualquer uma de três maneiras:

- Se essa propriedade for configurada como SERIALIZE, o modo de CUMPRIMENTO realizará a validação da lista de permissões apenas no método ObjectMessage.setObject().
- Se essa propriedade for configurada como DESERIALIZE, o modo de CUMPRIMENTO realizará a validação da lista de permissões apenas no método ObjectMessage.getObject().
- Se essa propriedade for desconfigurada ou configurada como qualquer outro valor, o modo de CUMPRIMENTO executará a validação da lista de permissões em ambos os métodos ObjectMessage.getObject() e ObjectMessage.setObject().

Formato do arquivo da lista de permissões

Estes são os principais recursos do formato do arquivo da lista de permissões:

- O arquivo de lista de permissões está em codificação de arquivo de plataforma padrão com finais de linha apropriados para a plataforma.

  Nota: Se um arquivo da lista de permissões estiver sendo usado, esse arquivo será sempre gravado e lido usando a codificação de arquivo padrão para a JVM.

  Não ocorrerão problemas se o arquivo da lista de permissões for gerado de qualquer uma das maneiras a seguir:

  - z/OS Gerado por um aplicativo independente em execução no z/OS e usado por outros aplicativos independentes que também estão em execução no z/OS.
  - Gerado por um aplicativo em execução dentro do WebSphere Application Server em qualquer plataforma e usado por outra instância do WebSphere Application Server.
  - Multi Gerado por um aplicativo independente em execução no IBM MQ for Multiplatforms e usado por outros aplicativos independentes em execução no IBM MQ for Multiplatforms ou por aplicativos em execução no WebSphere Application Server em qualquer plataforma.

  No entanto, como o WebSphere Application Server usa ASCII, e uma JVM independente usa EBCDIC, haverá problemas de codificação de arquivo se o arquivo da lista de permissões for gerado de uma das maneiras a seguir:

  - Gerado no z/OS; em seguida, usado por aplicativos independentes em execução em uma plataforma diferente do z/OS ou pelo WebSphere Application Server.
  - Gerado pelo WebSphere Application Server ou um aplicativo independente em execução em uma plataforma diferente do z/OS; em seguida, usado por um aplicativo independente no z/OS.
- Cada linha não vazia contém um nome completo de classe. As linhas vazias são ignoradas.
- Comentários podem ser incluídos - qualquer coisa após um caractere '#' até o final da linha é ignorada.
- Há um mecanismo curinga muito básico:
  - '*' pode ser o **último** elemento de um nome de classe.
  - '*' corresponde a um **único** elemento de um nome de classe, ou seja, a classe, mas nenhuma parte do pacote.

  Portanto, `com.ibm.mq.*` corresponderia a `com.ibm.mq.MQMessage`, mas não a `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  O curinga não funciona para classes no pacote padrão, ou seja, para classes sem um nome de pacote explícito, por isso, um nome de classe de "*" é rejeitado.
- Arquivos de lista de permissões mal formatados, por exemplo, arquivos que contêm uma entrada como `com.ibm.mq.*.Message`, em que o curinga não é o último elemento, resultam no lançamento de uma java.lang.IllegalArgumentException.
- Um arquivo da lista de permissões vazio tem o efeito de desativar totalmente o uso do ObjectMessage.

Formato da lista de permissões como uma lista separada por vírgula

O mesmo mecanismo de curinga está disponível para uma lista de permissões como uma lista separada por vírgula.

- O '*' poderá ser expandido pelo sistema operacional se especificado em uma linha de comandos, shell script ou arquivo em lote, por isso, ele poderá precisar de manipulação especial.
- O caractere de comentário '#' é aplicável apenas quando um arquivo é especificado. Se a lista de permissões for especificada como uma lista separada por vírgula de nomes de classes, supondo que o sistema operacional ou shell não o processe, pois ele é o caractere de comentário padrão em muitos shells AIX and Linux , ele será tratado como um caractere normal.

Quando a listagem de permissões acontece?

A listagem de permissões é iniciada quando o aplicativo executa primeiro um método ObjectMessage setMessage() ou getMessage().

As propriedades do sistema são avaliadas, o arquivo da lista de permissões é aberto e no modo de CUMPRIMENTO, a lista de classes da lista de permissões é carregada quando o mecanismo é inicializado. Nesse ponto, uma entrada é gravada no arquivo de log do IBM MQ JMS para o aplicativo.

Quando o mecanismo é inicializado, seus parâmetros podem não ser mudados. O horário da inicialização não é facilmente previsto, pois ele depende do comportamento do aplicativo. As configurações de propriedade do sistema e os conteúdos do arquivo da lista de permissões devem, portanto, ser considerados como fixos a partir do momento em que o aplicativo é iniciado. Não mude as propriedades ou os conteúdos do arquivo da lista de permissões enquanto o aplicativo estiver em execução, pois os resultados não são garantidos.

Pontos a serem considerados

A melhor abordagem para minimizar os riscos intrínsecos ao mecanismo de serialização do Java seria explorar abordagens alternativas para transferência de dados, como usar JSON, em vez de ObjectMessage. Usar os mecanismos do Advanced Message Security (AMS) pode incluir segurança adicional, assegurando que as mensagens venham de fontes confiáveis.

Se você usar o mecanismo do Java security manager com seu aplicativo, deverá conceder as seguintes permissões:

- FilePermission em qualquer arquivo da lista de permissões que você usa, com permissão de leitura para o modo de CUMPRIMENTO, permissão de gravação para o modo de DESCOBERTA.
- JMS 2.0 ProprityPermission (leitura) nas propriedades **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover** e **com.ibm.mq.jms.allowlist.mode**.
- JM 3.0 ProprityPermission (leitura) nas propriedades **com.ibm.mq.jakarta.jms.allowlist**, **com.ibm.mq.jakarta.jms.allowlist.discover** e **com.ibm.mq.jakarta.jms.allowlist.mode**.

Mais informações

Consulte “Configurando e usando uma lista de permissões JMS ou Jakarta Messaging” na página 137 e “Listagem de permissões no WebSphere Application Server” na página 139 para obter mais informações sobre listas de permissões.

Conceitos relacionados

“Executando aplicativos IBM MQ classes for JMS no Java security manager” na página 108
O IBM MQ classes for JMS pode executar com o Java security manager ativado. Para executar aplicativos com sucesso com o Java security manager ativado, deve-se configurar o Java Virtual Machine (JVM) com um arquivo de configuração de política adequado.

Configurando e usando uma lista de permissões JMS ou Jakarta Messaging

Essas informações informam como uma lista de permissões funciona e como você configura uma usando a funcionalidade contida no IBM MQ classes for JMS ou no IBM MQ classes for Jakarta Messaging para gerar um arquivo da lista de permissões, contendo uma lista dos tipos de ObjectMessages que um aplicativo pode processar

Antes de começar

Importante:

Sempre que possível, o termo *lista de permissões* substitui o termo *lista de desbloqueio*. Isso inclui alguns nomes de propriedade de sistema Java mencionados neste tópico Não é necessário mudar nenhuma configuração existente. Os nomes de propriedades do sistema anteriores também continuam funcionando.

Antes de iniciar essa tarefa, certifique-se de que tenha lido e entendido "Conceitos de listagem de permissões" na página 133

Sobre esta tarefa

Como JMS e Jakarta Messaging compartilham muito em comum, referências adicionais a JMS neste tópico podem ser consideradas referentes a ambos. Quaisquer diferenças são destacadas conforme necessário.

Quando você tiver ativado a funcionalidade da listagem de permissões, o IBM MQ classes for JMS usará essa funcionalidade das maneiras a seguir:

- Quando um aplicativo quiser enviar um ObjectMessage, ele poderá criá-lo de uma de duas maneiras, chamando o:
  - Método Session.createObjectMessage(Serializable), passando o objeto que deve estar contido na mensagem.
  - Método Session.createObjectMessage(), para criar um ObjectMessage vazio e, em seguida, chamando ObjectMessage.setObject(Serializable) para armazenar o objeto a ser enviado no ObjectMessage.

  Quando os métodos Session.createObjectMessage(Serializable) ou ObjectMessage.setObject(Serializable) são chamados, as classes para o JMS verificam se o objeto transmitido é de um tipo mencionado na lista de permissões.

  Se for de um tipo mencionado, o objeto será serializado e armazenado no ObjectMessage. No entanto, se o objeto for de um tipo que não esteja na lista de permissões, o IBM MQ classes for JMS lançará uma JMSException contendo a mensagem:

  ```
  JMSCC0052: ocorreu uma exceção ao serializar o objeto:
  'java.io.InvalidClassException: <object class>; A classe não pode ser serializada
  ou desserializada, pois não foi incluída na lista de permissões '<allowlist>'.
  ```

  de volta para o aplicativo.

  Importante: Se a exceção for lançada por meio do método Session.createObjectMessage(Serializable), o ObjectMessage não será criado. Da mesma forma, se a JMSException for lançada por meio do método ObjectMessage.setObject(Serializable), o objeto não será incluído no ObjectMessage.
- Se um aplicativo receber um ObjectMessage, ele chamará o método ObjectMessage.getObject() para obter o objeto contido nele. Quando este método é chamado, o IBM MQ classes for JMS verifica o tipo de objeto contido no ObjectMessage, para ver se esse objeto é de um tipo especificado na lista de permissões.

  Se for, o objeto será desserializado e retornado para o aplicativo. No entanto, se o objeto for de um tipo que não esteja na lista de permissões, o IBM MQ classes for JMS lançará uma JMSException contendo a mensagem:

  ```
  JMSCC0053: ocorreu uma exceção ao desserializar uma mensagem:
  'java.io.InvalidClassException: <object class>; A classe poderá não ser
  ```

```
serializada ou desserializada porque não está incluída na
lista de permissões '< allowlist>'. '.
```

de volta para o aplicativo.

Por exemplo, suponha que o seu aplicativo contenha o código a seguir para enviar um ObjectMessage contendo um objeto do tipo java.net.URI:

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

Como a listagem de permissões não está ativada, o aplicativo é capaz de colocar a mensagem com sucesso no destino necessário.

Se você criar um arquivo chamado `C:\allowlist.txt` contendo uma única entrada, `java.net.URL`, e iniciar o aplicativo novamente com o conjunto de propriedades de sistema Java :

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

a funcionalidade da lista de permissões será ativada. O aplicativo ainda é capaz de criar e enviar o ObjectMessage contendo um objeto do tipo java.net.URI, uma vez que esse tipo é especificado na lista de permissões.

No entanto, se você mudar o arquivo `allowlist.txt` para que o arquivo contenha a entrada única `java.util.Calendar`, uma vez que a funcionalidade da lista de permissões ainda estará ativada, quando o aplicativo chamar:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

o IBM MQ classes for JMS verificará a lista de permissões e descobrirá que ela não contém uma entrada para java.net.URI.

Como resultado, uma JMSException contendo a mensagem JMSCC0052 é lançada.

Da mesma forma, suponha que você tenha outro aplicativo que receba ObjectMessages usando este código:

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

Se a listagem de permissões não estiver ativada, o aplicativo será capaz de receber ObjectMessages que contêm um objeto de qualquer tipo. O aplicativo verificará então se o objeto é do tipo java.net.URL antes de executar o processamento apropriado.

Se agora você iniciar o aplicativo com a propriedade de sistema Java:

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

configurada, a funcionalidade da listagem de permissões será ativada. Quando o aplicativo chamar:

```
Object messageBody = objectMessage.getObject();
```

o método ObjectMessage.getObject() retornará apenas objetos do tipo java.net.URL.

Se o objeto contido no ObjectMessage não for desse tipo, o método ObjectMessage.getObject() lançará uma JMSException contendo a mensagem JMSCC0053. O aplicativo precisará, então, decidir o que fazer com a mensagem, por exemplo, a mensagem poderia ser movida para a fila de mensagens não entregues desse gerenciador de filas.

O aplicativo retornará normalmente apenas se o objeto no ObjectMessage for do tipo java.net.URL.

Procedimento

1. Execute o aplicativo que processa ObjectMessages, com as seguintes propriedades de sistema Java especificadas:

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Quando o aplicativo é executado, o IBM MQ classes for JMS cria um arquivo que contém os tipos de objetos processados pelo aplicativo.
2. Depois que o aplicativo processar uma amostra representativa de ObjectMessages durante um período de tempo, pare-o.

   O arquivo da lista de permissões agora contém uma lista de todos os tipos de objetos contidos no ObjectMessages que o aplicativo processou enquanto estava em execução.

   Se você tiver executado o aplicativo por um tempo suficiente, essa lista incluirá todos os tipos possíveis de objetos contidos nos ObjectMessages que provavelmente o aplicativo manipulará.
3. Reinicie o aplicativo com a propriedade de sistema a seguir configurada:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Isso permite a listagem de permissões e, se o IBM MQ classes for JMS detectar um ObjectMessage de um tipo que não esteja na lista de permissões, o lançamento de uma JMSException contendo a mensagem JMSCC0052 ou JMSCC0053.

Listagem de permissões no WebSphere Application Server

Como usar a listagem de permissões do IBM MQ classes for JMS no WebSphere Application Server.

Importante:

Sempre que possível, o termo *lista de permissões* substitui o termo *lista de desbloqueio*. Isso inclui alguns nomes de propriedade de sistema Java mencionados neste tópico Não é necessário mudar nenhuma configuração existente. Os nomes de propriedades do sistema anteriores também continuam funcionando.

Deve-se garantir que a instalação do WebSphere Application Server inclua uma versão do adaptador de recursos do IBM MQ que suporte a listagem de permissões.

Consulte “Usando o IBM MQ e o WebSphere Application Server juntos” na página 509 para obter informações adicionais sobre o uso dos dois produtos.

O IBM MQ 9.0.0 Fix Pack 1 em diante inclui a funcionalidade apropriada.

Depois que o servidor de aplicativos tiver sido atualizado, será possível usar as propriedades de sistema do Java:

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

descritas em “Configurando e usando uma lista de permissões JMS ou Jakarta Messaging” na página 137.

Nota: Será necessário configurar as propriedades de sistema do Java como argumentos genéricos da JVM, na Java Virtual Machine usada para executar o servidor de aplicativos e o servidor de aplicativos reiniciado para que as mudanças entrem em vigor.

Consulte a seção em *Argumentos genéricos da JVM* em Configurações de máquina virtual Java para obter mais informações.

Para configurar as propriedades, acesse a janela Java Virtual Machine em *Definições de processo* e insira o argumento apropriado.

A configuração a seguir:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

faz com que o servidor de aplicativos use a lista de permissões *youruserId_MyObject*. Apenas objetos do tipo são processados pelo servidor de aplicativos.

As definições a seguir:

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

configuram o servidor de aplicativos para usar o modo *Descobrir* e registram detalhes do JMS ObjectMensagens, que o servidor de aplicativos processa, para o arquivo `C:\allowlist.txt`

A configuração a seguir:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

faz o servidor de aplicativos carregar o arquivo `C:/allowlist.txt` e usa as informações nesse arquivo para determinar a lista de permissões.

Conceitos relacionados

“Executando aplicativos IBM MQ classes for JMS no Java security manager” na página 108
O IBM MQ classes for JMS pode executar com o Java security manager ativado. Para executar aplicativos com sucesso com o Java security manager ativado, deve-se configurar o Java Virtual Machine (JVM) com um arquivo de configuração de política adequado.

Conversões de sequências de caracteres no IBM MQ classes for JMS

Os IBM MQ classes for JMS usam CharsetEncoders e CharsetDecoders diretamente para conversão de sequência de caracteres. O comportamento padrão para a conversão de sequência de caracteres pode ser configurado com duas propriedades do sistema. A manipulação de mensagens que contêm caracteres não mapeáveis pode ser configurada por meio de propriedades da mensagem para configurar o UnmappableCharacterAction e os bytes de substituição.

Antes do IBM MQ 8.0, as conversões de sequências no IBM MQ classes for JMS foram feitas chamando os métodos `java.nio.charset.Charset.decode(ByteBuffer)` e `Charset.encode(CharBuffer)`.

Ao usar um desses métodos resultará em uma substituição padrão (REPLACE) de dados malformados ou não convertidos. Esse comportamento pode ocultar erros em aplicativos e conduzir a caracteres inesperados, por exemplo ?, em dados traduzidos.

A partir de IBM MQ 8.0, para detectar tais problemas mais cedo e de forma mais eficaz, os IBM MQ classes for JMS usam CharsetEncoders e CharsetDecoders diretamente e configuram o manuseio de dados malformados e intraduzíveis explicitamente. O comportamento padrão é para REPORT tais problemas ao lançar um `MQException` adequado.

Configurando

Converter de UTF-16 (a representação de caracteres usada no Java) para um conjunto de caracteres nativos, como UTF-8, é denominado *encoding*, enquanto converter na direção oposta é denominado *decoding*.

Decodificação assume o comportamento padrão para `CharsetDecoders`, relatando erros ao lançar uma exceção.

Uma configuração é usada para especificar um `java.nio.charset.CodingErrorAction` para controlar a manipulação de erros na codificação e decodificação. Uma outra configuração é usada para controlar o byte de substituição, ou bytes, ao codificar. A sequência de substituição padrão do Java será usada em operações de decodificação.

Configurações UnmappableCharacterAction e bytes de substituição em IBM MQ classes for JMS

A partir de IBM MQ 8.0, as duas propriedades a seguir estão disponíveis para configurar o UnmappableCharacterAction e os bytes de substituição. As definições constantes apropriadas estão em `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_UNMAPPABLE_ACTION
Configura ou obtém o `CodingErrorAction` para aplicar quando um caractere não puder ser mapeado em uma operação de codificação ou decodificação.

Você deve configurar isso como `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` da seguinte forma:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
Configura ou obtém os bytes de substituição a serem aplicados quando um caractere não puder ser mapeado em uma operação de codificação.

A sequência de substituição padrão do Java é usada em operações de decodificação.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

As propriedades `JMS_IBM_UNMAPPABLE_ACTION` e `JMS_IBM_UNMAPPABLE_REPLACEMENT` podem ser configuradas em destinos ou mensagens. Um valor configurado em uma mensagem substitui o valor configurado no destino para o qual a mensagem está sendo enviada.

Observe que `JMS_IBM_UNMAPPABLE_REPLACEMENT` deve ser configurada como um byte único.

Propriedades do sistema para configuração de padrões do sistema

No IBM MQ 8.0, as duas propriedades do sistema Java a seguir estão disponíveis para configurar o comportamento padrão sobre conversão de sequência de caracteres.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Especifica a ação a ser executada para os dados não convertidos na codificação e decodificação. O valor pode ser `REPORT`, `REPLACE` ou `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Configura ou obtém os bytes de substituição para aplicar quando um caractere não puder ser mapeado em uma operação de codificação. A sequência de substituição padrão do Java é usada em operações de decodificação.

Para evitar confusão entre as representações de byte nativo e caractere do Java, é necessário especificar `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` como um número decimal representando o byte de substituição no conjunto de caracteres nativo.

Por exemplo, o valor decimal de ?, como um byte nativo, será 63 se o conjunto de caracteres nativo for baseado em ASCII, como ISO-8859-1, enquanto que será 111, se o conjunto de caracteres nativo for EBCDIC.

Nota: Observe que se um objeto MQMD ou MQMessage tiver os campos **unmappableAction** ou **unMappableReplacement** configurados, os valores desses campos terão precedência sobre as propriedades do sistema Java. Isso permite que os valores especificados pelas propriedades do sistema Java sejam substituídos para cada mensagem, se necessário.

Conceitos relacionados

“Conversões de sequências de caracteres no IBM MQ classes for Java” na página 357
Os IBM MQ classes for Java usam CharsetEncoders e CharsetDecoders diretamente para conversão de sequência de caracteres. O comportamento padrão para a conversão de sequência de caracteres pode ser configurado com duas propriedades do sistema. A manipulação de mensagens que contêm caracteres não mapeáveis pode ser configurada por meio de `com.ibm.mq.MQMD`.

Gravando aplicativos IBM MQ classes for JMS/Jakarta Messaging

Após uma breve introdução ao modelo do JMS , esta seção fornece orientação detalhada sobre como gravar aplicativos IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Sobre esta tarefa

JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando IBM MQ classes para JMS/Jakarta Messaging.

Conceitos relacionados

IBM MQ classes for Jakarta Messaging: uma visão geral

O modelo JMS e Jakarta Messaging

O modelo JMS e Jakarta Messaging define um conjunto de interfaces que os aplicativos Java podem usar para executar operações do sistema de mensagens.. IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são provedores de sistemas de mensagens. Eles definem como os objetos JMS e Jakarta Messaging são relacionados aos conceitos IBM MQ As especificações JMS e Jakarta Messaging esperam que determinados objetos JMS e Jakarta Messaging sejam objetos administradas

JMS 2.0 O IBM MQ 8.0 incluiu suporte para a versão JMS 2.0 do padrão JMS , que introduziu uma API simplificada, enquanto também retém a API clássica do JMS 1.1.

JM 3.0 A partir do IBM MQ 9.3.0, o Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. O IBM MQ 9.3.0 continua a suportar JMS 2.0 para aplicativos existentes. Não é suportado usar a API JMS 2.0 e a API Jakarta Messaging 3.0 no mesmo aplicativo.

Nota: Para Jakarta Messaging 3.0, o controle da especificação JMS move de Oracle para o Java Community Process. No entanto, o Oracle retém o controle do nome "javax", que é usado em outras tecnologias Java que não foram movidas para o Java Community Process. Portanto, enquanto Jakarta Messaging 3.0 é funcionalmente equivalente a JMS 2.0 , há algumas diferenças na nomenclatura:

- O nome oficial da versão 3.0 é Jakarta Messaging em vez de Java Message Service.
- Os nomes de pacotes e constantes são prefixados com `jakarta` em vez de `javax`. Por exemplo, no JMS 2.0 , a conexão inicial com um provedor de sistemas de mensagens é um objeto `javax.jms.Connection` e no Jakarta Messaging 3.0 é um objeto `jakarta.jms.Connection` .

JMS 2.0 Os pacotes javax.jms definem as interfaces JMS e um provedor JMS implementa essas interfaces para um produto de sistema de mensagens específico. O IBM MQ classes for JMS é um provedor do JMS que implementa as interfaces do JMS para o IBM MQ.

JM 3.0 Os pacotes jakarta.jms definem as interfaces Jakarta Messaging e um provedor Jakarta Messaging implementa essas interfaces para um produto de sistema de mensagens específico. O IBM MQ classes for Jakarta Messaging é um provedor do Jakarta Messaging que implementa as interfaces do Jakarta Messaging para o IBM MQ.

Como JMS e Jakarta Messaging compartilham muito em comum, referências adicionais a JMS neste tópico podem ser consideradas referentes a ambos. Quaisquer diferenças são destacadas conforme necessário.

API simplificada

O JMS 2.0 introduziu a API simplificada, enquanto também retém as interfaces específicas do domínio e independentes do domínio do JMS 1.1. A API simplificada reduz o número de objetos que são necessários para enviar e receber mensagens e consiste nas interfaces a seguir:

ConnectionFactory
Um ConnectionFactory é um objeto administrado usado por um cliente JMS para criar um Connection. Essa interface também é usada na API clássica.

JMSContexto
Esse objeto combina os objetos Connection e Session da API clássica. Os objetos JMSContext podem ser criados a partir de outros objetos JMSContext, com a conexão subjacente sendo duplicada.

JMSProdutor
Um JMSProducer é criado por um JMSContext e é usado para enviar mensagens para uma fila ou tópico. O objeto JMSProducer cria objetos que são necessários para enviar a mensagem.

Consumidor do JMS
Um JMSConsumer é criado por um JMSContext e é usado para receber mensagens de um tópico ou uma fila.

A API simplificada tem diversos efeitos:

- O objeto JMSContext sempre inicia automaticamente a conexão subjacente.
- JMSProducers e JMSConsumers agora podem trabalhar diretamente com corpos de mensagens, sem precisarem obter todo o objeto de mensagem, usando o método getBody da Mensagem.
- Propriedades de mensagens podem ser configuradas no objeto JMSProducer, usando encadeamento de método antes de enviar um 'body', um conteúdo de mensagem. O JMSProducer manipulará a criação de todos os objetos necessários para enviar a mensagem. Usando o JMS 2.0, as propriedades podem ser configuradas e uma mensagem pode ser enviada da seguinte forma:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

O JMS 2.0 também introduziu assinaturas compartilhadas em que as mensagens podem ser compartilhadas entre diversos consumidores Todas as assinaturas do JMS 1.1 são tratadas como assinaturas não compartilhadas.

API clássica

A lista a seguir resume as principais interfaces do JMS da API clássica:

Destino
Um destino é para onde um aplicativo envia mensagens ou é uma origem da qual um aplicativo recebe mensagens, ou ambos.

ConnectionFactory
Um objeto ConnectionFactory contém um conjunto de propriedades de configuração para uma conexão. Um aplicativo usa um connection factory para criar uma conexão.

Conexão
Um objeto Connection contém a conexão ativa de um aplicativo com um servidor de sistema de mensagens. Um aplicativo usa uma conexão para criar sessões.

Sessão
Uma sessão é um único contexto encadeado para enviar e receber mensagens. Um aplicativo usa uma sessão para criar mensagens, produtores de mensagens e consumidores de mensagens. Uma sessão é transacionada ou não transacionada.

Mensagem
Um objeto Message contém uma mensagem que um aplicativo envia ou recebe.

MessageProducer
Um aplicativo usa um produtor de mensagem para enviar mensagens para um destino.

MessageConsumer
Um aplicativo usa um consumidor de mensagem para receber mensagens enviadas a um destino.

Figura 9 na página 144 mostra esses objetos e seus relacionamentos.

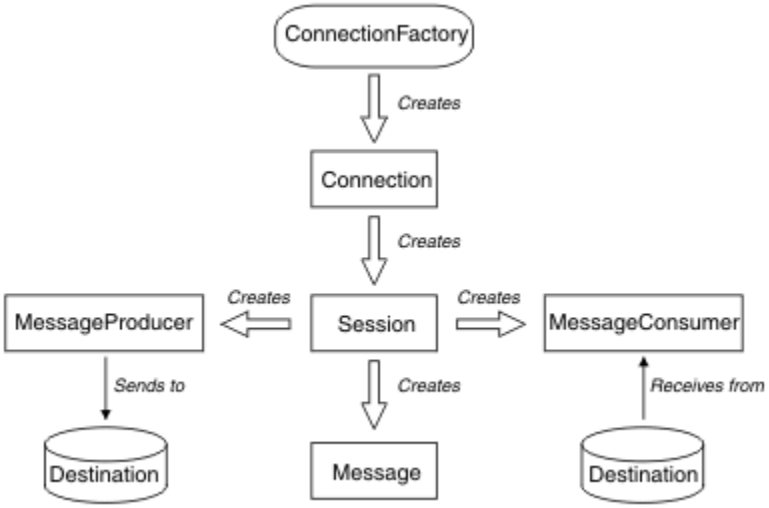


Figura 9. Os objetos do JMS e seus relacionamentos

Um objeto Destination, ConnectionFactory ou Connection pode ser utilizado simultaneamente por diferentes encadeamentos de um aplicativo multiencadeado, mas um objeto Session, MessageProducer ou MessageConsumer não pode ser usado simultaneamente por diferentes encadeamentos. A maneira mais simples de garantir que um objeto Session, MessageProducer ou MessageConsumer não é usado simultaneamente é criar um objeto Session separado para cada encadeamento.

domínios do sistema de mensagens

O JMS suporta dois estilos de sistema de mensagens:

- Sistema de mensagens ponto a ponto
- Sistema de Mensagens de Publicação/Assinatura

Esses estilos de sistema de mensagens também são referidos como *domínios do sistema de mensagens* e é possível combinar ambos os estilos do sistema de mensagens em um aplicativo. No domínio ponto a ponto, um destino é uma fila e, no domínio de publicar/assinar, um destino é um tópico.

Com versões do JMS anteriores ao JMS 1.1, a programação para o domínio ponto a ponto usa um conjunto de interfaces e métodos e a programação para o domínio de publicação/assinatura usa outro conjunto. Os dois conjuntos são semelhantes, mas separados. A partir do JMS 1.1, é possível usar um conjunto comum de interfaces e métodos que suportam ambos os domínios do sistema de mensagens. As interfaces comuns fornecem uma visualização independente de cada domínio de sistema de mensagens. Tabela 15 na página 144 lista as interfaces independentes de domínio do JMS e suas interfaces específicas de domínio correspondentes.

Tabela 15. As interfaces independentes de domínio e específicas de domínio do JMS

Interfaces independentes de domínio	**Interfaces específicas de domínio para o domínio ponto a ponto**	**Interfaces específicas de domínio para o domínio de publicar/assinar**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Conexão	QueueConnection	TopicConnection
Destino	Fila	Tópico
Sessão	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher

Tabela 15. As interfaces independentes de domínio e específicas de domínio do JMS (continuação)		
Interfaces independentes de domínio	**Interfaces específicas de domínio para o domínio ponto a ponto**	**Interfaces específicas de domínio para o domínio de publicar/assinar**
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 O IBM MQ classes for JMS 2.0 suporta ambas as interfaces específicas do domínio do JMS 1.1 anteriores e a API simplificada do JMS 2.0 IBM MQ classes for JMS 2.0 pode, portanto, ser usado para manter aplicativos existentes, incluindo o desenvolvimento de nova função em aplicativos existentes.

JM 3.0 O IBM MQ classes for Jakarta Messaging 3.0 suporta as versões do Jakarta Messaging das mesmas interfaces e é recomendado para o novo desenvolvimento de aplicativo

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, JMS objetos são relacionados a conceitos IBM MQ das seguintes maneiras:

- Um objeto Connection tem propriedades derivadas das propriedades do connection factory que foi usado para criar a conexão. Essas propriedades controlam como um aplicativo se conecta a um gerenciador de filas. Os exemplos dessas propriedades são o nome do gerenciador de filas e, para um aplicativo que se conecta ao gerenciador de filas no modo cliente, o nome do host ou o endereço IP do sistema em que o gerenciador de filas está em execução.
- Um objeto Session contém uma manipulação de conexões do IBM MQ que, portanto, define o escopo transacional da sessão.
- Cada objeto MessageProducer e cada objeto MessageConsumer contém uma manipulação de objetos do IBM MQ.

Ao usar IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, todas as regras normais de IBM MQ se aplicam.. Observe, especificamente, que um aplicativo pode enviar uma mensagem a uma fila remota, mas pode receber uma mensagem somente de uma fila de propriedade do gerenciador de filas ao qual o aplicativo está conectado.

A especificação do JMS espera que os objetos ConnectionFactory e Destination sejam objetos administrados. Um administrador cria e mantém objetos administrados em um repositório central e um aplicativo JMS recupera esses objetos usando a Java Naming and Directory Interface (JNDI).

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, a implementação da interface de Destino é uma superclasse abstrata de Fila e Tópico e, portanto, uma instância de Destino é um objeto Fila ou um objeto Tópico. As interfaces independentes de domínio tratam uma fila ou um tópico como um destino. O domínio do sistema de mensagens para um objeto MessageProducer ou MessageConsumer é determinado por se o destino é uma fila ou um tópico.

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging , portanto, os objetos dos seguintes tipos podem ser objetos administrados:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Fila
- Tópico
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Conceitos relacionados

Interfaces de linguagem do IBM MQ Java

“Criando e configurando connection factories e destinos” na página 208
Um aplicativo IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging pode criar connection factories e destinos recuperando-os como objetos administrados de um namespace Java Naming and Directory Interface (JNDI), usando as extensões do IBM JMS ou usando as extensões do IBM MQ JMS . Um aplicativo também pode usar as extensões do IBM JMS ou do IBM MQ JMS para configurar as propriedades de ações de connection factories.

Mensagens JMS

As mensagens do JMS são compostas de um cabeçalho, propriedades e um corpo. O JMS define cinco tipos de corpo da mensagem.

As mensagens do JMS são compostas das seguintes partes:

Cabeçalho
Todas as mensagens suportam o mesmo conjunto de campos de cabeçalho. Os campos de cabeçalho contêm valores que são usados pelos clientes e provedores para identificar e rotear as mensagens.

Propriedades
Cada mensagem contém um recurso integrado para suportar os valores de propriedade definidos pelo aplicativo. As propriedades fornecem um mecanismo eficiente para filtrar as mensagens definidas pelo aplicativo.

Conteúdo
O JMS define cinco tipos de corpo da mensagem que abrangem a maioria dos estilos de mensagens atualmente em uso:

Fluxo
Um fluxo de valores primitivos Java. É preenchido e lido sequencialmente.

Mapa
Um conjunto de pares nome-valor, em que os nomes são sequências e os valores são tipos primitivos Java. As entradas podem ser acessadas sequencialmente ou aleatoriamente pelo nome. A ordem das entradas é indefinida.

Texto
Uma mensagem que contém um java.lang.String.

Object
Uma mensagem que contém um objeto serializável do Java

Bytes
Um fluxo de bytes não interpretados. Este tipo de mensagem é para a codificação literal de um corpo para corresponder ao formato de mensagem existente.

O campo de cabeçalho JMSCorrelationID é usado para vincular uma mensagem à outra. Geralmente, ele vincula uma mensagem de resposta com sua mensagem de solicitação. O JMSCorrelationID pode reter um ID de mensagem específico do provedor, um Sequência específica do aplicativo ou um valor byte[] nativo do provedor.

Seletores de mensagens no JMS

As mensagens podem conter os valores de propriedades definidos pelo aplicativo. Um aplicativo pode usar seletores de mensagens para que um do provedor do JMS filtre mensagens.

Uma mensagem contém um recurso integrado para suportar os valores de propriedades definidos pelo aplicativo. Efetivamente, isso fornece um mecanismo para incluir campos de cabeçalho específicos do aplicativo em uma mensagem. Propriedades permitem que um aplicativo, usando seletores de mensagens, tenha um provedor do JMS que selecione ou filtre mensagens em seu nome, usando critérios específicos do aplicativo. Propriedades definidas pelo aplicativo devem obedecer às regras a seguir:

- Os nomes de propriedades devem obedecer às regras para um identificador de seletor de mensagem.
- Os valores de propriedades podem ser Boolean, byte, short, int, long, float, double e String.
- Os prefixos de nome JMSX e JMS_ são reservados.

Os valores de propriedades são configurados antes de enviar uma mensagem. Quando um cliente recebe uma mensagem, as propriedades da mensagem são somente leitura. Se um cliente tentar configurar

propriedades neste ponto, uma MessageNotWriteableException será lançada. Se clearProperties for chamado, as propriedades agora podem ser lidas e gravadas.

Um valor de propriedade pode duplicar um valor em um corpo da mensagem. O JMS não define uma política para o que pode ser feito em uma propriedade. No entanto, os desenvolvedores de aplicativos devem estar cientes de que os provedores do JMS provavelmente manipulam dados em um corpo de mensagem de forma mais eficiente do que os dados em propriedades de mensagem. Para melhor desempenho, os aplicativos devem usar as propriedades de mensagem somente quando precisam customizar um cabeçalho de mensagem. O motivo principal para fazer isso é suportar a seleção de mensagens customizadas.

Um seletor de mensagem do JMS permite que um cliente especifique as mensagens nas quais ele está interessado usando o cabeçalho da mensagem. Somente mensagens com cabeçalhos que correspondam ao seletor serão entregues.

os seletores de mensagens não podem fazer referência a valores do corpo da mensagem.

Um seletor de mensagem corresponde a uma mensagem quando o seletor é avaliado como verdadeiro quando o campo de cabeçalho da mensagem e os valores de propriedades são substituídos por seus identificadores correspondentes no seletor.

Um seletor de mensagem é uma String, com sintaxe baseada em um subconjunto da sintaxe de expressão condicional SQL92. A ordem na qual um seletor de mensagem é avaliado é da esquerda para a direita dentro de um nível de precedência. É possível usar parênteses para mudar essa ordem. Literais do seletor e nomes de operadores predefinidos estão gravados aqui em maiúsculas; no entanto, não fazem distinção entre maiúsculas e minúsculas.

Conteúdos de um seletor de mensagem

Um seletor de mensagem pode conter:

- Literais
  - Uma sequência literal é colocada entre aspas. Aspas duplas representam aspas simples. Os exemplos são: 'literal' e 'literal''s'. Como literais de sequência do Java, eles usam a codificação de caracteres Unicode.
  - Um literal numérico exato é um valor numérico sem um ponto decimal, como 57, -957 e +62. Os números no intervalo de Java long são suportados.
  - Um literal numérico aproximado é um valor numérico em notação científica, como 7E3 ou -57.9E2, ou um valor numérico com um decimal, como 7., -95,7 ou +6,2. Os números no intervalo de Java double são suportados.
  - Os literais booleanos TRUE e FALSE.
- Identificadores:
  - Um identificador é uma sequência de comprimento ilimitado de letras Java e dígitos Java, o primeiro dos quais deve ser uma letra Java. Uma letra é qualquer caractere para o qual o método Character.isJavaLetter retorna true. Isso inclui _ e $. Uma letra ou dígito é qualquer caractere para o qual o método Character.isJavaLetterOrDigit retorna true.
  - Identificadores não podem ser os nomes NULL, TRUE ou FALSE.
  - Identificadores não podem ser NOT, AND, OR, BETWEEN, LIKE, IN ou IS.
  - Identificadores são referências de campo de cabeçalho ou referências de propriedade.
  - Identificadores fazem distinção entre maiúsculas e minúsculas.
  - Referências de campo de cabeçalho da mensagem são restritas a:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp

- JMSCorrelationID
- JMSType

Os valores JMSMessageID, JMSTimestamp, JMSCorrelationID e JMSType podem ser nulos e, se forem, serão tratados como um valor NULL.

  - Qualquer nome que começa com JMSX é um nome de propriedade definido por JMS.
  - Qualquer nome que começa com JMS_ é um nome da propriedade específico do provedor.
  - Qualquer nome não iniciado por JMS é um nome de propriedade específico do aplicativo. Se houver uma referência a uma propriedade que não existe em uma mensagem, seu valor é NULL. Se existir, seu valor é o valor de propriedade correspondente.
- Espaço em branco é o mesmo que está definido para Java: espaço, tabulação horizontal, avanço de formulário e terminador de linha.
- Expressões:
  - Um seletor é uma expressão condicional. Um seletor avaliado como true tem correspondência; um seletor avaliado como false ou unknown não tem correspondência.
  - As expressões aritméticas são compostas de si mesmas, operações aritméticas, identificadores (com um valor que é tratado como um literal numérico) e literais numéricos.
  - Expressões condicionais são compostas por si mesmas, por operações de comparação e por operações lógicas.
- O uso padrão de parênteses () para configurar a ordem na qual as expressões são avaliadas é suportado.
- Os operadores lógicos em ordem de precedência: NOT, AND, OR.
- Operadores de comparação: =, >, >=, <, <=, <> (diferente).
  - Somente valores do mesmo tipo podem ser comparados. Uma exceção é que a validade de comparar valores numéricos exatos e valores numéricos aproximados. (A conversão de tipo necessária é definida pelas regras de promoção numérica Java.) Se houver uma tentativa de comparar tipos diferentes, o seletor será sempre false.
  - A comparação de String e Boolean é restrita a = e <>. Duas sequências serão iguais apenas se contiverem a mesma sequência de caracteres.
- Operadores aritméticos em ordem de precedência:
  - +, - unário.
  - *, /, multiplicação e divisão.
  - +, -, soma e subtração.
  - Operações aritméticas em um valor NULL não são suportadas. Se forem tentadas, o seletor completo será sempre false.
  - Operações aritméticas devem usar promoção numérica Java.
- Operador de comparação arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3:
  - Age BETWEEN 15 and 19 é equivalente a age >= 15 AND age <= 19.
  - Age NOT BETWEEN 15 and 19 é equivalente a age < 15 OR age> 19.
  - Se qualquer uma das expressões de uma operação BETWEEN for NULL, o valor da operação será false. Se qualquer uma das expressões de uma operação NOT BETWEEN for NULL, o valor da operação será true.
- identificador [NOT] IN (string-literal1, string-literal2, ...) o operador de comparação em que o identifier tem um valor String ou NULL.
  - Country IN ('UK', 'US', 'France') é true para 'UK' e false para 'Peru'. Ele é equivalente à expressão (Country = 'UK') OR (Country = 'US') OR (Country = 'France').
  - Country NOT IN ('UK', 'US', 'France') é false para 'UK' e true para 'Peru'. Ele é equivalente à expressão NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')).

  - Se o identificador de uma operação IN ou NOT IN for NULL, o valor da operação será desconhecido.
- Operador de comparação identifier [NOT] LIKE pattern-value [ESCAPE escape-character], em que identifier tem um valor de sequência. pattern-value é um literal de sequência, em que _ representa qualquer caractere único e % representa qualquer sequência de caracteres (incluindo a sequência vazia). Todos os outros caracteres representam eles mesmos. O escape-character opcional é uma sequência literal de caractere único, com um caractere usado para escapar o significado especial de _ e % no pattern-value.
  - phone LIKE '12%3' é true para 123 e 12993 e false para 1234.
  - word LIKE 'l_se' é true para "lose" e false para "loose".
  - underscored LIKE '_%' ESCAPE '\' é true para "_foo" e false para "bar".
  - phone NOT LIKE '12%3' é false para 123 e 12993 e true para 1234.
  - Se o identificador de uma operação LIKE ou NOT LIKE for NULL, o valor da operação será desconhecido.
- O operador de comparação identifier IS NULL testa para um valor de campo de cabeçalho nulo ou um valor de propriedade ausente.
  - prop_name IS NULL.
- O operador de comparação identifier IS NOT NULL testa a existência de um valor de campo de cabeçalho ou um valor de propriedade não nulo.
  - prop_name IS NOT NULL.

Exemplo de um seletor de mensagem

O seletor de mensagem a seguir seleciona mensagens com um tipo de mensagem de carro, cor azul e peso maior que 2500 lbs:

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

Valores de propriedades NULL

Conforme observado na lista anterior, os valores de propriedades podem ser NULL. A avaliação de expressões do seletor que contêm valores NULL é definida pela semântica de SQL 92 NULL. A lista a seguir fornece uma breve descrição dessa semântica:

- SQL trata um valor NULL como desconhecido.
- Comparação ou aritmética com um valor desconhecido sempre resulta em um valor desconhecido.
- O operador IS NULL converte um valor desconhecido em um valor TRUE.
- O operador IS NOT NULL converte um valor desconhecido em um valor FALSE.

Comportamento especial de JMSMessageID e JMSCorrelationID

As classes do IBM MQ para JMS contêm otimizações ao selecionar mensagens de uma fila baseada em JMSMessageID ou JMSCorrelationID.

Se um aplicativo especificar um seletor no formato:

JMSMessageID='ID:*message_id*'

em que *message_id* é uma sequência que contém um identificador de mensagem padrão do IBM MQ, as classes do IBM MQ para JMS usarão o **MatchOption** MQMO_MATCH_MSG_ID para obter a mensagem com o identificador de mensagem especificado.

Por exemplo, para obter uma mensagem com o identificador de mensagem 414D51207061756C745639314C545320C57C1A5F25ECE602 em uma fila, um aplicativo deve usar o seletor de mensagem a seguir:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

Da mesma forma, se o aplicativo especificar um seletor que tenha o formato:

JMSCorrelationID ='ID:*correlation_id*'

em que *correlation_id* é uma sequência que contém um identificador de correlação padrão do IBM MQ, as classes do IBM MQ para JMS usarão o **MatchOption** MQMO_MATCH_CORREL_ID para obter a mensagem com o identificador de correlação especificado na fila.

No exemplo a seguir, um seletor de mensagens é usado para obter uma mensagem que tem um identificador de correlação de 414D51207061756C745639314C545320846E5B5F25B1CC02:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Se um seletor de mensagem contiver um valor de todos os zeros para *message_id* ou *correlation_id*, ele corresponderá a qualquer mensagem na fila. Por exemplo, se um aplicativo estiver usando o seletor:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

qualquer mensagem na fila será considerada uma correspondência e retornada para o aplicativo.

Para obter mais informações sobre o **MatchOptions** MQMO_MATCH_MSG_ID e MQMO_MATCH_CORREL_ID, consulte MatchOptions (MQLONG).

Restrições

Embora SQL suporte comparação e aritmética de decimal fixo, seletores de mensagem JMS não suportam. Por isso literais numéricos exatos são restritos àqueles sem um decimal. Por isso também há valores numéricos com um decimal como uma representação alternativa para um valor numérico aproximado.

Comentários SQL não são suportados.

Mapeando mensagens do JMS para mensagens do IBM MQ
As mensagens do IBM MQ são compostas por um Message Descriptor, um cabeçalho MQRFH2 opcional e um corpo. Os conteúdos de uma mensagem do JMS são parcialmente mapeados e copiados para uma mensagem do IBM MQ.

Este tópico descreve como a estrutura de mensagem do JMS descrita na primeira parte desta seção é mapeada em uma mensagem do IBM MQ. É de interesse para programadores que desejam transmitir mensagens entre o JMS e aplicativos IBM MQ tradicionais. Também é de interesse para pessoas que desejam manipular mensagens transmitidas entre dois aplicativos JMS, por exemplo, em uma implementação do IBM Integration Bus.

Esta seção não se aplica se um aplicativo usar uma conexão em tempo real com um broker. Quando um aplicativo usa uma conexão em tempo real, toda a comunicação é executada diretamente em TCP/IP: nenhuma fila ou mensagem do IBM MQ está envolvida.

As mensagens do IBM MQ são compostas por três componentes:

- O IBM MQ Message Descriptor (MQMD)
- Um cabeçalho IBM MQ MQRFH2
- O corpo da mensagem.

O MQRFH2 é opcional e sua inclusão em uma mensagem não enviada é regida pela sinalização TARGCLIENT na classe Destination do JMS. É possível configurar essa sinalização usando a ferramenta de administração do IBM MQ JMS. Como o MQRFH2 transporta informações específicas do JMS, sempre o inclua na mensagem quando o emissor souber que o destino de recebimento é um aplicativo JMS.

Normalmente, omita o MQRFH2 ao enviar uma mensagem diretamente para um aplicativo não JMS. Isso porque esse tipo de aplicativo não espera um MQRFH2 em sua mensagem do IBM MQ.

Se uma mensagem recebida não tiver um cabeçalho MQRFH2, o objeto Fila ou Tópico do campo de cabeçalho derivado do campo de cabeçalho JMSReplyTo da mensagem, por padrão, terá este sinalizador configurado de modo que uma mensagem de resposta enviada à fila ou tópico também não tenha um cabeçalho MQRFH2. Será possível desativar esse comportamento de inclusão de um cabeçalho MQRFH2 em uma mensagem de resposta somente se a mensagem original tiver um cabeçalho MQRFH2, configurando a propriedade TARGCLIENTMATCHING do connection factory para NO.

Figura 10 na página 151 mostra como a estrutura de uma mensagem do JMS é transformada em uma mensagem do IBM MQ e de volta:

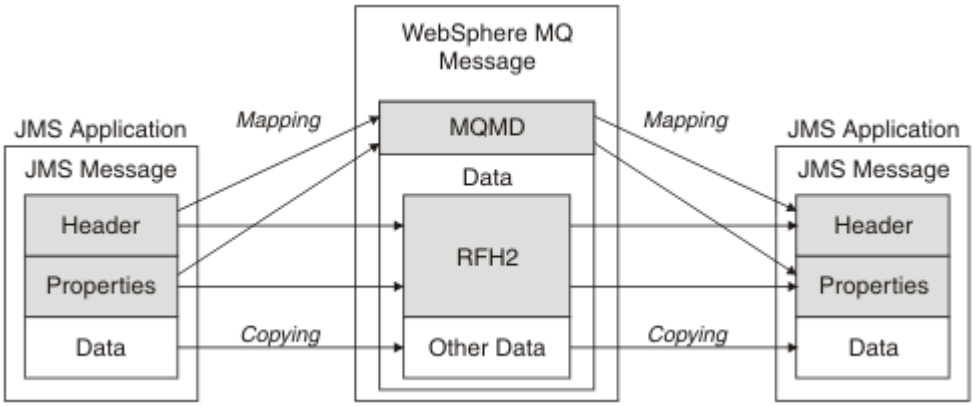


Figura 10. Como as mensagens são transformadas entre o JMS e o IBM MQ usando o cabeçalho MQRFH2

As estruturas são transformadas de duas maneiras:

Mapping
Quando o MQMD inclui um campo equivalente ao campo do JMS, o campo do JMS é mapeado para o campo MQMD. Campos MQMD adicionais são expostos como propriedades do JMS, pois um aplicativo JMS pode precisar obter ou configurar esses campos ao se comunicar com um aplicativo não JMS.

Copiando
Quando não houver nenhum MQMD equivalente, um campo ou propriedade de cabeçalho do JMS é passado, possivelmente transformado, como um campo dentro do MQRFH2.

O cabeçalho MQRFH2 e JMS
Esta coleção de tópicos descreve o cabeçalho MQRFH Versão 2, que transporta dados específicos do JMS que estão associados ao conteúdo da mensagem. O cabeçalho MQRFH Versão 2 é extensível e pode também transportar informações adicionais que não estão diretamente associadas ao JMS. No entanto, esta seção abrange apenas seu uso pelo JMS.Para obter uma descrição completa, consulte MQRFH2 - Regras e formatação do cabeçalho 2.

Há duas partes do cabeçalho, uma parte fixa e uma parte variável.

Parte fixa
A parte fixa é modelada no padrão de cabeçalho *padrão* IBM MQ e consiste nos seguintes campos:

StrucId (MQCHAR4)
Identificador de estruturação.

Deve ser MQRFH_STRUC_ID (valor: "RFH ") (valor inicial).

MQRFH_STRUC_ID_ARRAY (valor: "R", "F", "H", " ") também é definido.

Versão (MQLONG)
Número de versão da estrutura.

Deve ser MQRFH_VERSION_2 (valor: 2) (valor inicial).

StrucLength (MQLONG)
Comprimento total de MQRFH2, incluindo os campos NameValueData.

O valor configurado em StrucLength deve ser um múltiplo de 4 (os dados nos campos NameValueData podem ser preenchidos com caracteres de espaço para fazer isso).

Codificação (MQLONG)
Codificação de dados.

Codificação de quaisquer dados numéricos na parte da mensagem após MQRFH2 (o próximo cabeçalho ou os dados da mensagem após esse cabeçalho).

CodedCharSetId (MQLONG)
Identificador do conjunto de caracteres codificados.

Representação de quaisquer dados de caracteres na parte da mensagem após MQRFH2 (o próximo cabeçalho ou os dados da mensagem após esse cabeçalho).

Formato (MQCHAR8)
Nome do formato.

Nome do formato para a parte da mensagem após MQRFH2.

Sinalizadores (MQLONG)
Sinalizadores.

MQRFH_NO_FLAGS =0. Nenhum conjunto de sinalizadores.

NameValueCCSID (MQLONG)
O identificador do conjunto de caracteres codificados (CCSID) para as sequências de caracteres NameValueData contidas neste cabeçalho. O NameValueData pode ser codificado em um conjunto de caracteres que difere das outras sequências de caracteres que estão contidas no cabeçalho (StrucID e Format).

Se o NameValueCCSID for um CCSID Unicode de 2 bytes (1200, 13488 ou 17584), a ordem de bytes do Unicode será a mesma que a ordem de bytes dos campos numéricos no MQRFH2. (Por exemplo, Versão, StrucLength e NameValueCCSID em si.)

Tabela 16. Valores possíveis para o campo NameValueCCSID

CCSID	**Significado**
1200	UTF-16, a versão Unicode mais recente suportada
13488	UTF-16, o subconjunto da versão Unicode 2.0
17584	UTF-16, o subconjunto da versão Unicode 3.0 (inclui o símbolo do euro)
1208	UTF-8, a versão Unicode mais recente suportada

Parte variável
A parte variável segue a parte fixa. A parte variável contém um número variável de pastas do MQRFH2. Cada pasta contém um número variável de elementos ou propriedades. Propriedades relacionadas ao grupo de pastas. Os cabeçalhos MQRFH2 criados por JMS podem conter qualquer uma das pastas a seguir:

A pasta `mcd`

O `mcd` contém propriedades que descrevem o formato da mensagem. Por exemplo, a propriedade `Msd` do domínio de serviço de mensagem identifica uma mensagem JMS como sendo `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` ou nula.

A pasta `mcd` está sempre presente em uma mensagem JMS que contém um `MQRFH2`.

Ela está sempre presente em uma mensagem que contém um `MQRFH2` enviado de IBM Integration Bus. Isso descreve o domínio, o formato, o tipo e o conjunto de mensagens de uma mensagem.

Tabela 17. mcd nome da propriedade, sinônimo, tipo de dados e pasta			
Sinônimo da Propriedade	**Nome da Propriedade**	**Tipo de Dados**	**Pasta**
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

Não inclua suas próprias propriedades na pasta mcd.

A pasta jms

O jms contém campos de cabeçalho JMS, e propriedades JMSX que não são totalmente expressas no MQMD. A pasta jms está sempre presente em um JMS MQRFH2.

A pasta usr

O usr contém propriedades JMS definidas por aplicativo associadas à mensagem. A pasta usr estará presente apenas quando um aplicativo tiver configurado uma propriedade definida pelo aplicativo.

A pasta mqext

O mqext contém os seguintes tipos de propriedade:

- Propriedades que são usadas somente pelo WebSphere Application Server.
- Propriedades relacionadas ao atraso na entrega de mensagens.

A pasta somente estará presente se o aplicativo tiver configurado pelo menos uma das propriedades definidas do IBM ou usado o atraso de entrega.

Tabela 18. mqext nome da propriedade, sinônimo, tipo de dados e pasta			
Sinônimo da Propriedade	**Nome da Propriedade**	**Tipo de Dados**	**Pasta**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

Não inclua suas próprias propriedades na pasta mqext.

A pasta mqps

O mqps contém propriedades usadas apenas por IBM MQ publicar/assinar. A pasta estará presente somente se o aplicativo tiver configurado pelo menos uma das propriedades de publicação/assinatura integradas.

Tabela 19. mqps nome da propriedade, sinônimo, tipo de dados e pasta			
Sinônimo da Propriedade	**Nome da Propriedade**	**Tipo de Dados**	**Pasta**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData...*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

Não inclua suas próprias propriedades na pasta mqps.

Tabela 20 na página 154 mostra uma lista completa de nomes de propriedades.

Tabela 20. Pastas MQRFH2 e propriedades usadas pelo JMS				
Nome do campo JMS	**Tipo de Java**	**Nome da pasta MQRFH2**	**Nome da Propriedade**	**Tipo/valores**
JMSDestination	Destino	jms	Dst	sequência
JMSExpiration	long	jms	Expand.	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Sequência	jms	Cid	sequência
JMSReplyTo	Destino	jms	Rto	sequência
JMSTimestamp	long	jms	Tms	i8
JMSType	Sequência	mcd	Tipo, Configuração, Fmt	sequência
JMSXGroupID	Sequência	jms	Gid	sequência
JMSXGroupSeq	int	jms	Seq.	i4
xxx (definido pelo usuário)	Qualquer	usr	xxx	qualquer

Tabela 20. Pastas MQRFH2 e propriedades usadas pelo JMS (continuação)				
Nome do campo JMS	**Tipo de Java**	**Nome da pasta MQRFH2**	**Nome da Propriedade**	**Tipo/valores**
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength (MQLONG)
Comprimento em bytes da sequência NameValueData que segue imediatamente este campo de comprimento (não inclui seu próprio comprimento).

NameValueData (MQCHARn)
Uma única sequência de caracteres, cujo comprimento em bytes é dado pelo campo NameValueLength anterior. Contém uma pasta que mantém uma sequência de propriedades. Cada propriedade é um trio de nome/tipo/valor, contido em um elemento XML cujo nome é o nome da pasta, conforme a seguir:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

A tag de fechamento </foldername> pode ser seguida por espaços como caracteres de preenchimento. Cada trio é codificado usando uma sintaxe semelhante a XML:

```
<name dt='datatype'>value</name>
```

O elemento dt='datatype' é opcional e é omitido para muitas propriedades, pois o tipo de dado é predefinido. Se ele for incluído, um ou mais caracteres de espaço deverão ser incluídos antes da tag dt=.

name
é o nome da propriedade; consulte Tabela 20 na página 154.

datatype
deve corresponder, após compactação, a um dos tipos de dados listados em Tabela 21 na página 155.

value
é uma representação de sequência do valor a ser transmitido, usando as definições em Tabela 21 na página 155.

Um valor nulo é codificado usando a seguinte sintaxe:

```
<name dt='datatype' xsi:nil='true'></name>
```

Não use xsi:nil='false'.

Tabela 21. Tipos de dados de propriedade	
Tipo de Dados	**Definição**
sequência	Qualquer sequência de caracteres excluindo < e &
booleano	O caractere 0 ou 1 (0 = falso, 1 = verdadeiro)
bin.hex	Dígitos hexadecimais que representam octetos

Tabela 21. Tipos de dados de propriedade (continuação)	
Tipo de Dados	**Definição**
i1	Um número, expresso usando dígitos `0..9`, com sinal opcional (sem frações ou expoente). Deve estar no intervalo de -128 a 127, inclusive
i2	Um número, expresso usando dígitos `0..9`, com sinal opcional (sem frações ou expoente). Deve estar no intervalo de -32768 a 32767, inclusive
i4	Um número, expresso usando dígitos `0..9`, com sinal opcional (sem frações ou expoente). Deve estar no intervalo de -2147483648 a 2147483647, inclusive
i8	Um número, expresso usando dígitos `0..9`, com sinal opcional (sem frações ou expoente). Deve estar no intervalo de -9223372036854775808 a 9223372036854775807, inclusive
int	Um número, expresso usando dígitos `0..9`, com sinal opcional (sem frações ou expoente). Deve estar no mesmo intervalo que i8. Isso pode ser usado no lugar de um dos tipos i* se o emissor não desejar associar uma precisão específica à propriedade
r4	Número de vírgula flutuante, magnitude < = 3.40282347E+38,> = 1.175E-37 expressado usando dígitos `0..9`, sinal opcional, dígitos fracionários opcionais, expoente opcional
r8	Número de vírgula flutuante, magnitude < = 1.7976931348623E+308,> = 2.225E-307 é expressado usando dígitos `0..9`, sinal opcional, dígitos fracionários opcionais, expoente opcional

Um valor de sequência pode conter espaços. Deve-se usar as seguintes sequências de escape em um valor de sequência:

- `&` para o caractere &
- `<` para o caractere <

É possível usar as seguintes sequências de escape, mas elas não são requeridas:

- `>` para o caractere >
- `&apos;` para o caractere '
- `&quot;` para o caractere "

Campos e propriedades do JMS com campos MQMD correspondentes
Estas tabelas mostram os campos MQMD equivalentes a campos de cabeçalho do JMS, propriedades do JMS e propriedades específicas do provedor do JMS.

O Tabela 22 na página 156 lista os campos de cabeçalho do JMS e Tabela 23 na página 157 lista as propriedades do JMS que são mapeadas diretamente para os campos MQMD. O Tabela 24 na página 157 lista as propriedades específicas do provedor e os campos MQMD para os quais elas estão mapeadas.

Tabela 22. Mapeamento de campos de cabeçalho do JMS para campos MQMD			
Campo de cabeçalho doJMS	**Tipo de Java**	**Campo MQMD**	**tipo C**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiração	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	Sequência	MsgID	MQBYTE24

Tabela 22. Mapeamento de campos de cabeçalho do JMS para campos MQMD (continuação)			
Campo de cabeçalho doJMS	**Tipo de Java**	**Campo MQMD**	**tipo C**
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Sequência	CorrelId	MQBYTE24

Tabela 23. Mapeamento de propriedades do JMS para campos MQMD			
Propriedade JMS	**Tipo de Java**	**Campo MQMD**	**tipo C**
JMSXUserID	Sequência	UserIdentifier	MQCHAR12
JMSXAppID	Sequência	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	Sequência	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tabela 24. Mapeamento de propriedades específica do provedor do JMS para campos MQMD			
Propriedade específica do provedor do JMS	**Tipo de Java**	**Campo MQMD**	**tipo C**
JMS_IBM_Report_Exception	int	Relatório	MQLONG
JMS_IBM_Report_Expiration	int	Relatório	MQLONG
JMS_IBM_Report_COA	int	Relatório	MQLONG
JMS_IBM_Report_COD	int	Relatório	MQLONG
JMS_IBM_Report_PAN	int	Relatório	MQLONG
JMS_IBM_Report_NAN	int	Relatório	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Relatório	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Relatório	MQLONG
JMS_IBM_Report_Discard_Msg	int	Relatório	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	Sequência	Formatar “1” na página 158	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Codificação	MQLONG
JMS_IBM_Character_Set	Sequência	CodedCharacterSetId “2” na página 158	MQLONG

Tabela 24. Mapeamento de propriedades específica do provedor do JMS para campos MQMD (continuação)			
Propriedade específica do provedor do JMS	**Tipo de Java**	**Campo MQMD**	**tipo C**
JMS_IBM_PutDate	Sequência	PutDate	MQCHAR8
JMS_IBM_PutTime	Sequência	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	booleano	MsgFlags	MQLONG

Nota:

1. JMS_IBM_Format representa o formato do corpo da mensagem. Isso pode ser definido pelo aplicativo, configurando a propriedade JMS_IBM_Format da mensagem (observe que há um limite de 8 caracteres) ou padronizando para o formato IBM MQ do corpo da mensagem apropriado para o tipo de mensagem do JMS. JMS_IBM_Format é mapeado para o campo Formato de MQMD apenas se a mensagem não contém seções RFH ou RFH2. Em uma mensagem típica, ele mapeia para o campo Formato do RFH2 precedendo imediatamente o corpo da mensagem.
2. O valor da propriedade JMS_IBM_Character_Set é um valor de sequência de caracteres que contém o conjunto de caracteres Java equivalente para o valor numérico CodedCharacterSetId. O campo CodedCharacterSetId do MQMD é um valor numérico que contém o equivalente da sequência do conjunto de caracteres Java especificado pela propriedade JMS_IBM_Character_Set.

Mapeando campos do JMS para campos do IBM MQ (mensagens de saída)
Estas tabelas mostram como os campos de propriedade e cabeçalho do JMS são mapeados para campos MQMD e MQRFH2 no momento de send() ou publish().

O Tabela 25 na página 158 mostra como os campos de cabeçalho do JMS são mapeados para campos MQMD/RFH2 no momento de send() ou publish(). O Tabela 26 na página 159 mostra como as propriedades do JMS são mapeadas para campos MQMD/RFH2 no momento de send () ou publish(). O Tabela 27 na página 160 mostra como as propriedades específicas do provedor do JMS são mapeadas para os campos MQMD no momento de send() ou publish().

Para campos marcados como Definido pelo Objeto de Mensagem, o valor transmitido é o valor retido na mensagem do JMS imediatamente antes da operação send() ou publish(). O valor na mensagem JMS é deixado inalterado pela operação.

Para campos marcados como Definido pelo Método de Envio, um valor será designado quando o send() ou publish() for executado (qualquer valor retido na mensagem do JMS será ignorado). O valor na mensagem do JMS é atualizado para mostrar o valor usado.

Os campos marcados como Somente Receber não são transmitidos e são mantidos sem mudanças na mensagem por envio() ou publicação().

Tabela 25. Mapeamento do campo de mensagem de saída			
Nome do campo de cabeçalho do JMS	**Campo MQMD usado para a transmissão**	**Cabeçalho**	**Configurado por**
JMSDestination		MQRFH2	Método de envio
JMSDeliveryMode	Persistence	MQRFH2	Método de envio
JMSExpiration	Expiração	MQRFH2	Método de envio
JMSPriority	Priority	MQRFH2	Método de envio
JMSMessageID	MsgID		Método de envio

Tabela 25. Mapeamento do campo de mensagem de saída (continuação)			
Nome do campo de cabeçalho do JMS	**Campo MQMD usado para a transmissão**	**Cabeçalho**	**Configurado por**
JMSTimestamp	PutDate/PutTime		Método de envio
JMSCorrelationID	CorrelId	MQRFH2	Objeto de mensagem
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Objeto de mensagem
JMSType		MQRFH2	Objeto de mensagem
JMSRedelivered			Somente Receber

Nota:

1. O campo CodedCharacterSetId do MQMD é um valor numérico que contém o equivalente da sequência do conjunto de caracteres Java especificado pela propriedade JMS_IBM_Character_Set.

Tabela 26. Propriedade de mapeamento JMS da mensagem de saída			
Nome da propriedade JMS	**Campo MQMD usado para a transmissão**	**Cabeçalho**	**Configurado por**
JMSXUserID	UserIdentifier		Método de envio
JMSXAppID	PutApplName		Método de envio
JMSXDeliveryCount			Somente Receber
JMSXGroupID	GroupId	MQRFH2	Objeto de mensagem
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Objeto de mensagem

Nota:

Essas propriedades são definidas como somente leitura pela especificação do JMS e são configuradas (em alguns casos, opcionalmente) pelo provedor JMS.

No IBM MQ classes for JMS, duas dessas propriedades podem ser substituídas pelo aplicativo. Para fazer isso, certifique-se de que o destino tenha sido configurado adequadamente definindo as propriedades a seguir:

1. Configure a propriedade `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` como `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`.
2. Configure a propriedade `WMQConstants.WMQ_MQMD_WRITE_ENABLED` como `true`.

As propriedades a seguir podem ser substituídas pelo aplicativo:

JMSXAppID
Essa propriedade pode ser substituída configurando a propriedade `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` na mensagem-o valor deve ser uma sequência Java .

JMSXGroupID
Essa propriedade pode ser substituída configurando a propriedade `WMQConstants.JMS_IBM_MQMD_GROUPID` na mensagem - o valor deve ser uma matriz de bytes.

Tabela 27. Mensagem de saída do mapeamento de propriedade específico do provedor do JMS			
Nome da propriedade específico do provedor do JMS	**Campo MQMD usado para a transmissão**	**Cabeçalho**	**Configurado por**
JMS_IBM_Report_Exception	Relatório		Objeto de mensagem
JMS_IBM_Report_Expiration	Relatório		Objeto de mensagem
JMS_IBM_Report_COA/COD	Relatório		Objeto de mensagem
JMS_IBM_Report_NAN/PAN	Relatório		Objeto de mensagem
JMS_IBM_Report_Pass_Msg_ID	Relatório		Objeto de mensagem
JMS_IBM_Report_Pass_Correl_ID	Relatório		Objeto de mensagem
JMS_IBM_Report_Discard_Msg	Relatório		Objeto de mensagem
JMS_IBM_MsgType	MsgType		Objeto de mensagem
JMS_IBM_Feedback	Feedback		Objeto de mensagem
JMS_IBM_Format	Formato		Objeto de mensagem
JMS_IBM_PutApplType	PutApplType		Método de envio
JMS_IBM_Encoding	Codificação		Objeto de mensagem
JMS_IBM_Character_Set	CodedCharacterSetId		Objeto de mensagem
JMS_IBM_PutDate	PutDate		Método de envio
JMS_IBM_PutTime	PutTime		Método de envio
JMS_IBM_Last_Msg_In_Group	MsgFlags		Objeto de mensagem

Mapeamento de campos de cabeçalho do JMS no send() ou publish()
Estas observações relacionadas ao mapeamento de campos do JMS em send() ou publish().

JMSDestination para MQRFH2
Isso é armazenado como uma sequência que serializa as características salientes do objeto de destino para que um JMS de recebimento possa reconstituir um objeto de destino equivalente. O campo MQRFH2 é codificado como URI (consulte "Identificadores uniformes de recursos (URIs)" na página 226 para obter detalhes da notação de URI).

JMSReplyTo para MQMD.ReplyToQ, ReplyToQMgr, MQRFH2
O nome da fila é copiado para o campo MQMD.ReplyToQ e o nome do gerenciador de filas é copiado para os campos ReplyToQMgr. As informações de extensão de destino (outros detalhes úteis mantidos no objeto de destino) serão copiadas no campo MQRFH2. O campo MQRFH2 é codificado como um URI (consulte "Identificadores uniformes de recursos (URIs)" na página 226 para obter detalhes da notação URI).

JMSDeliveryMode para MQMD.Persistence

O valor JMSDeliveryMode é configurado pelo método send() ou publish() ou MessageProducer, a menos que o objeto de destino substitua-o. O valor JMSDeliveryMode é mapeado para o campo MQMD.Persistence, conforme a seguir:

- O valor JMS PERSISTENT é equivalente a MQPER_PERSISTENT
- O valor JMS NON_PERSISTENT é equivalente a MQPER_NOT_PERSISTENT

Se a propriedade de persistência MQQueue não estiver configurada como WMQConstants.WMQ_PER_QDEF, o valor do modo de entrega também será codificado no MQRFH2.

JMSExpiration para/de MQMD.Expiry, MQRFH2

JMSExpiration armazena a hora para expirar (a soma da hora atual e o tempo de vida), enquanto que MQMD armazena o tempo de vida. Além disso, JMSExpiration está em milissegundos, mas MQMD.Expiry está em décimos de um segundo.

- Se o método send() configura um tempo de vida ilimitado, MQMD.Expiry é configurado como MQEI_UNLIMITED e nenhum JMSExpiration é codificado no MQRFH2.
- Se o método send() configurar um tempo de vida menor que 214.748.364,7 segundos (aproximadamente 7 anos), o tempo de vida será armazenado em MQMD.Expiry e o tempo de expiração (em milissegundos) será codificada como um valor i8 no MQRFH2.
- Se o método send() configurar um tempo de vida superior a 214.748.364,7 segundos, MQMD.Expiry será configurado como MQEI_UNLIMITED. O prazo de expiração de true em milissegundos é codificado como um valor i8 no MQRFH2.

JMSPriority para MQMD.Priority

Mapeie diretamente o valor JMSPriority (0-9) para o valor de prioridade MQMD (0-9). Se JMSPriority está configurado para um valor não padrão, o nível de prioridade também é codificado no MQRFH2.

JMSMessageID de MQMD.MessageID

Todas as mensagens enviadas a partir do JMS possuem identificadores de mensagem exclusivos designado pelo IBM MQ. O valor designado é retornado no campo MQMD.MessageId após a chamada MQPUT e transmitido de volta para o aplicativo no campo JMSMessageID. O IBM MQ messageId é um valor binário de 24 bytes, enquanto que o JMSMessageID é uma sequência. O JMSMessageID é composto do valor messageId binário convertido para uma sequência de 48 caracteres hexadecimais, prefixados com o ID de caracteres: JMS fornece uma sugestão que pode ser configurada para desativar a produção de identificadores de mensagem. Essa sugestão é ignorada e um identificador exclusivo designado em todos os casos. Qualquer valor que é configurado para o campo JMSMessageID antes de um send() é sobrescrito.

Se você requerer a capacidade de especificar o MQMD.MessageID, poderá fazer isso com uma das extensões do IBM MQ JMS descritas em "Lendo e gravando o descritor de mensagens a partir de um aplicativo IBM MQ classes for JMS" na página 249.

JMSTimestamp para MQRFH2

Durante um envio, o campo JMSTimestamp é configurado de acordo com o relógio da JVM. Esse valor é configurado no MQRFH2. Qualquer valor que é configurado para o campo JMSTimestamp antes de um send() é sobrescrito. Consulte também as propriedades JMS_IBM_PutDate e JMS_IBM_PutTime.

JMSType para MQRFH2

Esta sequência é configurada no campo MQRFH2 mcd.Type. Se ela estiver em formato URI, também poderá afetar os campos mcd.Set e mcd.Fmt.

JMSCorrelationID para MQMD.CorrelId, MQRFH2

O JMSCorrelationID pode conter um dos seguintes:

Um ID de mensagem específico do provedor

Esse é um identificador de mensagem de uma mensagem enviada ou recebida anteriormente e, por isso, deve ser uma sequência de 48 dígitos hexadecimais minúsculos prefixados com `ID:` O prefixo é removido, os caracteres restantes são convertidos em binário e, em seguida, configurados no campo MQMD.CorrelId.

Um valor provider-native byte[]
O valor é copiado no campo MQMD.CorrelId preenchido com nulos ou truncado para 24 bytes, se necessário. Nenhum valor CorrelId é codificado no MQRFH2.

Uma sequência específica do aplicativo
O valor é copiado no MQRFH2. Os primeiros 24 bytes da sequência, no formato UTF8, são gravados no MQMD.CorrelID.

Mapeamento de campos de propriedade do JMS
Estas observações se referem ao mapeamento de campos de propriedade do JMS em mensagens do IBM MQ.

JMSXUserID de MQMD UserIdentifier
JMSXUserID é configurado no retorno da chamada de envio.

JMSXAppID de MQMD PutApplName
JSMXAppID está configurado no retorno a partir da chamada de envio.

JMSXGroupID para MQRFH2 (ponto a ponto)
Para mensagens ponto a ponto, o JMSXGroupID é copiado no campo GroupID do MQMD. Se o JMSXGroupID começar com o ID de prefixo, ele será convertido em binário. Caso contrário, ele será codificado como uma sequência UTF8. O valor será preenchido ou truncado, se necessário, para um comprimento de 24 bytes. O sinalizador MQMF_MSG_IN_GROUP está configurado.

JMSXGroupID para MQRFH2 (publicar/assinar)
Para mensagens de publicação/assinatura, o JMSXGroupID é copiado no MQRFH2 como uma sequência.

JMSXGroupSeq MQMD MsgSeqNumber (ponto a ponto)
Para mensagens ponto a ponto, o JMSXGroupSeq é copiado para o campo MsgSeqNumber do MQMD. O sinalizador MQMF_MSG_IN_GROUP está configurado.

JMSXGroupSeq MQMD MsgSeqNumber (publicar/assinar)
Para mensagens de publicação/assinatura, o JMSXGroupSeq é copiado no MQRFH2 como um i4.

Mapeamento de campos específicos de provedor do JMS
As notas a seguir se referem ao mapeamento de campos específicos do provedor do JMS em mensagens do IBM MQ.

JMS_IBM_Report_*XXX* para relatório do MQMD
Um aplicativo JMS pode configurar as opções de Relatório do MQMD, usando as seguintes propriedades JMS_IBM_Report_*XXX*. O MQMD único é mapeado para diversas propriedades JMS_IBM_Report_*XXX*.

As constantes JMS_IBM_Report_*XXX* estão em `com.ibm.msg.client.jakarta.wmq.WMQConstants` ou `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_Report_Exception

MQRO_EXCEPTION ou
MQRO_EXCEPTION_WITH_DATA ou
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION ou
MQRO_EXPIRATION_WITH_DATA ou
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA ou
MQRO_COA_WITH_DATA ou
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD ou
MQRO_COD_WITH_DATA ou
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

Os valores de MQRO estão em `com.ibm.mq.constants.CMQC`

JMS_IBM_MsgType para MQMD MsgType
O valor é mapeado diretamente no MQMD MsgType. Se o aplicativo não tiver configurado um valor explícito do JMS_IBM_MsgType, um valor padrão será usado. Esse valor padrão é determinado conforme a seguir:

- Se JMSReplyTo é configurado para um destino de fila do IBM MQ, MSGType é configurado para o valor MQMT_REQUEST
- Se JMSReplyTo não for configurado ou for configurado para algo diferente de um destino de fila do IBM MQ, MsgType será configurado para o valor MQMT_DATAGRAM

JMS_IBM_Feedback para MQMD Feedback
O valor é mapeado diretamente no MQMD Feedback.

JMS_IBM_Format para MQMD Format
O valor é mapeado diretamente no Formato MQMD.

JMS_IBM_Encoding para MQMD Encoding
Se configurada, esta propriedade substitui a codificação numérica da Fila de Destino ou Tópico.

JMS_IBM_Character_Set para MQMD CodedCharacterSetId
Se configurada, esta propriedade substitui a propriedade de conjunto de caracteres codificados da Fila de Destino ou Tópico.

JMS_IBM_PutDate de MQMD PutDate
O valor desta propriedade é configurado, durante o envio, diretamente do campo PutDate no MQMD. Qualquer valor que é configurado para a propriedade JMS_IBM_PutDate antes de um envio é sobrescrito. Este campo é uma Sequência de oito caracteres, no formato de Data IBM MQ AAAAMMDD. Essa propriedade pode ser usada com a propriedade JMS_IBM_PutTime para determinar o horário em que a mensagem foi colocada, de acordo com o gerenciador de filas.

JMS_IBM_PutTime de MQMD PutTime
O valor dessa propriedade é configurado, durante o envio, diretamente do campo PutTime no MQMD. Qualquer valor que é configurado para a propriedade JMS_IBM_PutTime antes de um envio é sobrescrito. Este campo é uma Sequência de oito caracteres, no formato de Tempo IBM MQ HHMMSSTH. Essa propriedade pode ser usada com a propriedade JMS_IBM_PutDate para determinar o horário em que a mensagem foi colocada, de acordo com o gerenciador de filas.

JMS_IBM_Last_Msg_In_Group para MQMD MsgFlags
Para sistema de mensagens ponto a ponto, esse valor booleano é mapeado para a sinalização MQMF_LAST_MSG_IN_GROUP no campo MQMD MsgFlags. Normalmente, ela é usada com as propriedades JMSXGroupID e JMSXGroupSeq para indicar a um aplicativo IBM MQ legado que essa mensagem é a última em um grupo. Essa propriedade é ignorada para o sistema de mensagens de publicação/assinatura.

Mapeamento de campos IBM MQ nos campos JMS (mensagens de entrada)
Estas tabelas mostram como os campos de cabeçalho JMS e de propriedade são mapeados em campos MQMD e MQRFH2 em get() ou receive() time.

O Tabela 28 na página 164 mostra como campos de cabeçalho JMS são mapeados em campos MQMD/MQRFH2 em get() ou receive() time. O Tabela 29 na página 164 mostra como campos de propriedade JMS são mapeados em campos MQMD/MQRFH2 em get() ou receive() time. O Tabela 30 na página 165 mostra como as propriedades específicas do provedor JMS são mapeadas.

Tabela 28. Mapeamento de campo de cabeçalho JMS da mensagem de entrada

Nome do campo de cabeçalho do JMS	**Campo MQMD recuperado**	**Campo MQRFH2 recuperado**
JMSDestination		jms.Dst ou mqps.Top "1" na página 164
JMSDeliveryMode	Persistence "2" na página 164	jms.Dlv "2" na página 164
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate "2" na página 164 PutTime "2" na página 164	jms.Tms "2" na página 164
JMSCorrelationID	CorrelId "2" na página 164	jms.Cid "2" na página 164
JMSReplyTo	ReplyToQ "2" na página 164 ReplyToQMgr "2" na página 164	jms.Rto "2" na página 164
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Nota:

1. Se jms.Dst e mqps.Top estão definidos, o valor em jms.Dst é usado.
2. Para propriedades que podem ter valores recuperados de MQRFH2 ou MQMD, se ambos estiverem disponíveis, a configuração no MQRFH2 é usada.
3. O valor da propriedade JMS_IBM_Character_Set é um valor de sequência de caracteres que contém o conjunto de caracteres Java equivalente para o valor numérico CodedCharacterSetId.

Tabela 29. Mapeamento de propriedade da mensagem de entrada

Nome da propriedade JMS	**Campo MQMD recuperado**	**Campo MQRFH2 recuperado**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId "1" na página 165	jms.Gid "1" na página 165
JMSXGroupSeq	MsgSeqNumber "1" na página 165	jms.Seq "1" na página 165

Nota:

1. Para propriedades que podem ter valores recuperados de MQRFH2 ou MQMD, se ambos estiverem disponíveis, a configuração no MQRFH2 é usada. As propriedades são configuradas a partir dos valores MQMD somente se os sinalizadores de mensagens MQMF_MSG_IN_GROUP ou MQMF_LAST_MSG_IN_GROUP estiverem configurados.

Tabela 30. Mapeamento de propriedade JMS específico do provedor da mensagem de entrada

Nome da propriedade JMS	**Campo MQMD recuperado**	**Campo MQRFH2 recuperado**
JMS_IBM_Report_Exception	Relatório	
JMS_IBM_Report_Expiration	Relatório	
JMS_IBM_Report_COA	Relatório	
JMS_IBM_Report_COD	Relatório	
JMS_IBM_Report_PAN	Relatório	
JMS_IBM_Report_NAN	Relatório	
JMS_IBM_Report_Pass_Msg_ID	Relatório	
JMS_IBM_Report_Pass_Correl_ID	Relatório	
JMS_IBM_Report_Discard_Msg	Relatório	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Formato	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding "1" na página 165	Codificação	
JMS_IBM_Character_Set "1" na página 165	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Somente configure se a mensagem de entrada for uma Mensagem de bytes.

Trocando mensagens entre um aplicativo JMS e um aplicativo tradicional IBM MQ
Este tópico descreve o que acontece quando um aplicativo JMS troca mensagens com um aplicativo tradicional IBM MQ que não pode processar o cabeçalho MQRFH2.

Figura 11 na página 166 mostra o mapeamento.

O administrador indica que o aplicativo JMS está se comunicando com um aplicativo tradicional IBM MQ configurando a propriedade TARGCLIENT do destino para *MQ*. Isso indica que nenhum cabeçalho MQRFH2 deve ser produzido. Se isso não for feito, o aplicativo de recebimento deverá ser capaz de manipular o cabeçalho MQRFH2.

O mapeamento a partir de JMS para MQMD destinado a um aplicativo tradicional IBM MQ é o mesmo que o mapeamento a partir de JMS para MQMD destinado a um aplicativo JMS. Se IBM MQ classes for JMS receber uma mensagem IBM MQ com o campo MQMD *Format* configurado para algo diferente de MQFMT_RFH2, os dados estão sendo recebidos de um aplicativo nãoJMS . Se o formato for MQFMT_STRING, a mensagem será recebida como uma mensagem de texto do JMS. Caso contrário, ela será recebida como uma mensagem de bytes do JMS. Como não há nenhum MQRFH2, apenas as propriedades do JMS transmitidas no MQMD podem ser restauradas.

Se o IBM MQ classes for JMS receber uma mensagem que não tem um cabeçalho MQRFH2, a propriedade TARGCLIENT do objeto Fila ou Tópico derivado do campo de cabeçalho do JMSReplyTo da mensagem será configurada como MQ, por padrão. Isso significa que uma mensagem de resposta enviada para a fila ou tópico também não tem um cabeçalho MQRFH2. Será possível desativar esse comportamento de inclusão de um cabeçalho MQRFH2 em uma mensagem de resposta somente se a mensagem original tiver um cabeçalho MQRFH2, configurando a propriedade TARGCLIENTMATCHING do connection factory para NO.

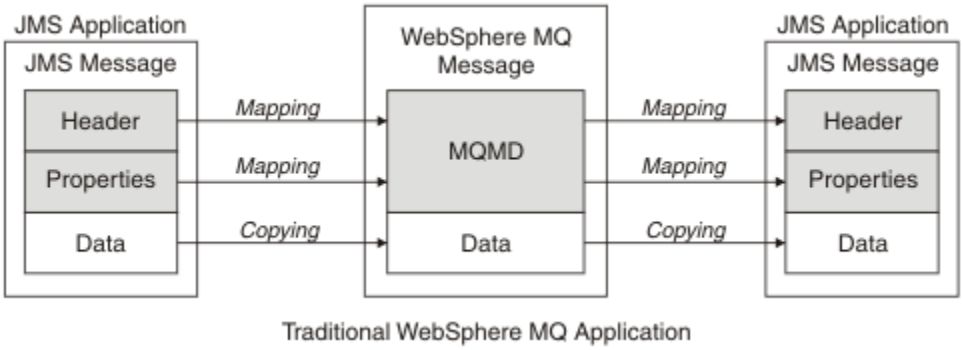


Figura 11. Como as mensagens do JMS são transformadas em mensagens do IBM MQ sem o cabeçalho MQRFH2

O corpo da mensagem do JMS
Este tópico contém informações sobre a codificação do corpo da mensagem em si. A codificação depende do tipo de mensagem do JMS.

ObjectMessage
Um ObjectMessage é um objeto serializado pelo Java Runtime da maneira normal.

TextMessage
TextMessage é uma sequência codificada. Para uma mensagem de saída, a sequência está codificada no conjunto de caracteres fornecido pelo objeto de destino. O padrão usado é a codificação UTF8 (a codificação UTF8 começa com o primeiro caractere da mensagem; não há campo de comprimento no início). No entanto, é possível especificar qualquer outro conjunto de caracteres suportados pelo IBM MQ classes for JMS. Esses conjuntos de caracteres são usados principalmente quando você envia uma mensagem para um aplicativo não JMS.

Se o conjunto de caracteres for um conjunto de byte duplo (incluindo UTF16), a especificação de codificação de número inteiro do objeto de destino determina a ordem dos bytes.

Uma mensagem recebida é interpretada usando o conjunto de caracteres e a codificação especificados na própria mensagem. Essas especificações estão no último cabeçalho do IBM MQ (ou MQMD se não houver cabeçalhos). Para mensagens do JMS, o último cabeçalho é normalmente o MQRFH2.

BytesMessage
BytesMessage é, por padrão, uma sequência de bytes, conforme definido pela especificação JMS 1.0.2 e documentação associada de Java.

Para uma mensagem não enviada que foi montada pelo próprio aplicativo, a propriedade de codificação do objeto de destino pode ser usada para substituir as codificações de número inteiro e os campos de vírgula flutuante na mensagem. Por exemplo, é possível solicitar que os valores de vírgula flutuante sejam armazenados em S/390 em vez de no formato IEEE.

Uma mensagem recebida é interpretada usando a codificação numérica especificada na própria mensagem. Essa especificação está no último cabeçalho do IBM MQ (ou MQMD se não houver cabeçalhos). Para mensagens do JMS, o último cabeçalho é normalmente o MQRFH2.

Se um BytesMessage for recebido e reenviado sem modificação, seu corpo será transmitido byte por byte, como foi recebido. A propriedade de codificação do objeto de destino não tem efeito no corpo. A única entidade semelhante a uma sequência que pode ser enviada explicitamente em um

BytesMessage é uma sequência UTF8. É codificada no formato UTF8 Java e começa com um campo de comprimento de 2 bytes. A propriedade do conjunto de caracteres do objeto de destino não tem efeito na codificação de um BytesMessage de saída. O valor do conjunto de caracteres em uma mensagem recebida do IBM MQ não tem efeito sobre a interpretação dessa mensagem como um JMS BytesMessage.

Aplicativos não Java provavelmente não reconhecerão a codificação Java UTF8. Portanto, para um aplicativo JMS enviar um BytesMessage que contenha dados de texto, o próprio aplicativo deve converter suas sequências em matrizes de bytes e gravar essas matrizes de bytes no BytesMessage.

MapMessage
Um MapMessage é uma sequência que contém os trios nome/tipo/valor XML codificados como:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

em que `datatype` é um dos tipos de dados listados em Tabela 21 na página 155. O tipo de dados padrão é `string` e, portanto, o atributo `dt="string"` é omitido para elementos de sequência.

O conjunto de caracteres usado para codificar ou interpretar a sequência XML que forma o corpo de uma mensagem de mapa é determinado de acordo com as regras que se aplicam a uma mensagem de texto.

As versões do IBM MQ classes for JMS anteriores à 5.3 codificavam o corpo de uma mensagem do mapa no formato a seguir:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

O IBM MQ classes for JMS 5.3 e mais recente pode interpretar qualquer um dos formatos, mas as versões do IBM MQ classes for JMS anteriores à 5.3 não podem interpretar o formato atual.

Se um aplicativo precisar enviar mensagens do mapa para outro aplicativo que esteja usando uma versão do IBM MQ classes for JMS anterior à 5.3, o aplicativo de envio deverá chamar o método de connection factory `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)` para especificar que as mensagens do mapa são enviadas no formato anterior. Por padrão, todas as mensagens do mapa são enviadas no formato atual.

StreamMessage
Um StreamMessage é semelhante a uma mensagem do mapa, mas sem nomes de elementos:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

em que `datatype` é um dos tipos de dados listados em Tabela 21 na página 155. O tipo de dados padrão é `string` e, portanto, o atributo `dt="string"` é omitido para elementos de sequência.

O conjunto de caractere usado para codificar ou interpretar a sequência XML que compõe o corpo do StreamMessage é determinado segundo as regras que se aplicam a um TextMessage.

O campo MQRFH2.format é configurado da seguinte forma:

MQFMT_NONE
para ObjectMessage, BytesMessage ou mensagens sem corpo.

MQFMT_STRING
para TextMessage, StreamMessage ou MapMessage.

Conversão de mensagens doJMS
A conversão de dados de mensagens no JMS é executada enviando e recebendo mensagens. IBM MQ executa a maioria dos dados de conversão automaticamente. Ele converte texto e dados numéricos ao transferir uma mensagem entre aplicativos do JMS. O texto é convertido ao trocar uma `JMSTextMessage` entre um aplicativo JMS e um aplicativo IBM MQ.

Se você estiver planejando fazer trocas de mensagens mais complexas, os tópicos a seguir serão de seu interesse. As trocas de mensagens complexas incluem:

- Transferindo mensagens não de texto entre um aplicativo IBM MQ e um aplicativo JMS.
- Trocando dados de texto no formato de byte.
- Convertendo o texto em seu aplicativo.

Dados da mensagem do JMS

A conversão de dados é necessária para troca de texto e dados numéricos entre aplicativos, mesmo entre dois aplicativos JMS. A representação interna de texto e números deve ser codificada para que possam ser transferidas em uma mensagem. A codificação força uma decisão sobre como números e texto são representados. IBM MQ gerencia a codificação de texto e números em mensagens JMS, exceto para `JMSObjectMessage`, consulte "JMSObjectMessage" na página 175. Ele usa três atributos de mensagem. Os três atributos são `CodedCharacterSetId`, `Encoding` e `Format`.

Esses três atributos de mensagem são normalmente armazenados no cabeçalho do JMS, `MQRFH2`, campos de uma mensagem do JMS. Se o tipo de mensagem for um `MQ`, em vez de o tipo de mensagem do `JMS`, os atributos serão armazenados no descritor de mensagens, `MQMD`. Os atributos são usados para converter os dados da mensagem do JMS. Os dados da mensagem do JMS são transferidos na parte de dados da mensagem de uma mensagem do IBM MQ.

propriedades de mensagem doJMS

Propriedades de mensagem do JMS, como `JMS_IBM_CHARACTER_SET`, são trocadas na parte do cabeçalho `MQRFH2` de uma mensagem do JMS, a menos que a mensagem tenha sido enviada sem um `MQRFH2`. Apenas `JMSTextMessage` e `JMSBytesMessage` podem ser enviados sem um `MQRFH2`. Se uma propriedade do JMS for armazenada como uma propriedade de mensagem do IBM MQ no descritor de mensagens, `MQMD`, será convertido como parte da conversão `MQMD`. Se uma propriedade do JMS for armazenada no `MQRFH2`, ela será armazenada no conjunto de caracteres especificado por `MQRFH2.NameValueCCSID`. Quando uma mensagem é enviada ou recebida, as propriedades da mensagem são convertidas para e a partir de sua representação interna na JVM. A conversão é para e a partir do conjunto de caracteres do descritor de mensagens ou `MQRFH2.NameValueCCSID`. Os dados numéricos são convertidos para texto.

Conversão de mensagens doJMS

Os tópicos a seguir contêm exemplos e tarefas que são úteis se você planeja trocar mensagens mais complexas que requerem conversão.

Abordagens de conversão de mensagem doJMS
Um número de abordagens de conversão de dados é aberto para designers de aplicativos JMS. Essas abordagens não são exclusivas; alguns aplicativos devem usar uma combinação dessas abordagens. Se seu aplicativo estiver trocando somente texto ou mensagens com outros aplicativos JMS, não considere a conversão de dados normalmente. A conversão de dados é executada automaticamente para você, por IBM MQ.

É possível fazer uma série de perguntas sobre como abordar a conversão de mensagens:

É necessário pensar sobre a conversão de mensagens mesmo?
Em alguns casos, como JMS para as transferências de mensagem do JMS e troca de mensagens de texto com programas IBM MQ, o IBM MQ executa as conversões necessárias para você, automaticamente. Você pode desejar controlar a conversão de dados por motivos de desempenho ou

trocar mensagens complexas que possuem um formato predefinido. Em casos como esses, deve-se entender a conversão de mensagens e ler os seguintes tópicos.

Quais tipos de conversão existem?

Há quatro tipos principais de conversão, que são explicadas nas seções a seguir:

1. “Conversão de dados do clienteJMS” na página 169
2. “Conversão de Dados do Aplicativo” na página 170
3. “Conversão de dados do gerenciador de filas” na página 170
4. “Conversão de dados do canal de mensagens” na página 171

Onde a conversão deve ser executada?

A seção, “Escolha uma abordagem para a conversão de mensagem: receiver makes good” na página 171, descreve a abordagem comum de "receiver makes good". "Receiver makes good" também se aplica à conversão de dados do JMS.

Conversão de dados do clienteJMS

JMS client[1]A conversão de dados é a conversão de primitivas e objetos do Java em bytes em uma mensagem do JMS à medida que é enviada a um destino e a conversão de volta novamente, quando recebida. A conversão de dados do cliente JMS usa os métodos das classes `JMSMessage`. Os métodos estão listados pelo tipo de classe `JMSMessage` em Tabela 31 na página 172.

A conversão de e para a representação interna da JVM de números e de texto é executada para os métodos read, get, set e write. A conversão será executada quando a mensagem for enviada e quando qualquer um dos métodos read ou get for chamado em uma mensagem que tenha sido recebida.

A página de códigos e a codificação numérica usadas para gravar ou configurar o conteúdo de uma mensagem são definidas como atributos do destino. A página de códigos de destino e a codificação numérica podem ser mudadas de forma administrativa. Um aplicativo também pode substituir a página de códigos de destino e a codificação configurando as propriedades de mensagens que controlam o conteúdo da mensagem de configuração ou gravação.

Se você deseja converter a codificação do número quando uma mensagem `JMSBytesMessage` é enviada a um destino que não é definido como codificação `Native`, deve-se configurar a propriedade de mensagem `JMS_IBM_ENCODING` antes de enviar a mensagem. Se estiver seguindo o padrão "receiver makes good" ou se estiver trocando mensagens entre aplicativos JMS, o aplicativo não precisará configurar `JMS_IBM_ENCODING`. Na maioria dos casos, é possível deixar a propriedade `Encoding` como `Native`.

Para mensagens `JMSStreamMessage`, `JMSMapMessage` e `JMSTextMessage`, as propriedades com o identificador do conjunto de caracteres do destino são usadas. A codificação é ignorada em enviar como os números são gravados no formato de texto. O programa do aplicativo cliente JMS não precisará configurar `JMS_IBM_CHARACTER_SET` antes de enviar a mensagem se a propriedade do conjunto de caracteres de destino se aplicar.

Para obter os dados em uma mensagem, um aplicativo chama os métodos read ou get da mensagem do JMS. Os métodos se referem a página de códigos e a codificação definida no cabeçalho da mensagem anterior para criar as primitivas e os objetos do Java corretamente.

A conversão de dados do cliente JMS atenda às necessidades da maioria dos aplicativos JMS que estão trocando mensagens entre um cliente JMS e outro. Não codifique qualquer conversão de dados explícitos. Não use a classe `java.nio.charset.Charset`, que geralmente é usada ao gravar texto em um arquivo. Os métodos `writeString` e `setString` fazem a conversão para você.

Para obter mais detalhes sobre a conversão de dados do cliente JMS, consulte “Conversão e codificação de mensagem do clienteJMS” na página 182.

[1] "JMS Client" refere-se ao IBM MQ classes for JMS que implementa a interface JMS, que é executada no modo cliente ou de ligações.

Conversão de Dados do Aplicativo

Um aplicativo cliente JMS pode realizar a conversão de dados de caracteres explícitos usando a classe `java.nio.charset.Charset`; consulte os exemplos em Figura 14 na página 174 e Figura 15 na página 174. Os dados da sequência são convertidos em bytes, usando o método `getBytes` e enviados como bytes. Os bytes são convertidos novamente em texto usando um construtor `String` que usa uma matriz de bytes e uma `Charset`. Os dados de caracteres são convertidos usando os métodos `encode` e `decode` `Charset`. Geralmente a mensagem é enviada ou recebida como `JMSBytesMessage`, pois a parte da mensagem de um `JMSBytesMessage` não contém nada além dos dados gravados pelo aplicativo[2]. Também é possível enviar e receber bytes usando `JMSStreamMessage`, `JMSMapMessage` ou `JMSObjectMessage`.

Não há métodos do Java para codificar e decodificar bytes que contenham dados numéricos representados em diferentes formatos de codificação. Os dados numéricos são codificados e decodificados automaticamente usando os métodos numéricos read e write `JMSMessage`. Os métodos read e write usam o valor do atributo `JMS_IBM_ENCODING` dos dados da mensagem.

Um uso típico para conversão de dados do aplicativo será se um cliente JMS enviar ou receber uma mensagem formatada a partir de um aplicativo não JMS. Uma mensagem formatada contém dados numéricos, de texto e de bytes organizados pelo comprimento dos campos de dados. A menos que o aplicativo não JMS tenha especificado o formato de mensagem como "MQSTR ", a mensagem será construída como um `JMSBytesMessage`. Para receber dados da mensagem formatada em uma `JMSBytesMessage`, deve-se chamar uma sequência de métodos. Os métodos devem ser chamados na mesma ordem que os campos foram gravados na mensagem. Se os campos forem numéricos, deverá saber a codificação e o comprimento dos dados numéricos. Se algum dos campos contiver dados de byte ou de texto, deverá conhecer o comprimento de quaisquer dados de byte na mensagem. Há duas maneiras para converter uma mensagem formatada em um objeto do Java que é fácil de usar.

1. Construa uma classe Java que corresponda ao registro para conter a mensagem de leitura e gravação. O acesso aos dados no registro é com os métodos get e set da classe.
2. Construa uma classe Java correspondente ao registro estendendo a classe `com.ibm.mq.headers`. O acesso aos dados na classe é com os acessadores específicos de tipo no formato `getStringValue(`*`fieldname`*`);`

Consulte o “Trocando um registro formatado com um aplicativo não JMS” na página 190.

Conversão de dados do gerenciador de filas

A conversão da página de códigos pode ser executada pelo gerenciador de fila quando um programa cliente do JMS obtém uma mensagem A conversão é a mesma que a executada para um programa C. Um programa C configura `MQGMO_CONVERT` como uma opção de parâmetro `MQGET` `GetMsgOpts`; consulte Figura 13 na página 174. Um gerenciador de filas executará a conversão para um programa do cliente JMS que está recebendo uma mensagem, se a propriedade de destino `WMQ_RECEIVE_CONVERSION` for configurada como `WMQ_RECEIVE_CONVERSION_QMGR`. O programa do cliente JMS também poderá configurar a propriedade de destino; consulte Figura 12 na página 171.

[2] Uma exceção: dados gravados usando `writeUTF` começam com um campo de 2 bytes de comprimento

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Ou

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 12. Ativar conversão de dados do gerenciador de filas

O principal benefício da conversão do gerenciador de filas aparece ao trocar mensagens com aplicativos não JMS. Se o campo `Format` na mensagem for definido e o conjunto de caracteres de destino, ou codificação, for diferente para a mensagem, o gerenciador de filas executará a conversão de dados para o aplicativo de destino, se o aplicativo solicitá-la. O gerenciador de filas converte dados da mensagem formatados de acordo com um dos tipos de mensagens predefinidos IBM MQ , como um cabeçalho CICS bridge (`MQCIH`). Se o campo `Format` for definido pelo usuário, o gerenciador de filas procurará uma saída de conversão de dados com o nome fornecido no campo `Format`

A conversão de dados do gerenciador de filas é usada para melhorar o efeito com o design padrão "receiver makes good". Um cliente JMS de envio não é necessário para executar a conversão. Um programa de recebimento não JMS depende da saída de conversão para assegurar-se de que a mensagem seja entregue na codificação e na página de códigos necessárias. Com um cliente JMS de envio e receptor nãoJMS , o exemplo se aplica a IBM MQ.

É possível criar uma saída de conversão de dados, usando o utilitário de saída de conversão de dados, **`crtmqcvx`**, para permitir que o gerenciador de filas converta seus próprios dados formatados de registro. É possível construir seu próprio formato de registro, usar o `com.ibm.mq.headers` para acessá-lo como uma classe Java e usar sua própria saída de conversão para convertê-lo. No z/OS, o utilitário é chamado **`CSQUCVX`** e no IBM i, **`CVTMQMDTA`**. Consulte o “Trocando um registro formatado com um aplicativo não JMS” na página 190.

Conversão de dados do canal de mensagens

Os canais IBM MQ Sender, Server, Cluster-receiver e Cluster-sender possuem uma opção de conversão de mensagens, `CONVERT`. O conteúdo de uma mensagem poderá, opcionalmente, ser convertido quando uma mensagem for enviada. A conversão ocorre na extremidade de envio do canal. A definição do cluster-receiver é usada para definir automaticamente o canal cluster-sender correspondente.

A conversão de dados por canais de mensagem será geralmente usada se não for possível usar outras formas de conversão.

Escolha uma abordagem para a conversão de mensagem: "receiver makes good"

A abordagem usual no design do aplicativo IBM MQ para a conversão de código é "receiver makes good". "Receiver makes good" reduz o número de conversões de mensagens. Ele também evitará o problema de erros do canal inesperados se a conversão de mensagem falhar em algum gerenciador de filas intermediário durante a transferência de mensagens. A regra "receiver makes good" só será interrompida se houver algum motivo pelo qual não é possível efetuar o receiver makes good. A plataforma de recebimento pode não ter o conjunto de caracteres à direita, por exemplo.

"Receiver makes good" também é uma boa orientação geral para os aplicativos cliente JMS. Mas em casos específicos, a conversão para o conjunto de caracteres correto na origem pode ser mais eficiente. A conversão a partir da representação interna de JVM deverá ocorrer quando uma mensagem que contém tipos numéricos ou texto for enviada. A conversão para o conjunto de caracteres requerida pelo receptor, se o receptor não for um cliente JMS, poderá remover a necessidade do destinatário não JMS para

executar a conversão. Se o destinatário for um cliente JMS, ele vai ser convertido novamente, de qualquer forma, para decodificar os dados da mensagem e criar primitivas e objetos do Java.

A diferença entre os aplicativos do cliente JMS e os aplicativos gravados em uma linguagem como C, é que o Java deve executar a conversão de dados. Um aplicativo Java deve converter os números e texto a partir de sua representação interna para um formato codificado usado em mensagens.

Ao configurar o destino ou as propriedades da mensagem, será possível configurar o conjunto de caracteres e a codificação usada pelo IBM MQ para codificar os números e o texto em mensagens. Normalmente, você deixaria o conjunto de caracteres como `1208` e a codificação como `Native`.

IBM MQ não converte matrizes de bytes. Para codificar sequências e matrizes de caracteres em matrizes de byte, use o pacote `java.nio.charset`. `Charset` especifica o conjunto de caracteres usado para converter uma sequência ou matriz de caracteres em uma matriz de bytes. Também é possível decodificar uma matriz de bytes em uma matriz de sequência ou caracteres usando um `Charset`. Não é uma boa prática contar com o `java.nio.charset.Charset.defaultCodePage` ao codificar sequências e matrizes de caracteres. O padrão `Charset` é geralmente `windows-1252` no Windows e `UTF-8` no AIX and Linux. `windows-1252` é um conjunto de caracteres de byte único e `UTF-8` é um conjunto de caracteres multibyte.

Geralmente deixe o conjunto de caracteres de destino e as propriedades de codificação em seus valores padrão de `UTF-8` e `Native` ao trocar mensagens com outros aplicativos JMS. Se você estiver trocando mensagens que contêm números ou texto com um aplicativo JMS, escolha um dos tipos de mensagens `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage` ou `JMSObjectMessage` que se ajustam a sua finalidade. Não há outras tarefas conversão a serem feitas.

Se você estiver trocando mensagens com aplicativos não JMS que usam um formato de registro, isso será mais complicado. A menos que todo o registro contenha texto e possa ser transferido como uma `JMSTextMessage`, deve-se codificar e decodificar o texto no aplicativo. Configure o tipo de mensagem de destino como `MQe` use `JMSBytesMessage` para evitar que o IBM MQ classes for JMS inclua informações de identificação e cabeçalhos adicionais nos dados da mensagem. Use os métodos `JMSBytesMessage` para gravar números e bytes, e a classe `Charset` converte texto em matrizes de bytes explicitamente. Um número de fatores podem influenciar na sua escolha do conjunto de caracteres:

- Desempenho: é possível reduzir o número de conversões transformando texto em um conjunto de caracteres usado no maior número de servidores?
- Uniformidade: transfira todas as mensagens no mesmo conjunto de caracteres.
- Abundância: quais conjuntos de caracteres têm todos os pontos de código que os aplicativos devem usar?
- Simplicidade: os conjuntos de caracteres de byte único são mais simples de usar do que o comprimento variável e os conjuntos de caracteres multibyte.

Consulte o “Trocando um registro formatado com um aplicativo não JMS” na página 190. para obter exemplos de converter mensagens trocadas com aplicativos não JMS.

Exemplos

Tabela de tipos de mensagens e tipos de conversões

Tabela 31. Tipos de mensagens e tipos de conversão

	Tipo de conversão			
Tipo de mensagem	**Texto**	**Numérico**	**Outro**	**Nenhum**
`JMSObjectMessage`				`getObject` `setObject`

Tabela 31. Tipos de mensagens e tipos de conversão (continuação)

	Tipo de conversão			
Tipo de mensagem	**Texto**	**Numérico**	**Outro**	**Nenhum**
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Figura 13. Snippet de código de amqsget0.c

Enviando e recebendo texto em uma JMSBytesMessage

O código em Figura 14 na página 174 envia uma sequência em uma BytesMessage. Para simplicidade, o exemplo envia uma sequência única, para a qual uma JMSTextMessage é mais apropriada. Para receber uma sequência de texto em mensagem de bytes que contêm uma mistura de tipos, deve-se saber o comprimento da sequência em bytes, chamado *TEXT_LENGTH* no Figura 15 na página 174. Mesmo para uma sequência com um número fixo de caracteres, o comprimento da representação de bytes pode ser maior.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                     .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Figura 14. Enviando uma String em um JMSBytesMessage

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Figura 15. Recebendo um String de um JMSBytesMessage

Conceitos relacionados

Conversão e codificação de mensagem do clienteJMS
Os métodos usados para conversão e codificação de mensagem do cliente JMS são listados com exemplos de códigos de cada tipo de conversão.

Conversão de dados do gerenciador de filas

A conversão de dados do gerenciador de filas sempre esteve disponível para aplicativos não JMS recebendo mensagens de clientes do JMS. Os clientes JMS que recebem mensagens também usam a conversão de dados do gerenciador de fila, que é opcional.

Tarefas relacionadas

Trocando um registro formatado com um aplicativo não JMS
Siga as etapas sugeridas nesta tarefa para projetar e construir uma saída de conversão de dados e um aplicativo cliente do JMS que podem trocar mensagens com um aplicativo não JMS usando `JMSBytesMessage`. A troca de uma mensagem formatada com um aplicativo não JMS pode ocorrer com ou sem chamar uma saída de conversão de dados.

Referências relacionadas

Conversão e tipos de mensagem doJMS
A opção de tipo de mensagem afeta sua abordagem de conversão de mensagens. A interação de conversão de mensagem e o tipo de mensagem são descritos para os tipos de mensagens do JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` e `JMSBytesMessage`.

Conversão e tipos de mensagem doJMS
A opção de tipo de mensagem afeta sua abordagem de conversão de mensagens. A interação de conversão de mensagem e o tipo de mensagem são descritos para os tipos de mensagens do JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` e `JMSBytesMessage`.

JMSObjectMessage

`JMSObjectMessage` contém um objeto e quaisquer objetos que ele faz referência, serializado em um fluxo de bytes pela JVM. O texto é serializado em `UTF-8` e limitado para sequências ou matrizes de caracteres que não ultrapasse 65.534 bytes. Uma vantagem de `JMSObjectMessage` é que os aplicativos não estejam envolvidos em quaisquer problemas de conversão de dados, contanto que eles usem apenas os métodos e os atributos do objeto. `JMSObjectMessage` fornece a conversão de dados para objetos complexos sem o programador de aplicativos considerando como codificar um objeto em uma mensagem. A desvantagem de usar o `JMSObjectMessage` é que ele pode ser trocado apenas com outros aplicativos JMS. Ao escolher um dos outros tipos de mensagens do JMS, será possível trocar mensagens do JMS com aplicativos não JMS.

"Enviando e recebendo uma JMSObjectMessage" na página 178 mostra um objeto `String` que está sendo trocado em uma mensagem.

Um aplicativo cliente JMS pode receber um `JMSObjectMessage` apenas em uma mensagem que tem um corpo de estilo do JMS. O destino deve especificar um corpo de estilo do JMS.

JMSTextMessage

`JMSTextMessage` contém uma sequência de texto única. Quando uma mensagem de texto for enviada, o texto `Format` será configurado como `"MQSTR   "`, `WMQConstants.MQFMT_STRING`. O `CodedCharacterSetId` do texto é configurado para o identificador do conjunto de caracteres codificados definido para seu destino. O texto é codificado no `CodedCharacterSetId` por IBM MQ. Os campos `CodedCharacterSetId` e `Format` é um conjunto no descritor de mensagens, `MQMD` ou nos campos do JMS em um `MQRFH2`. Se a mensagem for definida como tendo um estilo do corpo da mensagem `WMQ_MESSAGE_BODY_MQ` ou o estilo de corpo não estiver especificado, mas o destino for `WMQ_TARGET_DEST_MQ` então os campos do descritor de mensagens serão configurados. Caso contrário, a mensagem possui um JMS RFH2 e os campos serão configurados na parte fixa do `MQRFH2`.

Um aplicativo pode substituir o identificador do conjunto de caracteres codificados definidos para um destino. Ele deve configurar a propriedade de mensagem `JMS_IBM_CHARACTER_SET` para um identificador do conjunto de caracteres codificados; consulte o exemplo em "Enviando e recebendo um JMSTextmessage" na página 178.

Quando o cliente JMS chamar o conversão do gerenciador de filas do método `consumer.receive` será opcional. A conversão do gerenciador de filas é ativado configurando a propriedade de destino `WMQ_RECEIVE_CONVERSION` para `WMQ_RECEIVE_CONVERSION_QMGR`. O gerenciador de filas converte a mensagem de texto do `JMS_IBM_CHARACTER_SET` especificado para a mensagem antes de ser

transferida para o cliente JMS. O conjunto de caracteres da mensagem convertida é 1208, UTF-8, a menos que o destino tenha um WMQ_RECEIVE_CCSID diferente. O CodedCharacterSetId na mensagem que se refere à JMSTextMessage é atualizado para o ID do conjunto de caracteres de destino. O texto é decodificado a partir do conjunto de caracteres de destino em Unicode pelo método getText; consulte o exemplo em "Enviando e recebendo um JMSTextmessage" na página 178.

Um JMSTextMessage pode ser enviado em um corpo da mensagem no estilo MQ, sem um cabeçalho do JMS MQRFH2 O valor dos atributos de destino, WMQ_MESSAGE_BODY e WMQ_TARGET_DEST determinam o estilo de corpo da mensagem, a menos que substituído pelo aplicativo. O aplicativo pode substituir os valores configurados no destino, chamando `destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ)` ou `destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)`.

Se você enviar uma JMSTextMessage com um corpo de estilo MQ enviando-a para um destino com WMQ_MESSAGE_BODY configurado como WMQ_MESSAGE_BODY_MQ, não será possível recebê-la, como uma JMSTextMessage a partir do mesmo destino. Todas as mensagens recebidas a partir de um destino com WMQ_MESSAGE_BODY configurado como WMQ_MESSAGE_BODY_MQ são recebidas como uma JMSBytesMessage. Se você tentar receber a mensagem como um JMSTextMessage , isso causará uma exceção, `ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage`

Nota: O texto em uma JMSBytesMessage não é convertido pelo cliente JMS. O cliente pode receber apenas o texto na mensagem como uma matriz de bytes. Se a conversão do gerenciador de filas estiver ativada, o texto será convertido pelo gerenciador de filas, mas o cliente JMS ainda deverá recebê-lo como uma matriz de bytes em uma JMSBytesMessage.

Geralmente é melhor usar a propriedade WMQ_TARGET_DEST para controlar se uma JMSTextMessage é enviada com um estilo de corpo MQ ou JMS. É possível, então, receber a mensagem de um destino que tenha WMQ_TARGET_DEST configurado como WMQ_TARGET_DEST_MQ ou WMQ_TARGET_DEST_JMS. WMQ_TARGET_DEST não tem efeito no receptor.

JMSMapMessage e JMSStreamMessage

Estes dois tipos de mensagens do JMS são semelhantes. É possível ler e gravar os tipos primitivos para as mensagens usando métodos baseados nas interfaces DataInputStream e DataOutputStream; consulte "Tabela de tipos de mensagens e tipos de conversões" na página 180. Os detalhes são descritos em "Conversão e codificação de mensagem do clienteJMS" na página 182. Cada primitivo é identificado; consulte "O corpo da mensagem do JMS" na página 166.

Os dados numéricos são lidos e gravados na mensagem codificada como texto XML. Nenhuma referência é feita para a propriedade de destino, JMS_IBM_ENCODING. Os dados de texto são tratados da mesma forma que o texto em uma JMSTextMessage. Se você fosse observar os conteúdos da mensagem criada pelo exemplo em Figura 20 na página 179, todos os dados da mensagem seria em EBCDIC, conforme ela foi enviada com um valor do conjunto de caracteres de 37.

É possível enviar vários itens em uma JMSMapMessage ou JMSStreamMessage.

É possível recuperar os itens de dados individuais pelo nome a partir de uma JMSMapMessage ou pela posição a partir de uma JMSStreamMessage. Cada item será decodificado quando um método get ou read for chamado usando o valor CodedCharacterSetId armazenado na mensagem. Se o método usado para recuperar o item retornar um tipo diferente para o tipo que foi enviado, o tipo será convertido. Se o tipo não puder ser convertido, uma exceção será lançada. Consulte Class JMSStreamMessage para obter detalhes. O exemplo em "Enviando dados em uma JMSStreamMessage e JMSMapMessage" na página 179 ilustra a conversão de tipo e obtenção do conteúdo JMSMapMessage fora de sequÊncia.

O campo MQRFH2.format para o JMSMapMessage e JMSStreamMessage é configurado como "MQSTR ". Se a propriedade de destino WMQ_RECEIVE_CONVERSION for configurada como WMQ_RECEIVE_CONVERSION_QMGR, os dados da mensagem serão convertidos pelo gerenciador de filas antes de serem enviados para o cliente JMS. O MQRFH2.CodedCharacterSetId da mensagem é o WMQ_RECEIVE_CCSID do destino. O MQRFH2.Encoding é Native. Se WMQ_RECEIVE_CONVERSION for WMQ_RECEIVE_CONVERSION_CLIENT_MSG, o CodedCharacterSetId e Encoding do MQRFH2 será o valor configurado pelo emissor.

Um aplicativo cliente JMS pode receber uma JMSMapMessage ou JMSStreamMessage apenas em uma mensagem que tenha um corpo de estilo do JMS e a partir de um destino que não especifica um corpo de estilo MQ.

JMSBytesMessage

Uma JMSBytesMessage pode conter vários tipos primitivos. É possível ler e gravar os tipos primitivos para as mensagens usando métodos baseados nas interfaces DataInputStream e DataOutputStream; consulte "Tabela de tipos de mensagens e tipos de conversões" na página 180. Os detalhes são descritos em "Conversão e tipos de mensagem doJMS" na página 175.

A codificação dos dados numéricos na mensagem é controlada pelo valor de JMS_IBM_ENCODING, que é configurado antes de gravar dados numéricos para o JMSBytesMessage. Um aplicativo pode substituir a codificação padrão Native definida para JMSBytesMessage configurando a propriedade de mensagem JMS_IBM_ENCODING.

Os dados de texto podem ser lidos e gravados em UTF-8 usando o readUTF e writeUTF ou em Unicode usando os métodos readChar e writeChar. Não há métodos que usem o CodedCharacterSetId. Como alternativa, o cliente JMS pode codificar e decodificar o texto em bytes usando a classe Charset. Ele transfere os bytes entre a JVM e a mensagem sem que o IBM MQ classes for JMS execute nenhuma conversão; consulte "Enviando e recebendo texto em uma JMSBytesMessage" na página 179.

Um JMSBytesMessage enviado para um aplicativo MQ geralmente é enviado em um corpo de mensagem no estilo MQ, sem um cabeçalho JMS MQRFH2 Se for enviado a um aplicativo JMS, o estilo do corpo da mensagem geralmente será JMS. O valor dos atributos de destino, WMQ_MESSAGE_BODY e WMQ_TARGET_DEST determinam o estilo de corpo da mensagem, a menos que substituído pelo aplicativo. O aplicativo pode substituir os valores configurados no destino, chamando destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) ou destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Se você enviar uma JMSBytesMessage com um corpo de estilo MQ, será possível receber a mensagem a partir de um destino que define um estilo de corpo de mensagem do JMS ou um MQ. Se você enviar uma JMSBytesMessage com um corpo de estilo JMS, então deverá receber a mensagem a partir de um destino que define um estilo de corpo da mensagem do JMS. Se você não enviar, o MQRFH2 será tratado como parte dos dados da mensagem do usuário, o que pode não ser o que você está esperando.

Se uma mensagem tiver um estilo de corpo do JMS ou um MQ, a maneira como ela é recebida não será afetada pela configuração de WMQ_TARGET_DEST.

A mensagem poderá ser transformada posteriormente, pelo gerenciador de filas, se um Format for fornecido para os dados da mensagem e a conversão de dados do gerenciador de filas estiver ativada. Não use o campo de formato para nada além de especificar o formato dos dados da mensagem ou deixe-o em branco, MQConstants.MQFMT_NONE

É possível enviar vários itens em uma JMSBytesMessage. Cada item numérico será convertido quando a mensagem for enviada usando a codificação definida para a mensagem.

É possível recuperar os itens de dados individuais a partir de JMSBytesMessage. Chame os métodos read na mesma ordem que os métodos write foram chamados para criar a mensagem. Cada item numérico será convertido quando a mensagem for chamada usando o valor Encoding armazenado na mensagem.

Diferentemente de JMSMapMessage e JMSStreamMessage, JMSBytesMessage contém somente dados gravados pelo aplicativo. Nenhum dado adicional é armazenado nos dados da mensagem, como as identificações XML usadas para definir os itens em uma JMSMapMessage e JMSStreamMessage. Por esse motivo, use JMSBytesMessage para transferir mensagens formatadas para outros aplicativos.

A conversão entre JMSBytesMessage e DataInputStream e DataOutputStream é útil em alguns aplicativos. O código com base no exemplo, "Lendo e gravando mensagens usando DataInputStream e DataOutputStream" na página 179, é necessário usar o pacote com.ibm.mq.header com o JMS.

Exemplos

Enviando e recebendo uma JMSObjectMessage

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

Figura 16. Enviando e recebendo uma JMSObjectMessage

Enviando e recebendo um JMSTextmessage

Uma mensagem de texto não pode conter texto em conjuntos de caracteres diferentes. O exemplo mostra texto em conjuntos de caracteres diferentes, enviado em duas mensagens diferentes.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figura 17. Enviar mensagem de texto no conjunto de caracteres definido pelo destino

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figura 18. Enviar mensagem de texto em ccsid 37

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figura 19. Receber mensagem de texto

Enviando dados em uma JMSStreamMessage e JMSMapMessage

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                         " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                       " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

Figura 20. Envie dados em JMSStreamMessage e JMSMapMessage

Enviando e recebendo texto em uma JMSBytesMessage

O código em Figura 21 na página 179 envia uma sequência em uma BytesMessage. Para simplicidade, o exemplo envia uma sequência única, para a qual uma JMSTextMessage é mais apropriada. Para receber uma sequência de texto em mensagem de bytes que contêm uma mistura de tipos, deve-se saber o comprimento da sequência em bytes, chamado *TEXT_LENGTH* no Figura 22 na página 179. Mesmo para uma sequência com um número fixo de caracteres, o comprimento da representação de bytes pode ser maior.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Figura 21. Enviando uma String em um JMSBytesMessage

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Figura 22. Recebendo um String de um JMSBytesMessage

Lendo e gravando mensagens usando DataInputStream e DataOutputStream

O código em Figura 23 na página 180 cria um JMSBytesMessage usando um DataOutputStream.

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

Figura 23. Envie uma JMSBytesMessage usando um DataOutputStream

A instrução que configura a propriedade JMS_IBM_ENCODING é comentada. A instrução será válida, se estiver sendo gravada diretamente em uma JMSBytesMessage, mas não entrará em vigor ao gravar o DataOutputStream. Os números gravados no DataOutputStream são codificados na codificação Native. Configurar JMS_IBM_ENCODING não tem efeito.

O código em Figura 24 na página 180 recebe uma JMSBytesMessage usando um DataInputStream.

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

Figura 24. Receba uma JMSBytesMessage usando um DataInputStream

A página de códigos é impressa usando a propriedade de página de códigos dos dados da mensagem de entrada, JMS_IBM_CHARACTER_SET. Na entrada, JMS_IBM_CHARACTER_SET é uma página de códigos Java e não um identificador de conjunto de caracteres codificados numéricos.

Tabela de tipos de mensagens e tipos de conversões

Tabela 32. Tipos de mensagens e tipos de conversão

	Tipo de conversão			
Tipo de mensagem	**Texto**	**Numérico**	**Outro**	**Nenhum**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

Tabela 32. Tipos de mensagens e tipos de conversão (continuação)

	Tipo de conversão			
Tipo de mensagem	**Texto**	**Numérico**	**Outro**	**Nenhum**
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Conceitos relacionados

Abordagens de conversão de mensagem doJMS
Um número de abordagens de conversão de dados é aberto para designers de aplicativos JMS. Essas abordagens não são exclusivas; alguns aplicativos devem usar uma combinação dessas abordagens. Se seu aplicativo estiver trocando somente texto ou mensagens com outros aplicativos JMS, não considere a conversão de dados normalmente. A conversão de dados é executada automaticamente para você, por IBM MQ.

Conversão e codificação de mensagem do clienteJMS
Os métodos usados para conversão e codificação de mensagem do cliente JMS são listados com exemplos de códigos de cada tipo de conversão.

Conversão de dados do gerenciador de filas
A conversão de dados do gerenciador de filas sempre esteve disponível para aplicativos não JMS recebendo mensagens de clientes do JMS. Os clientes JMS que recebem mensagens também usam a conversão de dados do gerenciador de fila, que é opcional.

Tarefas relacionadas

Trocando um registro formatado com um aplicativo não JMS
Siga as etapas sugeridas nesta tarefa para projetar e construir uma saída de conversão de dados e um aplicativo cliente do JMS que podem trocar mensagens com um aplicativo não JMS usando `JMSBytesMessage`. A troca de uma mensagem formatada com um aplicativo não JMS pode ocorrer com ou sem chamar uma saída de conversão de dados.

Conversão e codificação de mensagem do clienteJMS
Os métodos usados para conversão e codificação de mensagem do cliente JMS são listados com exemplos de códigos de cada tipo de conversão.

Conversão e codificação ocorrem quando as primitivas ou objetos Java são lidos ou escritos em mensagens do JMS e a partir delas. A conversão é chamada de conversão de dados do cliente JMS para distingui-la da conversão de dados do gerenciador de filas e a conversão de dados do aplicativo. A conversão ocorre estritamente quando dados são lidos ou gravados em uma mensagem do JMS. O texto é convertido em/de uma representação Unicode interna de 16 bits[3]para o conjunto de caracteres usado para texto em mensagens. Dados numéricos e tipos numéricos de primitivas Java são convertidos para a codificação definida para a mensagem. Se a conversão é executada e que tipo de conversão é executado dependem do tipo de mensagem do JMS e da operação de leitura ou gravação.

Tabela 33 na página 182 categoriza os métodos de leitura e gravação para diferentes tipos de mensagens do JMS pelo tipo de conversão executado. Os tipos de conversões estão descritos no texto após a tabela.

Tabela 33. Tipos de mensagens e tipos de conversão

	Tipo de conversão			
Tipo de mensagem	**Texto**	**Numérico**	**Outro**	**Nenhum**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

`JMSBytesMessage`	`readUTF`
	`writeUTF`
		`readDouble`
		`readFloat`
		`readInt`
		`readLong`
		`readShort`
		`readUnsignedShort`
		`writeDouble`
		`writeFloat`
		`writeInt`
		`writeLong`
		`writeShort`
			`readBoolean`
			`readObject`
			`writeBoolean`
			`writeObject`
				`readByte`
				`readUnsignedByte`
				`readBytes`
				`readChar`
				`writeByte`
				`writeBytes`
				`writeChar`

[3] Algumas representações Unicode requerem mais de 16 bits. Consulte uma referência do Java SE.

Tabela 33. Tipos de mensagens e tipos de conversão (continuação)				
	Tipo de conversão			
Tipo de mensagem	**Texto**	**Numérico**	**Outro**	**Nenhum**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Texto

O CodedCharacterSetId padrão para um destino é 1208, UTF-8. Por padrão, o texto é convertido de Unicode e enviado como uma sequência de texto UTF-8. No recebimento, o texto é convertido do conjunto de caracteres codificados na mensagem recebida pelo cliente para Unicode.

Os métodos setText e writeString convertem texto de Unicode no conjunto de caracteres definido para o destino. Um aplicativo pode substituir o conjunto de caracteres de destino configurando a propriedade de mensagem JMS_IBM_CHARACTER_SET. Ao enviar uma mensagem, JMS_IBM_CHARACTER_SET deve ser um identificador de conjunto de caracteres codificados numéricos[4].

Os segmentos de código em "Enviando e recebendo um JMSTextmessage" na página 186 enviam duas mensagens. Uma é enviada no conjunto de caracteres definido para o destino e a outra no conjunto de caracteres 37 definido pelo aplicativo.

Os métodos getText e readString convertem o texto na mensagem do conjunto de caracteres definido na mensagem para Unicode. Os métodos usam a página de códigos definida na propriedade de mensagem, JMS_IBM_CHARACTER_SET. A página de códigos é mapeada de MQRFH2.CodedCharacterSetId a menos que a mensagem seja uma mensagem do tipo MQ e não tenha nenhum MQRFH2. Se a mensagem for uma mensagem do tipo MQ, sem nenhum MQRFH2, a página de códigos será mapeada de MQMD.CodedCharacterSetId.

O fragmento de código em Figura 29 na página 186 recebe a mensagem que foi enviada para o destino. O texto na mensagem é convertido da página de códigos IBM037 de volta para Unicode.

Nota: Uma maneira simples de verificar se o texto é convertido para o conjunto de caracteres codificados 37 é usar o IBM MQ Explorer. Procure a fila e mostre as propriedades da mensagem antes que ela seja recuperada.

[4] Ao receber uma mensagem, JMS_IBM_CHARACTER_SET é um nome de página de código Java Charset.

Compare o fragmento de código em Figura 28 na página 186 ao fragmento de código incorreto em Figura 25 na página 184. No fragmento incorreto, a sequência de texto é convertida duas vezes, uma vez pelo aplicativo e novamente pelo IBM MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Figura 25. Conversão de página de códigos incorreta

O método `writeUTF` converte texto de Unicode para 1208, UTF-8. A sequência de texto tem como prefácio um comprimento de 2 bytes. O comprimento máximo da sequência de texto é 65534 bytes. O método `readUTF` lê um item em uma mensagem gravada pelo método `writeUTF`. Ele lê exatamente o número de bytes gravados pelo método `writeUTF`.

Numérico

A codificação numérica padrão para um destino é `Native`. A constante da codificação `Native` para Java tem o valor 273, `x'00000111'`, que é o mesmo para todas as plataformas. No recebimento, os números na mensagem são convertidos corretamente em primitivas Java numéricas. A transformação usa a codificação definida na mensagem e o tipo retornado pelo método de leitura.

O método de envio converte números que são incluídos em uma mensagem por `set` e `write` para a codificação numérica definida para o destino. A codificação de destino pode ser substituída para uma mensagem por um aplicativo que esteja configurando a propriedade de mensagem, `JMS_IBM_ENCODING`; por exemplo:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

Os métodos numéricos `get` e `read` convertem números na mensagem da codificação numérica definida na mensagem. Eles convertem os números para o tipo que especificado pelo método `read` ou `get`; consulte A propriedade ENCODING. Os métodos usam a codificação definida em `JMS_IBM_ENCODING`. A codificação é mapeada de `MQRFH2.Encoding`, a menos que a mensagem seja uma mensagem do tipo MQ e não tenha nenhum `MQRFH2`. Se a mensagem for uma mensagem do tipo MQ, sem nenhum `MQRFH2`, então, os métodos usam a codificação definida em `MQMD.Encoding`.

O exemplo em Figura 30 na página 186 mostra um aplicativo codificando um número no formato de destino e enviando-o em um `JMSStreamMessage`. Compare o exemplo em Figura 30 na página 186 ao exemplo em Figura 31 na página 187. A diferença é que `JMS_IBM_ENCODING` deve ser configurado em um `JMSBytesMessage`.

Nota: Uma maneira simples para verificar se o número está codificado corretamente é usar o IBM MQ Explorer. Procure na fila e mostre as propriedades da mensagem antes de ser consumida.

Outro

Os métodos `boolean` codificam `true` e `false` como `x'01'` e `x'00'` em um `JMSByteMessage`, `JMSStreamMessage` e `JMSMapMessage`.

Os métodos UTF codificam e decodificam Unicode para sequências de texto UTF-8. As sequências são limitadas a menos de 65536 caracteres e são precedidas pelo campo de comprimento de 2 bytes.

Os métodos Object agrupam tipos primitivos como objetos. Tipos numéricos e de texto são codificados ou convertidos como se os tipos primitivos tivessem sido lidos ou gravados usando os métodos numérico e de texto.

Nenhum

Os métodos `readByte`, `readBytes`, `readUnsignedByte`, `writeByte` e `writeBytes` efetuam get ou put de bytes únicos, ou matrizes de bytes, entre o aplicativo e a mensagem sem conversão. Os

métodos `readChar` e `writeChar` efetuam get e put caracteres Unicode de 2 bytes entre o aplicativo e a mensagem sem conversão.

Usando os métodos `readBytes` e `writeBytes`, o aplicativo pode executar sua própria conversão de ponto de código, como em "Enviando e recebendo texto em uma JMSBytesMessage" na página 187.

O IBM MQ não executa nenhuma conversão de página de códigos no cliente, já que a mensagem é um `JMSBytesMessage` e porque os métodos `readBytes` e `writeBytes` são usados. Contudo, se os bytes representarem texto, certifique-se de que a página de códigos usada pelo aplicativo corresponda ao conjunto de caracteres codificados do destino. A mensagem pode ser convertida novamente por uma saída de conversão do gerenciador de filas. Outra possibilidade é que o cliente receptor do JMS pode seguir a convenção de converter qualquer matriz de bytes que representa o texto na mensagem em sequências ou caracteres que usam a propriedade `JMS_IBM_CHARACTER_SET` na mensagem.

Neste exemplo, o cliente usa o conjunto de caracteres codificados de destino para sua conversão:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Como alternativa, o cliente pode ter escolhido uma página de códigos e, em seguida, configurado o conjunto de caracteres codificados correspondente na propriedade `JMS_IBM_CHARACTER_SET` da mensagem. O IBM MQ classes for Java usa `JMS_IBM_CHARACTER_SET` para configurar o campo `CodedCharacterSetId` nas propriedades do JMS no `MQRFH2` ou no descritor de mensagens `MQMD`:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

Se uma matriz de bytes for gravada em um `JMSStringMessage` ou `JMSMapMessage`, o IBM MQ classes for JMS não executa conversão de dados, pois os bytes são digitados como dados hexadecimais, não como texto no `JMSStringMessage` e `JMSMapMessage`.

Se os bytes representarem caracteres em seu aplicativo, deve-se levar em consideração quais pontos de código ler e gravar na mensagem. O código em Figura 26 na página 185 segue a convenção de usar o conjunto de caracteres codificados de destino. Se você criar a sequência usando o conjunto de caracteres padrão para a JVM, os conteúdos de bytes dependem da plataforma. Uma JVM no Windows geralmente tem um `Charset` padrão de `windows-1252` e AIX and Linux tem `UTF-8`. A troca entre o Windows e o AIX and Linux requer a seleção de uma página de códigos explícita para trocar texto como bytes.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Figura 26. Gravando bytes que representam uma sequência em uma JMS`StreamMessage` usando o conjunto de caracteres de destino

Exemplos

[5] `SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage)` currently accepts only numeric character set identifiers.

Enviando e recebendo um JMSTextmessage

Uma mensagem de texto não pode conter texto em conjuntos de caracteres diferentes. O exemplo mostra texto em conjuntos de caracteres diferentes, enviado em duas mensagens diferentes.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figura 27. Enviar mensagem de texto no conjunto de caracteres definido pelo destino

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figura 28. Enviar mensagem de texto em ccsid 37

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figura 29. Receber mensagem de texto

Exemplos de codificação

Exemplos mostrando um número que está sendo enviado na codificação definida para um destino. Observe que se deve configurar a propriedade JMS_IBM_ENCODING de um JMSBytesMessage para o valor especificado para o destino.

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

Figura 30. Enviando um número usando a codificação de destino em um JMSStreamMessage

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

Figura 31. Enviando um número usando a codificação de destino em um JMSBytesMessage

Enviando e recebendo texto em uma JMSBytesMessage

O código em Figura 32 na página 187 envia uma sequência em uma BytesMessage. Para simplicidade, o exemplo envia uma sequência única, para a qual uma JMSTextMessage é mais apropriada. Para receber uma sequência de texto em mensagem de bytes que contêm uma mistura de tipos, deve-se saber o comprimento da sequência em bytes, chamado *TEXT_LENGTH* no Figura 33 na página 187. Mesmo para uma sequência com um número fixo de caracteres, o comprimento da representação de bytes pode ser maior.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Figura 32. Enviando uma String em um JMSBytesMessage

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Figura 33. Recebendo um String de um JMSBytesMessage

Conceitos relacionados

Abordagens de conversão de mensagem doJMS

Um número de abordagens de conversão de dados é aberto para designers de aplicativos JMS. Essas abordagens não são exclusivas; alguns aplicativos devem usar uma combinação dessas abordagens. Se seu aplicativo estiver trocando somente texto ou mensagens com outros aplicativos JMS, não considere a conversão de dados normalmente. A conversão de dados é executada automaticamente para você, por IBM MQ.

Conversão de dados do gerenciador de filas

A conversão de dados do gerenciador de filas sempre esteve disponível para aplicativos não JMS recebendo mensagens de clientes do JMS. Os clientes JMS que recebem mensagens também usam a conversão de dados do gerenciador de fila, que é opcional.

Tarefas relacionadas

Trocando um registro formatado com um aplicativo não JMS

Siga as etapas sugeridas nesta tarefa para projetar e construir uma saída de conversão de dados e um aplicativo cliente do JMS que podem trocar mensagens com um aplicativo não JMS usando `JMSBytesMessage`. A troca de uma mensagem formatada com um aplicativo não JMS pode ocorrer com ou sem chamar uma saída de conversão de dados.

Referências relacionadas

Conversão e tipos de mensagem doJMS
A opção de tipo de mensagem afeta sua abordagem de conversão de mensagens. A interação de conversão de mensagem e o tipo de mensagem são descritos para os tipos de mensagens do JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` e `JMSBytesMessage`.

Conversão de dados do gerenciador de filas
A conversão de dados do gerenciador de filas sempre esteve disponível para aplicativos não JMS recebendo mensagens de clientes do JMS. Os clientes JMS que recebem mensagens também usam a conversão de dados do gerenciador de fila, que é opcional.

O gerenciador de filas pode converter dados de caracteres e numéricos em dados da mensagem usando os valores de `CodedCharacterSetId`, `Encoding` e `Format` configurados para os dados da mensagem. Para aplicativos não JMS, o recurso de conversão sempre esteja disponível configurando `GetMessageOption`, `GMO_CONVERT`.

O gerenciador de filas pode converter mensagens que são enviadas para clientes JMS . A conversão do gerenciador de filas é controlada configurando a propriedade de destino, `WMQ_RECEIVE_CONVERSION`, como `WMQ_RECEIVE_CONVERSION_QMGR`ou `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`. O aplicativo pode mudar a configuração de destino:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Ou

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 34. Ativar conversão de dados do gerenciador de filas

Conversão de dados do gerenciador de filas para um cliente JMS ocorre quando o cliente chama um método `consumer.receive`. Dados de texto são transformados em UTF-8 (1208) por padrão. Métodos read e get subsequente decodificam texto nos dados recebidos de UTF-8, criando primitivas de texto Java em sua codificação Unicode interna. UTF-8 não é o único conjunto de caracteres de destino de conversão de dados do gerenciador de filas. É possível escolher um CCSID diferente configurando a propriedade de destino `WMQ_RECEIVE_CCSID`.

Um aplicativo também pode mudar a configuração de destino, por exemplo, configurando-o para 437, DOS-US:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

Ou

```
((MQDestination)destination).setReceiveCCSID(437);
```

Figura 35. Configurar o conjunto de caracteres codificado de destino para a conversão do gerenciador de filas

A razão para mudar `WMQ_RECEIVE_CCSID` é especializada; o CCSID escolhido não faz diferença para os objetos de texto criados na JVM. No entanto, algumas JVMs, em algumas plataformas, podem não ser capazes de manipular a conversão do CCSID de texto na mensagem para Unicode. A opção fornece uma opção de CCSID para qualquer texto entregue ao cliente na mensagem. Algumas plataformas clientes JMS têm tido problemas com o texto da mensagem que está sendo entregue em UTF-8.

O código JMS é equivalente ao texto em negrito no código C em Figura 36 na página 189,

```
gmo.Options = MQGMO_WAIT         /* wait for new messages            */
            | MQGMO_NO_SYNCPOINT /* no transaction                   */
            | MQGMO_CONVERT;     /* convert if necessary             */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle                */
           Hobj,                /* object handle                    */
           &md,                 /* message descriptor               */
           &gmo,                /* get message options              */
           buflen,              /* buffer length                    */
           buffer,              /* message buffer                   */
           &messlen,            /* message length                   */
           &CompCode,           /* completion code                  */
           &Reason);            /* reason code                      */
```

Figura 36. Snippet de código de `amqsget0.c`

Nota:

A conversão do gerenciador de filas é executada apenas nos dados de mensagem que possuem um formato IBM MQ conhecido `MQSTR`, ou `MQCIH` são exemplos de formatos conhecidos predefinidos. Um formato conhecido também pode ser um formato definido pelo usuário, contanto que você tenha fornecido uma saída de conversão de dados.

As mensagens construídas como `JMSTextMessage`, `JMSMapMessage` e `JMSStreamMessage` têm um formato `MQSTR` e podem ser convertidas pelo gerenciador de filas.

Conceitos relacionados

Abordagens de conversão de mensagem doJMS

Um número de abordagens de conversão de dados é aberto para designers de aplicativos JMS. Essas abordagens não são exclusivas; alguns aplicativos devem usar uma combinação dessas abordagens. Se seu aplicativo estiver trocando somente texto ou mensagens com outros aplicativos JMS, não considere a conversão de dados normalmente. A conversão de dados é executada automaticamente para você, por IBM MQ.

Conversão e codificação de mensagem do clienteJMS

Os métodos usados para conversão e codificação de mensagem do cliente JMS são listados com exemplos de códigos de cada tipo de conversão.

"Chamando a saída de conversão de dados" na página 996
Uma saída de conversão de dados é uma saída escrita pelo usuário que recebe controle durante o processamento de uma chamada MQGET.

Tarefas relacionadas

Trocando um registro formatado com um aplicativo não JMS
Siga as etapas sugeridas nesta tarefa para projetar e construir uma saída de conversão de dados e um aplicativo cliente do JMS que podem trocar mensagens com um aplicativo não JMS usando `JMSBytesMessage`. A troca de uma mensagem formatada com um aplicativo não JMS pode ocorrer com ou sem chamar uma saída de conversão de dados.

Referências relacionadas

Conversão e tipos de mensagem doJMS
A opção de tipo de mensagem afeta sua abordagem de conversão de mensagens. A interação de conversão de mensagem e o tipo de mensagem são descritos para os tipos de mensagens do JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` e `JMSBytesMessage`.

Trocando um registro formatado com um aplicativo não JMS
Siga as etapas sugeridas nesta tarefa para projetar e construir uma saída de conversão de dados e um aplicativo cliente do JMS que podem trocar mensagens com um aplicativo não JMS usando `JMSBytesMessage`. A troca de uma mensagem formatada com um aplicativo não JMS pode ocorrer com ou sem chamar uma saída de conversão de dados.

Antes de começar

Você pode ser capaz de projetar uma solução mais simples para a troca de mensagens com um aplicativo não JMS usando uma `JMSTextMessage`. Elimine que possibilidade antes de seguir as etapas nesta tarefa.

Sobre esta tarefa

Um cliente JMS será mais fácil de gravar se ele não estiver envolvido nos detalhes de formatação das mensagens do JMS trocadas com outros clientes do JMS. Desde que, o tipo de mensagem seja `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` ou `JMSObjectMessage`, IBM MQ trate os detalhes de formatação da mensagem. IBM MQ lida com as diferenças nas páginas de códigos e a codificação numérica em diferentes plataformas.

É possível usar esses tipos de mensagens para trocar mensagens com aplicativos não JMS. Para fazer isso, deve-se entender como essas mensagens são construídas pelo IBM MQ classes for JMS. Você pode ser capaz de modificar o aplicativo não JMS para interpretar as mensagens; consulte "Mapeando mensagens do JMS para mensagens do IBM MQ" na página 150.

Uma vantagem de usar um destes tipos de mensagens é que a programação do cliente JMS não depende do tipo de aplicativo que está sendo trocado com as mensagens. Uma desvantagem é que ela pode requerer uma modificação para outro programa e você pode não ser capaz de mudar o outro programa.

Uma abordagem alternativa é gravar um aplicativo cliente JMS que pode lidar com formatos de mensagens existentes. Geralmente as mensagens existentes são formatos fixos e contêm uma mistura de dados não formatados, textos e números. Use as etapas nesta tarefa e o exemplo do cliente JMS no "Gravando classes para encapsular um layout de registro em uma JMSBytesMessage" na página 194, como um ponto de início para construir um cliente JMS que pode trocar registros formatados com aplicativos não JMS.

Procedimento

1. Defina o layout de registro ou use uma das classes do cabeçalho do IBM MQ predefinido.

   Para manipular cabeçalhos predefinidos do IBM MQ, consulte Manipulando cabeçalhos da mensagem do IBM MQ.

Figura 37 na página 192 é um exemplo de um usuário definido, layout de registro de comprimento fixo, que pode ser processado pelo utilitário de conversão de dados.

2. Crie a saída de conversão de dados.

   Siga as instruções em Gravando um programa de saída de conversão de dados para gravar uma saída de conversão de dados.

   Para tentar o exemplo em "Gravando classes para encapsular um layout de registro em uma JMSBytesMessage" na página 194, nomeie a saída de conversão de dados MYRECORD.
3. Grave as classes do Java para conter o layout de registro e o envio e o recebimento de registro. Duas abordagens que você pode executar são:

   - Grave uma classe para que leia e grave o JMSBytesMessage que contém o registro; consulte "Gravando classes para encapsular um layout de registro em uma JMSBytesMessage" na página 194.
   - Grave uma classe estendendo com.ibm.mq.header.Header para definir a estrutura de dados do registro; consulte Criando classes para novos tipos de cabeçalho.
4. Decida qual conjunto de caracteres codificados ao qual trocar mensagens.

   Consulte Escolhendo uma abordagem para conversão de mensagem: receiver makes good.
5. Configure o destino para trocar mensagens do MQ-type, sem um cabeçalho JMS MQRFH2.

   O destino de envio e de recebimento devem estar configurados para trocar mensagens do MQ-type. É possível usar o mesmo destino para o envio e o recebimento.

   O aplicativo pode substituir a propriedade do corpo da mensagem de destino:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   O exemplo na "Gravando classes para encapsular um layout de registro em uma JMSBytesMessage" na página 194 substitui a propriedade do corpo da mensagem de destino, assegurando que uma mensagem estilo MQ seja enviada.
6. Teste a solução com aplicativos JMS e não JMS

   As ferramentas úteis para testar uma saída de conversão de dados são:

   - O programa de amostra amqsgetc0.c é útil para testar o recebimento de uma mensagem enviada por um cliente JMS. Consulte as modificações sugeridas para usar o cabeçalho de exemplo, RECORD.h, em Figura 38 na página 193. Com as modificações, o amqsgetc0.c recebe uma mensagem enviada pelo cliente JMS de exemplo, TryMyRecord.java ; consulte "Gravando classes para encapsular um layout de registro em uma JMSBytesMessage" na página 194.
   - O programa de procura do IBM MQ de amostra, amqsbcg0.c, é útil para inspecionar o conteúdo do cabeçalho da mensagem, o cabeçalho do JMS, MQRFH2 e o conteúdo da mensagem.
   - O programa **rfhutil** , anteriormente disponível em SupportPac IH03, permite que as mensagens de teste sejam capturadas e armazenadas em arquivos e, em seguida, usadas para conduzir Fluxos de mensagens As mensagens de saída também podem ser lidas e exibidas em uma variedade de formatos. Os formatos incluem dois tipos de XML, bem como correspondências com relação a um copybook COBOL. Os dados podem estar em EBCDIC ou ASCII. Um cabeçalho RFH2 pode ser incluído na mensagem antes de a mensagem ser enviada.

   Se você tentar receber mensagens usando o programa de amostra amqsgetc0.c modificado e obter um erro com o código de razão 2080, verifique se a mensagem possui um MQRFH2. As modificações presumem que a mensagem foi enviada para um destino que não especifica nenhum MQRFH2.

Exemplos

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                               MQLONG StructLength;
                               MQLONG Encoding;
                               MQLONG CodeCharSetId;
                               MQCHAR Format[8];
                               MQLONG Flags;
                               MQCHAR RecordData[32];
        };
```

Figura 37. RECORD.h

- Declare a estrutura de dados RECORD.h

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- Modifique a chamada MQGET para usar o RECORD,

  1. Antes da modificação:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. Após a modificação:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- Mude a instrução de impressão,

  1. De:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. Para:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

Figura 38. Modifique amqsget0.c

Conceitos relacionados

Abordagens de conversão de mensagem doJMS

Um número de abordagens de conversão de dados é aberto para designers de aplicativos JMS. Essas abordagens não são exclusivas; alguns aplicativos devem usar uma combinação dessas abordagens. Se seu aplicativo estiver trocando somente texto ou mensagens com outros aplicativos JMS, não considere a conversão de dados normalmente. A conversão de dados é executada automaticamente para você, por IBM MQ.

Conversão e codificação de mensagem do clienteJMS
Os métodos usados para conversão e codificação de mensagem do cliente JMS são listados com exemplos de códigos de cada tipo de conversão.

Conversão de dados do gerenciador de filas
A conversão de dados do gerenciador de filas sempre esteve disponível para aplicativos não JMS recebendo mensagens de clientes do JMS. Os clientes JMS que recebem mensagens também usam a conversão de dados do gerenciador de fila, que é opcional.

Utilitário para criação de código de saída de conversão

Referências relacionadas

Conversão e tipos de mensagem doJMS
A opção de tipo de mensagem afeta sua abordagem de conversão de mensagens. A interação de conversão de mensagem e o tipo de mensagem são descritos para os tipos de mensagens do JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` e `JMSBytesMessage`.

Gravando classes para encapsular um layout de registro em uma JMSBytesMessage
A finalidade desta tarefa é explorar, por exemplo, como combinar a conversão de dados e um layout de registro fixo em um `JMSBytesMessage`. Na tarefa, crie algumas classes do Java para trocar uma estrutura de registro de exemplo em um `JMSBytesMessage`. É possível modificar o exemplo para gravar classes para trocar outras estruturas de registro.

Um `JMSBytesMessage` é a melhor opção do tipo de mensagem JMS para trocar registros de tipo de dados mistos com programas não JMS. Ele não tem dados adicionais inseridos no corpo da mensagem pelo provedor JMS. Portanto, será a melhor opção do tipo de mensagem a ser usado, se um programa cliente JMS interoperar com um programa IBM MQ existente. O principal desafio em usar um `JMSBytesMessage` está em corresponder à codificação e ao conjunto de caracteres esperados pelo outro programa. Uma solução é criar uma classe que contenha o registro. Uma classe que contenha a leitura e gravação de um `JMSBytesMessage`, para um tipo de registro específico, torna mais fácil enviar e receber os registros de formato fixo em um programa JMS. Ao capturar os aspectos genéricos da interface em uma classe abstrata, grande parte da solução poderá ser reutilizada para formatos de registros diferentes. Os formatos de registros diferentes podem ser implementados em classes que estendem a classe genérica abstrata.

Uma abordagem alternativa é estender a classe `com.ibm.mq.headers.Header`. A classe `Header` tem métodos, como `addMQLONG`, para construir um formato de registro de uma maneira mais declarativa. A desvantagem de usar a classe `Header` é obter e configurar atributos usando uma interface interpretativa mais complicada. As duas abordagens resultam na mesma quantidade do código do aplicativo.

Um `JMSBytesMessage` pode conter apenas um único formato, além de um `MQRFH2`, em uma mensagem, a menos que cada registro use o mesmo formato, conjunto de caracteres codificado e codificação. O formato, a codificação e o conjunto de caracteres de um `JMSBytesMessage` são propriedades de todas as mensagens a seguir no `MQRFH2`. O exemplo é escrito na suposição de que um `JMSBytesMessage` contém apenas um registro do usuário.

Antes de começar

1. O nível de habilidade: deve-se estar familiarizado com programação do Java e JMS. Não são fornecidas instruções sobre como configurar o ambiente de desenvolvimento do Java. É vantajoso ter gravado um programa para trocar uma `JMSTextMessage`, `JMSStreamMessage` ou `JMSMapMessage`. É possível ver as diferenças em troca de uma mensagem usando um `JMSBytesMessage`.
2. O exemplo requer o IBM WebSphere MQ 7.0.
3. O exemplo foi criado usando a perspectiva do Java do ambiente de trabalho Eclipse. Ele requer o JRE 6.0 ou superior. É possível usar a perspectiva do Java no IBM MQ Explorer para desenvolver e executar as classes do Java. Como alternativa, usar seu próprio ambiente de desenvolvimento do Java.
4. Usando o IBM MQ Explorer torna a configuração do ambiente de teste e depuração, mais simples usando os utilitários de linha de comandos.

Sobre esta tarefa

Você é guiado através da criação de duas classes: RECORD e MyRecord. Juntos, essas duas classes contêm um registro de formato fixo. Eles têm métodos para obter e configurar os atributos. O método get lê o registro a partir de um JMSBytesMessage e o método put grava um registro em um JMSBytesMessage.

O propósito da tarefa é não criar uma classe de qualidade de produção que possa ser reutilizada. Você pode escolher usar os exemplos na tarefa para iniciar em suas próprias classes. A propósito da tarefa é fornecer notas de orientação, principalmente sobre o uso de conjuntos de caracteres, formatos e codificação, ao usar um JMSBytesMessage. Cada etapa da criação das classes é explicada e os aspectos de uso do JMSBytesMessage, que às vezes são omitidos, são descritos.

A classe RECORD é abstrata e define alguns campos comuns para um registro do usuário. Os campos comuns são modelados no layout de cabeçalho padrão do IBM MQ que tem um destaque, uma versão e um campo de comprimento. A codificação, o conjunto de caracteres e os campos de formato, localizados em muitos cabeçalhos do IBM MQ, são omitidos. Outro cabeçalho não pode seguir um formato definido pelo usuário. A classe MyRecord, que estende a classe RECORD, o faz literalmente, estendendo o registro com campos de usuário adicionais. Uma JMSBytesMessage, criada pelas classes, pode ser processada pela saída de conversão de dados do gerenciador de filas.

"Classes usadas para executar o exemplo" na página 201 inclui uma listagem completa de RECORD e MyRecord. Ele também inclui listagens de classes "andaime" extras para testar o RECORD e MyRecord. As classes extras são:

TryMyRecord
O programa principal para testar RECORD e MyRecord.

EndPoint
Uma classe abstrata que contém a conexão JMS, um destino e a sessão em uma única classe. Sua interface simplesmente atende às necessidades de teste das classes RECORD e MyRecord. Não é um padrão de design estabelecido para gravar aplicativos JMS.

Nota: A classe Endpoint incluirá esta linha de código após criar um destino:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Em V7.0, a partir de V7.0.1.5, é necessário para ativar a conversão do gerenciador de filas. Ele está desativado por padrão. Em V7.0, até V7.0.1.4, a conversão do gerenciador de filas é ativada por padrão e essa linha de códigos causa um erro.

MyProducer e MyConsumer
As classes que estendem EndPoint e criam um MessageConsumer e MessageProducer, conectadas e prontas para aceitar as solicitações.

Juntas todas as classes formam um aplicativo completo o qual é possível construir e testar, para entender como usar a conversão de dados em um JMSBytesMessage.

Procedimento

1. Crie uma classe abstrata para conter os campos padrão em um cabeçalho IBM MQ, com um construtor padrão. Posteriormente, estenda a classe para customizar o cabeçalho para seus requisitos.

```
public abstract class RECORD implements Serializable {
   private static final long serialVersionUID = -1616617232750561712L;
   protected final static int UTF8 = 1208;
   protected final static int MQLONG_LENGTH = 4;
   protected final static int RECORD_STRUCT_ID_LENGTH = 4;
   protected final static int RECORD_VERSION_1 = 1;
   protected final String RECORD_STRUCT_ID = "BLNK";
   protected final String RECORD_TYPE = "BLANK   ";
   private String structID = RECORD_STRUCT_ID;
```

```
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
```

Nota:

a. Os atributos, `structID` para `nextFormat`, são listados na ordem em que são definidos em um cabeçalho da mensagem padrão do IBM MQ.

b. Os atributos, `format`, `messageEncoding` e `messageCharset`, descrevem o próprio cabeçalho e não fazem parte do cabeçalho.

c. Deve-se decidir se armazenar o identificador do conjunto de caracteres codificados ou o conjunto de caracteres do registro. Java usa conjuntos de caracteres e as mensagens do IBM MQ usam identificadores de conjuntos de caracteres codificados. O código de exemplo usa conjuntos de caracteres.

d. `int` é serializado para `MQLONG` pelo IBM MQ. `MQLONG` é de 4 bytes.

2. Crie os getters e setters para os atributos privados.

a) Crie ou gere os getters:

```
     public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }
```

b) Crie ou gere os setters:

```
     public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

3. Crie um construtor para criar uma instância `RECORD` a partir de um `JMSBytesMessage`.

```
  public RECORD(BytesMessage message) throws JMSException, IOException,
         MQDataException {
     super();
     setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
     setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
     byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
     message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
     setStructID(new String(structID, getMessageCharset()));
     setVersion(message.readInt());
     setStructLength(message.readInt());
  }
```

Nota:

a. O `messageCharset` e `messageEncoding` são capturados a partir das propriedades da mensagem, pois eles substituem o conjunto de valores para o destino. `format` não é atualizado. O exemplo não faz verificação de erros. Se o construtor `Record(BytesMessage)` for chamado, será presumido que o `JMSBytesMessage` é um tipo de mensagem RECORD. A linha "`setStructID(new String(structID, getMessageCharset()))`" configura o destaque.

b. As linhas de códigos que concluem os campos do método deserialize na mensagem, em ordem, atualizando os valores padrão configurados na instância RECORD.

4. Crie um método put para gravar os campos de cabeçalho em um `JMSBytesMessage`.

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
       JMSException, UnsupportedEncodingException {
   setHeaderEncoding(myProducer.getEncoding());
   setHeaderCharset(myProducer.getCharset());
   myProducer.setMQClient(true);
   BytesMessage bytes = myProducer.session.createBytesMessage();
   bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
   bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
   bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                        myProducer.getCCSID());
   bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                  + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                  .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
   bytes.writeInt(getVersion());
   bytes.writeInt(getStructLength());
   return bytes;
}
```

Nota:

a. `MyProducer` contém o JMS `Connection`, `Destination`, `Session` e `MessageProducer` em uma única classe. MyConsumer, usado posteriormente, contém o JMS `Connection`, `Destination`, `Session` e `MessageConsumer` em uma única classe.

b. Para um `JMSBytesMessage`, se a codificação for diferente de `Native`, a codificação deverá ser configurada na mensagem. A codificação de destino é copiada para o atributo de codificação de mensagem, `JMS_IBM_CHARACTER_SET` e salvo como um atributo da classe RECORD.

   i) "`setMessageEncoding(myProducer.getEncoding());`" chama "`(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));`" para obter a codificação de destino.

   ii) "`Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());`" configura a codificação de mensagem.

c. O conjunto de caracteres usado para transformar o texto em bytes é obtido a partir do destino e salvo como um atributo da classe RECORD. Ele não é configurado na mensagem, pois não será usado pelo IBM MQ classes for JMS ao gravar um `JMSBytesMessage`.

   "`messageCharset = myProducer.getCharset();`" chama

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

   Ele obtém o conjunto de caracteres do Java a partir de um identificador do conjunto de caracteres codificados.

   "`CCSID.getCodepage(ccsid)`" está no pacote `com.ibm.mq.headers`. O `ccsid` é obtido a partir de um outro método no `MyProducer`, que consulta o destino:

```
public int getCCSID() throws JMSException {
    return (((MQDestination) destination)
            .getIntProperty(WMQConstants.WMQ_CCSID));
}
```

d. "myProducer.setMQClient(true);" substitui a configuração de destino para o tipo de cliente, forçando-a para um IBM MQ MQI client. Você pode preferir omitir esta linha de código, conforme ela oculta um erro de configuração administrativa.

"myProducer.setMQClient(true);" chama:

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

O código possui o efeito colateral de configuração do estilo de corpo do IBM MQ para não especificado, se ele deve substituir uma configuração de JMS.

Nota:

O IBM MQ classes for JMS grava o formato, a codificação e o identificador do conjunto de caracteres da mensagem no descritor de mensagens, MQMD ou no cabeçalho do JMS, MQRFH2. Depende se a mensagem tem um corpo de estilo do IBM MQ. Não configure os campos MQMD manualmente.

Um método existe para configurar as propriedades do descritor de mensagens manualmente. Ele usa as propriedades JMS_IBM_MQMD_*. Deve-se configurar a propriedade de destino, WMQ_MQMD_WRITE_ENABLED para configurar as propriedades JMS_IBM_MQMD_*:

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

Deve-se configurar a propriedade de destino WMQ_MQMD_READ_ENABLED para ler as propriedades.

Use o JMS_IBM_MQMD_* somente se você tomar o controle total da carga útil da mensagem. Diferente das propriedades JMS_IBM_*, as propriedades JMS_IBM_MQMD_* não controlam como o IBM MQ classes for JMS constrói uma mensagem do JMS. É possível criar propriedades do descritor de mensagens que entre em conflito com as propriedades da mensagem do JMS.

e. As linhas de códigos que concluem o método serializam os atributos na classe como campos na mensagem.

Os atributos de sequência são preenchidos com espaços em branco. As sequências são convertidas em bytes usando o conjunto de caracteres definido para o registro e truncado para o comprimento dos campos de mensagem.

5. Conclua a classe incluindo as importações.

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. Crie uma classe para estender a classe RECORD para incluir campos adicionais. Incluir um construtor padrão.

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

   public MyRecord() {
       super();
```

```
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
```

Nota:

a. A subclasse RECORD, MyRecord, customiza o destaque, formato e comprimento do cabeçalho.

7. Crie ou gere os getters e setters.

   a) Crie os getters:

```
     public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .
```

   b) Crie os setters:

```
     public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

8. Crie um construtor para criar uma instância MyRecord a partir de um JMSBytesMessage.

```
  public MyRecord(BytesMessage message) throws JMSException, IOException,
         MQDataException {
     super(message);
     setFlags(message.readInt());
     byte[] recordData = new byte[DATA_LENGTH];
     message.readBytes(recordData, DATA_LENGTH);
     setRecordData(new String(recordData, super.getMessageCharset()));
  }
```

Nota:

a. Os campos que formam o modelo de mensagem padrão são lidos primeiro pela classe RECORD.

b. O texto recordData será convertido em String usando a propriedade do conjunto de caracteres da mensagem.

9. Crie um método estático para obter uma mensagem a partir de um consumidor e crie uma nova instância MyRecord.

```
  public static MyRecord get(MyConsumer myConsumer) throws JMSException,
         MQDataException, IOException {
     BytesMessage message = (BytesMessage) myConsumer.receive();
     return new MyRecord(message);
  }
```

Nota:

a. No exemplo, para abreviar, o construtor MyRecord(BytesMessage) é chamado a partir do método get estático. Geralmente, você pode separar o recebimento da mensagem a partir da criação de uma nova instância MyRecord.

10. Crie um método put para anexar os campos do cliente a uma JMSBytesMessage que contém um cabeçalho da mensagem.

```
  public BytesMessage put(MyProducer myProducer) throws JMSException,
         IOException {
     BytesMessage bytes = super.put(myProducer);
     bytes.writeInt(getFlags());
```

```
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
```

Nota:

a. As chamadas de método no código serializam os atributos na classe MyRecord como campos na mensagem.

- O atributo `recordData` `String` é preenchido com espaços em branco, convertidos em bytes usando o conjunto de caracteres definido para o registro e truncado para o comprimento dos campos `RecordData`.

11. Conclua a classe, incluindo as instruções de inclusão.

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Resultados

- Os resultados da execução da classe `TryMyRecord`:
  - Enviando mensagem no conjunto de caracteres codificados 37 e usando uma saída de conversão do gerenciador de filas:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - Enviando mensagem no conjunto de caracteres codificados 37 e não usando uma saída de conversão do gerenciador de filas:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- Os resultados da modificação da classe `TryMyRecord` para não receber a mensagem, e em vez do recebimento, usando a amostra `amqsget0.c` modificada. A amostra modificada aceita um registro formatado; consulte Figura 38 na página 193 no “Trocando um registro formatado com um aplicativo não JMS” na página 190.
  - Enviando mensagem no conjunto de caracteres codificados 37 e usando uma saída de conversão do gerenciador de filas:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - Enviando mensagem no conjunto de caracteres codificados 37 e não usando uma saída de conversão do gerenciador de filas:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

Para tentar o exemplo e experimento com diferentes páginas de códigos e uma saída de conversão de dados. Crie as classes do Java, configure IBM MQ e execute o programa principal, `TryMyRecord`; consulte “#unique_196/unique_196_Connect_42_Try” na página 201.

1. Configure o IBM MQ e JMS para executar o exemplo. As instruções são para executar o exemplo no Windows.
   a. Crie um gerenciador de filas

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. Crie uma fila

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. Crie um diretório JNDI

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. Alterne para o diretório bin do JMS

   O programa JMS Administration deve ser executado a partir daqui. O caminho é *MQ_INSTALLATION_PATH*`\java\bin`.

   e. Crie as seguintes definições JMS em um arquivo chamado `JMSQM1Q1.txt`

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. Execute o programa JMSAdmin para criar os recursos do JMS

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. É possível criar, alterar e procurar as definições criadas usando o IBM MQ Explorer.
3. Execute o `TryMyRecord`.

Classes usadas para executar o exemplo

As classes listada nos blocos de códigos a seguir estão também disponíveis em um arquivo compactados Faça o download de jm25529_.zip ou jm25529_.tar.gz

`TryMyRecord`

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
```

```
    }
}
```

RECORD

JM 3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

JM 3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
```

```
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

JM 3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
```

```
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
```

```
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

JM 3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

JM 3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

Criando e configurando connection factories e destinos

Um aplicativo IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging pode criar connection factories e destinos recuperando-os como objetos administrados de um namespace Java Naming and Directory Interface (JNDI), usando as extensões do IBM JMS ou usando as extensões do IBM MQ JMS . Um aplicativo também pode usar as extensões do IBM JMS ou do IBM MQ JMS para configurar as propriedades de ações de connection factories.

Connection factories e destinos são pontos de início no fluxo de lógica de um aplicativo JMS ou Jakarta Messaging Um aplicativo usa um objeto ConnectionFactory para criar uma conexão com um servidor de

sistema de mensagens, e usa um objeto Fila ou Tópico como um destino para enviar mensagens para ou uma origem da qual receber mensagens. Um aplicativo, portanto, precisa criar pelo menos um connection factory e um ou mais destinos. Ter criado uma connection factory ou destino, o aplicativo poderá então precisar configurar o objeto, configurando uma ou mais de suas propriedades.

Em resumo, um aplicativo pode criar e configurar connection factories e destinos das seguintes maneiras:

Usando JNDI para recuperar objetos administrados
Um administrador pode usar a ferramenta de administração do IBM MQ JMS conforme descrito em Configurando objetos JMS e Jakarta Messaging usando as ferramentas de administraçãoou IBM MQ Explorer conforme descrito em Configurando objetos JMS 2.0 usando o IBM MQ Explorer, para criar e configurar connection factories e destinos como objetos administrados em um namespace JNDI. Um aplicativo pode então recuperar os objetos administrados do espaço de nomes JNDI. Tendo recuperado um objeto administrado, o aplicativo pode, se necessário, configurar ou mudar uma ou mais de suas propriedades usando as extensões do IBM JMS ou as extensões do IBM MQ JMS.

Nota: JM 3.0 Para Jakarta Messaging 3.0, não é possível administrar o JNDI usando IBM MQ Explorer A administração de JNDI é suportada pela variante Jakarta Messaging 3.0 de **JMSAdmin**, que é **JMS30Admin**

usando as extensões IBM JMS
Um aplicativo pode usar as extensões do IBM JMS para criar connection factories e destinos dinamicamente no tempo de execução. O aplicativo cria primeiramente um objeto JmsFactoryFactory e, em seguida, usa métodos desse objeto para criar connection factories e destinos. Depois de criar um connection factory ou o destino, o aplicativo pode usar métodos herdado da interface JmsPropertyContext para configurar suas propriedades. Como alternativa, o aplicativo pode utilizar um URI (Uniform Resource Identifier) para especificar uma ou mais propriedades de um destino, quando ele cria o destino.

usando as extensões IBM MQ JMS
Um aplicativo também pode usar extensões do IBM MQ JMS para criar connection factories e destinos dinamicamente no tempo de execução. O aplicativo usa os construtores fornecidos para criar connection factories e destinos. Depois de criado um connection factory ou destino, o aplicativo pode usar os métodos do objeto para configurar suas propriedades. Como alternativa, o aplicativo pode utilizar um URI para especificar uma ou mais propriedades de um destino, quando ele cria o destino.

Tarefas relacionadas
Configurando recursos do JMS e do Jakarta Messaging

Usando JNDI para recuperar objetos administrados em um aplicativo JMS ou Jakarta Messaging
Para recuperar objetos administrados de um namespace Java Naming and Directory Interface (JNDI), um aplicativo JMS ou Jakarta Messaging deve criar um contexto inicial e, em seguida, usar o método lookup () para recuperar os objetos.

Antes de um aplicativo poder recuperar objetos administrados a partir de um namespace do JNDI, um administrador deverá primeiro criar os objetos administrados.

JMS 2.0 Para JMS 2.0, o administrador pode usar a ferramenta de administração IBM MQ JMS , **JMSAmin**ou IBM MQ Explorer para criar e manter objetos administrados em um namespace JNDI. Para obter mais informações, consulte Configurando connection factories e destinos em um namespace do JNDI.

JM 3.0 Para Jakarta Messaging 3.0, não é possível administrar o JNDI usando IBM MQ Explorer A administração de JNDI é suportada pela variante Jakarta Messaging 3.0 de **JMSAdmin**, que é **JMS30Admin**

Um servidor de aplicativos, geralmente fornece seu próprio repositório para objetos administrados e suas próprias ferramentas para criar e manter os objetos.

Para recuperar objetos administrados de um namespace do JNDI, um aplicativo deve criar primeiro um contexto inicial, conforme mostrado no exemplo a seguir:

JM 3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

Nesse código, as variáveis String `url` e `icf` possuem os seguintes significados:

url
O localizador uniforme de recursos (URL) do serviço de diretório. A URL pode ter um dos formatos a seguir:

- `ldap://`*`hostname`*`/`*`contextName`* , para um serviço de diretório com base em um servidor LDAP
- `file:/`*`directoryPath`* , para um serviço de diretório com base no sistema de arquivos local

icf
O nome da classe da factory de contexto inicial, que pode ser um dos seguintes valores:

- `com.sun.jndi.ldap.LdapCtxFactory`, para um serviço de diretório com base em um servidor LDAP
- `com.sun.jndi.fscontext.RefFSContextFactory`, para um serviço de diretório com base no sistema de arquivos local

Observe que algumas combinações de um pacote JNDI e um provedor de serviços Lightweight Directory Access Protocol (LDAP) podem causar o erro LDAP 84 para ocorrer. Para resolver esse problema, insira a seguinte linha de código antes da chamada para InitialDirContext():

```
environment.put(Context.REFERRAL, "throw");
```

Após um contexto inicial ser obtido, o aplicativo poderá recuperar objetos administrados a partir do namespace do JNDI, usando o método lookup(), conforme mostrado no exemplo a seguir:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
```

```
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

Esse código recupera os seguintes objetos a partir de um namespace base do LDAP:

- Um limite de objeto ConnectionFactory com o nome myCF
- Um limite de objeto Queue com o nome myQ
- Um limite de objeto Topic com o nome myT

Para obter mais informações sobre o uso do JNDI, consulte a documentação do JNDI fornecida pela Oracle Corporation.

Tarefas relacionadas

Configurando objetos JMS 2.0 usando o IBM MQ Explorer

Configurando objetos JMS e Jakarta Messaging usando as ferramentas de administração

Configurando recursos do JMS 2.0 no WebSphere Application Server

usando as extensões IBM JMS

Cada IBM MQ classes for JMS (JMS 2.0) e IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) contém um conjunto de extensões funcionalmente idênticas para a API JMS chamada de extensões IBM JMS . Um aplicativo pode utilizar essas extensões para criar connection factories e destinos dinamicamente no tempo de execução e configurar as propriedades de objetos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . As extensões podem ser usadas com qualquer provedor de sistema de mensagens.

As extensões do IBM JMS são um conjunto de interfaces e classes nos pacotes a seguir:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Para o Jakarta Messaging 3.0, esses pacotes estão no `com.ibm.jakarta.client.jar`

JMS 2.0 Para o JMS 2.0, esses pacotes estão em `com.ibm.mqjms.jar` ou `com.ibm.mq.allclient.jar`

Essas extensões fornecem a seguinte função:

- Um mecanismo baseado em fábrica para criar connection factories e destinos dinamicamente no tempo de execução, em vez de recuperá-los como objetos administrados a partir de um namespace Java Naming and Directory Interface (JNDI)
- Um conjunto de métodos para configurar as propriedades de objetos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging
- Um conjunto de classes de exceção com métodos para obter informações detalhadas sobre um problema
- Um conjunto de métodos para controlar o rastreio
- Um conjunto de métodos para obter informações de versão sobre IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

Para criar connection factories e destinos dinamicamente no tempo de execução e configurar e obter suas propriedades, as extensões IBM JMS fornecem um conjunto alternativo de interfaces para as extensões do IBM MQ JMS . No entanto, considerando que as extensões IBM MQ JMS são específicas para o provedor de mensagens IBM MQ, as extensões IBM JMS não são específicas para IBM MQ e podem ser usadas com qualquer provedor de mensagens dentro da arquitetura em camadas, descrita nas classes do IBM MQ para a arquitetura do JMS.

A interface com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) ou com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) contém as definições de constantes que um aplicativo pode usar ao configurar as propriedades de objetos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging usando as extensões do IBM JMS . A interface contém constantes do provedor do sistema de mensagens do IBM MQ e constantes do JMS que são independentes de qualquer provedor de sistema de mensagens.

Os exemplos de código a seguir assumem que as instruções de importação a seguir estão incluídas na classe Java :

JM 3.0
```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0
```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Criando connection factories e destinos

Antes que um aplicativo possa criar connection factories e destinos usando as extensões do IBM JMS, ele deve primeiro criar um objeto JmsFactoryFactory. Para criar um objeto JmsFactoryFactory, o aplicativo chama o método getInstance() da classe JmsFactoryFactory, conforme mostrado no exemplo a seguir:

JM 3.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

O parâmetro na chamada getInstance() é uma constante que identifica o provedor de mensagens do IBM MQ como o provedor de sistema de mensagens escolhido. O aplicativo pode então usar o objeto JmsFactoryFactory para criar connection factories e destinos.

Para criar um connection factory, o aplicativo chama o método createConnectionFactory() do objeto JmsFactoryFactory, conforme mostrado no exemplo a seguir:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

Esta instrução cria um objeto JmsConnectionFactory com os valores padrão para todas as suas propriedades, o que significa que o aplicativo se conecta ao gerenciador de filas padrão no modo de ligações. Se desejar que um aplicativo se conecte no modo cliente ou se conecte a um gerenciador de filas diferente do gerenciador de filas padrão, o aplicativo deve configurar as propriedades apropriadas do objeto JmsConnectionFactory antes de criar a conexão. Para obter informações sobre como fazer isso, consulte “Configurando as propriedades de objetos do IBM MQ classes for JMS” na página 213.

A classe JmsFactoryFactory também contém métodos para criar connection factories dos tipos a seguir:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

Para criar um objeto Queue, o aplicativo chama o método createQueue() do objeto JmsFactoryFactory, conforme mostrado no exemplo a seguir:

```
JmsQueue q1 = ff.createQueue("Q1");
```

Esta instrução cria um objeto JmsQueue com os valores padrão para todas as suas propriedades. O objeto representa uma fila do IBM MQ chamada Q1 que pertence ao gerenciador de filas local. Essa fila pode ser uma fila local, uma fila de alias ou uma definição de fila remota.

O método createQueue() também pode aceitar um identificador uniforme de recursos (URI) da fila como um parâmetro. Uma fila de URI é uma sequência que especifica o nome de uma fila do IBM MQ e, como opção, o nome do gerenciador de filas que possui a fila e uma ou mais propriedades do objeto JmsQueue. A instrução a seguir contém um exemplo de um URI de fila:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

O objeto JmsQueue criado por esta instrução representa uma fila IBM MQ chamada Q2 que é de propriedade do gerenciador de filas QM2, e todas as mensagens enviadas para este destino são persistentes e têm prioridade de 5. Para obter mais informações sobre URIs de filas, consulte “Identificadores uniformes de recursos (URIs)” na página 226. Para um modo alternativo para configurar propriedades de um objeto JmsQueue, consulte “Configurando as propriedades de objetos do IBM MQ classes for JMS” na página 213.

Para criar um objeto Topic, um aplicativo pode usar o método createTopic() do objeto JmsFactoryFactory, conforme mostrado no exemplo a seguir:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

Esta instrução cria um objeto JmsTopic com os valores padrão para todas as suas propriedades. O objeto representa um tópico chamado Sport/Football/Results.

O método createTopic() também pode aceitar um URI de tópico como um parâmetro. Um URI de tópico é uma sequência que especifica o nome de um tópico e, como opção, uma ou mais propriedades do objeto JmsTopic. As instruções a seguir contêm um exemplo de um URI de tópico:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

O objeto JmsTopic criado por essas instruções representa um tópico chamado Esporte/Tênis/Resultados, e todas as mensagens enviadas para este destino são não persistentes e têm prioridade de 0. Para obter mais informações sobre URIs de tópicos, consulte “Identificadores uniformes de recursos (URIs)” na página 226. Para um modo alternativo para configurar propriedades de um objeto JmsTopic, consulte “Configurando as propriedades de objetos do IBM MQ classes for JMS” na página 213.

Após um aplicativo ter criado um connection factory ou destino, esse objeto pode ser usado somente com o provedor de sistema de mensagens selecionado.

Configurando as propriedades de objetos do IBM MQ classes for JMS

Para configurar as propriedades de objetos do IBM MQ classes for JMS usando as extensões do IBM JMS, um aplicativo usa os métodos da interface com.ibm.msg.client.JmsPropertyContext. Da mesma forma, para configurar as propriedades dos objetos IBM MQ classes for Jakarta Messaging usando as extensões IBM JMS , um aplicativo usa os métodos da interface com.ibm.msg.jakarta.client.JmsPropertyContext .

Para cada tipo de dados Java, a interface JmsPropertyContext contém um método para configurar o valor de uma propriedade com esse tipo de dados e um método para obter o valor de uma propriedade com esse tipo de dados. Por exemplo, um aplicativo chama o método setIntProperty() para configurar uma propriedade com um valor de número inteiro e chama o método getIntProperty() para obter uma propriedade com um valor de número inteiro.

As instâncias de classes nos pacotes com.ibm.mq.jms e com.ibm.mq.jakarta.jms herdam os métodos das interfaces de Contexto JmsPropertycorrespondentes Um aplicativo pode, portanto, usar esses métodos para configurar as propriedades dos objetos MQConnectionFactory, MQQueue e MQTopic.

Quando um aplicativo cria um objeto IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , todas as propriedades com valores padrão são configuradas automaticamente. Quando um aplicativo configura uma propriedade, o novo valor substitui qualquer valor anterior da propriedade. Após uma propriedade ser configurada, ela não pode ser excluída, mas seu valor poderá ser mudado.

Se um aplicativo tentar configurar uma propriedade para um valor que não seja um valor válido para a propriedade, IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging emitirá uma exceção JMSException. Se um aplicativo tentar obter uma propriedade que não foi configurada, o comportamento será conforme descrito na especificação JMS. IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging lançam uma exceção de exceção NumberFormatpara tipos de dados primitivos e retornam nulo para tipos de dados referenciados.

Além das propriedades predefinidas de um objeto IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , um aplicativo pode definir suas propriedades. Essas propriedades definidas pelo aplicativo são ignorados por IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Para obter mais informações sobre as propriedades de objetos IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging , consulte Propriedades de objetos IBM MQ classes for JMS

O código a seguir é um exemplo de como configurar propriedades usando as extensões do IBM JMS. O código configura cinco propriedades de um connection factory.

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

O efeito de configurar estas propriedades é que o aplicativo se conecta ao gerenciador de filas QM1 no modo cliente, usando um canal MQI chamado QM1.SVR. O gerenciador de filas está em execução em um sistema com nome do host HOST1 e o listener para o gerenciador de filas está atendendo no número da porta 1415. Esta conexão e outras conexões do gerenciador de filas associadas a outras sessões sob ela têm o nome do aplicativo “Meu aplicativo" associado a elas.

Nota: Os gerenciadores de filas em execução em plataformas z/OS não suportam a configuração de nomes de aplicativos, portanto, essa configuração é ignorada.

A interface JmsPropertyContext também contém o método setObjectProperty(), que um aplicativo pode usar para configurar as propriedades. O segundo parâmetro do método é um objeto que contém o valor da propriedade. Por exemplo, o código a seguir cria um objeto de Número inteiro que contém o número inteiro 1415 e, em seguida, chama setObjectProperty() para configurar a propriedade PORT de um connection factory para o valor 1415:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

Esse código é, portanto, equivalente à instrução a seguir:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

Por outro lado, o método getObjectProperty() retorna um objeto que contém o valor de uma propriedade.

Conversão implícita de um valor de propriedade de um tipo de dados para outro

Quando um aplicativo usa um método da interface de contexto JmsPropertypara configurar ou obter a propriedade de um objeto IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , o valor da propriedade pode ser convertido implicitamente de um tipo de dados para outro...

Por exemplo, a instrução a seguir configura a propriedade PRIORITY do objeto JmsQueue q1:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

A propriedade PRIORITY tem um valor de número inteiro e, portanto, a chamada setStringProperty() converte implicitamente a sequência "5" (o valor de origem) para o número inteiro 5 (o valor de destino), que, então, se torna o valor da propriedade PRIORITY.

Por outro lado, a instrução a seguir obtém a propriedade PRIORITY do objeto JmsQueue q1:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

O número inteiro 5 (o valor de origem), que é o valor da propriedade PRIORITY, é convertido implicitamente para a sequência "5" (o valor de destino) pela chamada getStringProperty().

As conversões suportadas pelo IBM MQ classes for JMS e pelo IBM MQ classes for Jakarta Messaging são mostradas em Tabela 34 na página 215

Tabela 34. Conversões suportadas de um tipo de dados para outro

Tipo de dados de origem	**Tipos de dados de destino suportados**
booleano	Sequência
byte	int, long, short, String
char	Sequência
double	Sequência
float	double, String
int	long, String
long	Sequência
short	int, long, String
Sequência	boolean, byte, double, float, int, long, short

As regras gerais que regem as conversões suportadas são as seguintes:

- Valores numéricos podem ser convertidos de um tipo de dados para outro desde que nenhum dado seja perdido durante a conversão. Por exemplo, um valor com tipo de dados `int` pode ser convertido para um valor com tipo de dados `long`, mas não pode ser convertido para um valor com tipo de dados `short`.
- Um valor de qualquer tipo de dados pode ser convertido para uma sequência.
- Uma sequência pode ser convertida para um valor de qualquer outro tipo de dados (exceto `char`) desde que a sequência esteja no formato correto para a conversão. Se um aplicativo tentar converter uma sequência que não esteja no formato correto, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging lançará uma exceção de exceção NumberFormat.
- Se um aplicativo tentar uma conversão que não seja suportada, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging emitirá uma exceção de Exceção MessageFormat

As regras específicas para converter um valor de um tipo de dados para outro são as seguintes:

- Ao converter um valor booleano para uma sequência, o valor `true` é convertido para a sequência "true" e o valor `false` para a sequência "false".
- Ao converter uma sequência para um valor booleano, a sequência "true" (sem distinção entre maiúsculas e minúsculas) é convertida para `true` e a sequência "false" (sem distinção entre maiúsculas e minúsculas) é convertido para `false`. Qualquer outra sequência é convertida para `false`.
- Ao converter uma sequência para um valor com tipo de dados `byte`, `int`, `long` ou `short`, a sequência deve ter o seguinte formato:

  [*espaços em branco*][*sinal*] *dígitos*

  Os significados dos componentes da sequência são os seguintes:

  espaços em branco
  Caracteres em branco à esquerda opcionais.

sinal
Um sinal de mais (+) ou de menos (-) opcional.

dígitos
Uma sequência contínua de dígitos (0 a 9). Pelo menos um dígito deve estar presente.

Após a sequência de dígitos, a sequência pode conter outros caracteres que não são dígitos, mas a conversão para assim que o primeiro desses caracteres for atingido. A sequência é assumida para representar um número inteiro decimal.

Se a sequência não estiver no formato correto, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging lançam uma exceção de exceção NumberFormat.

- Ao converter uma sequência para um valor com tipo de dados `double` ou `float`, a sequência deve ter o formato a seguir:

  [*espaços em branco*][*sinal*] *dígitos* [*e_char* [*e_sign*] *e_digits*]

  Os significados dos componentes da sequência são os seguintes:

  espaços em branco
  Caracteres em branco à esquerda opcionais.

  sinal
  Um sinal de mais (+) ou de menos (-) opcional.

  dígitos
  Uma sequência contínua de dígitos (0 a 9). Pelo menos um dígito deve estar presente.

  e_char
  Um caractere expoente, que é um *E* ou *e*.

  e_sign
  Um sinal de mais (+) ou de menos (-) opcional para o expoente.

  e_digits
  Uma sequência contínua de dígitos (0 a 9) para o expoente. Pelo menos um dígito deve estar presente se a sequência contiver um caractere expoente.

  Após a sequência de dígitos ou dos caracteres opcionais que representam um expoente, a sequência pode conter outros caracteres que não são dígitos, mas a conversão para assim que o primeiro desses caracteres for atingido. Supõe-se que a sequência represente um número de vírgula flutuante decimal com um expoente que é uma potência de 10.

  Se a sequência não estiver no formato correto, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging lançam uma exceção de exceção NumberFormat.

- Ao converter um valor numérico (incluindo um valor com tipo de dados `byte`) para uma sequência, o valor será convertido para a representação em sequência do valor como um número decimal, não a sequência que contém o caractere ASCII para esse valor. Por exemplo, o número inteiro 65 será convertido para a sequência "65", não para a sequência "A".

Configurando mais de uma propriedade em uma única chamada

A interface JmsPropertyContext também contém o método setBatchProperties(), que um aplicativo pode usar para configurar mais de uma propriedade em uma única chamada. O parâmetro do método é um objeto de Mapa que contém um conjunto de pares nome-valor de propriedades.

Por exemplo, o código a seguir usa o método setBatchProperties() para configurar as mesmas cinco propriedades de um connection factory conforme mostrado em “Configurando as propriedades de objetos do IBM MQ classes for JMS” na página 213. O código cria uma instância da classe HashMap, que implementa a interface de Map.

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
```

```
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

Observe que o segundo parâmetro do método Map.put() deve ser um objeto. Portanto, um valor de propriedade com um tipo de dados primitivo deve estar contido em um objeto ou ser representado por uma sequência conforme mostrado no exemplo.

O método setBatchProperties() valida cada propriedade. Se o método setBatchProperties() não puder configurar uma propriedade porque, por exemplo, seu valor não é válido, nenhuma das propriedades especificadas serão configuradas.

Nomes e valores de propriedade

Se um aplicativo usar os métodos da interface de contexto JmsPropertyapropriada para configurar e obter as propriedades de objetos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , o aplicativo poderá especificar os nomes e valores de propriedades de qualquer uma das seguintes maneiras. Cada um dos exemplos fornecidos mostra como configurar a propriedade PRIORITY do objeto JmsQueue q1 de forma que uma mensagem enviada para a fila tenha a prioridade especificada na chamada send().

Usando os nomes e valores de propriedades definidos como constantes na interface com.ibm.msg.client.wmq.WMQConstants

A instrução a seguir é um exemplo de como especificar os nomes e valores de propriedades dessa maneira:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

Usando os nomes e valores de propriedades que podem ser usados nos identificadores uniformes de recursos (URIs) da fila e do tópico

A instrução a seguir é um exemplo de como especificar os nomes e valores de propriedades dessa maneira:

```
q1.setIntProperty("priority", -2);
```

Somente os nomes e valores de propriedades de destinos podem ser especificados dessa maneira.

Usando os nomes e os valores de propriedades reconhecidos pela ferramenta de administração do IBM MQ JMS

A instrução a seguir é um exemplo de como especificar os nomes e valores de propriedades dessa maneira:

```
q1.setStringProperty("PRIORITY", "APP");
```

A forma abreviada do nome da propriedade também é aceitável, conforme mostrado na instrução a seguir:

```
q1.setStringProperty("PRI", "APP");
```

Quando um aplicativo obtém uma propriedade, o valor retornado depende da maneira que o aplicativo especifica o nome da propriedade. Por exemplo, se um aplicativo especifica a constante WMQConstants.WMQ_PRIORITY como o nome da propriedade, o valor retornado é o número inteiro -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

O mesmo valor será retornado se o aplicativo especificar a sequência “priority" como o nome da propriedade:

```
int n2 = getIntProperty("priority");
```

No entanto, se o aplicativo especificar a sequência "PRIORITY" ou "PRI" como o nome da propriedade, o valor retornado será a sequência "APP":

```
String s1 = getStringProperty("PRI");
```

Internamente, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging armazenam nomes e valores de propriedades como os valores literais definidos na interface WMQConstants correspondente Esse é o formato canônico definido para os nomes e valores de propriedades. Como regra geral, se um aplicativo configurar propriedades usando uma das outras duas maneiras de especificar nomes e valores de propriedades, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging terão que converter os nomes e valores do formato de entrada especificado no formato canônico. Da mesma forma, se um aplicativo obtiver propriedades usando uma das outras duas maneiras de especificar nomes e valores de propriedades, o IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging deverão converter os nomes do formato de entrada especificado no formato canônico e converter os valores do formato canônico no formato de saída necessário para. Precisar executar essas conversões pode ter implicações no desempenho.

Os nomes de propriedades e valores retornados por exceções, em arquivos de rastreio ou no log IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging estão sempre no formato canônico.

Usando a interface Map

A interface JmsPropertyContext estende a interface java.util.Map. Um aplicativo pode, portanto, usar os métodos da interface de Mapa para acessar as propriedades de um objeto IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

Por exemplo, o código a seguir imprime os nomes e os valores de todas as propriedades de um connection factory. O código usa somente os métodos da interface Map para obter os nomes e os valores das propriedades.

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

Usar os métodos da interface Map não ignora quaisquer validações ou conversões de propriedades.

Usando as extensões IBM MQ JMS
O IBM MQ classes for JMS contém um conjunto de extensões para a API do JMS chamado extensões do IBM MQ JMS. Um aplicativo pode usar estas extensões para criar connection factories e destinos dinamicamente no tempo de execução, e para configurar suas propriedades.

O IBM MQ classes for JMS contém um conjunto de classes nos pacotes com.ibm.jms e com.ibm.mq.jms. Essas classes implementam as interfaces do JMS e contêm as extensões do IBM MQ JMS. Os exemplos de código a seguir supõem que estes pacotes tenham sido importados pelas seguintes instruções:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Um aplicativo pode usar as extensões do IBM MQ JMS para executar as seguintes funções:

- Crie connection factories e destinos dinamicamente no tempo de execução, em vez de recuperá-los como objetos administrados a partir de um namespace Java Naming and Directory Interface (JNDI)

- Configure as propriedades de connection factories e destinos

Criando connection factories

Para criar um connection factory, um aplicativo pode usar o construtor MQConnectionFactory, conforme mostrado no exemplo a seguir:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

Esta instrução cria um objeto MQConnectionFactory com os valores padrão para todas as suas propriedades, o que significa que o aplicativo se conecta ao gerenciador de filas padrão no modo de ligações. Se desejar que um aplicativo se conecte no modo cliente ou se conecte a um gerenciador de filas diferente do gerenciador de filas padrão, o aplicativo deve configurar as propriedades apropriadas do objeto MQConnectionFactory antes de criar a conexão. Para obter informações sobre como fazer isso, consulte “Configurando as propriedades de connection factories” na página 219.

Um aplicativo pode criar connection factories dos tipos a seguir de maneira semelhante:

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

Configurando as propriedades de connection factories

Um aplicativo pode configurar as propriedades de um connection factory chamando os métodos apropriados do connection factory. O connection factory pode ser um objeto administrado ou um objeto criado dinamicamente no tempo de execução.

Considere o seguinte código, por exemplo:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

Este código cria um objeto MQConnectionFactory e, em seguida, define cinco propriedades do objeto. O efeito de configurar estas propriedades é que o aplicativo se conecta ao gerenciador de filas QM1 no modo cliente usando um canal de MQI chamado QM1.SVR. O gerenciador de filas está em execução em um sistema com nome do host HOST1 e o listener para o gerenciador de filas está atendendo no número da porta 1415.

Um aplicativo que usa uma conexão em tempo real com um broker pode usar apenas o estilo de sistema de mensagens de publicação/assinatura. Ele não pode usar o estilo ponto a ponto do sistema de mensagens.

Apenas determinadas combinações de propriedades de um connection factory são válidas. Para obter informações sobre as combinações válidas, consulte Dependências entre propriedades de objetos do IBM MQ classes for JMS.

Para obter mais informações sobre as propriedades de um connection factory, além dos métodos usados para configurar as propriedades dele, consulte Propriedades de objetos do IBM MQ classes for JMS.

Criando destinos

Para criar um objeto de Fila, um aplicativo pode usar o construtor MQQueue, conforme mostrado no exemplo a seguir:

```
MQQueue q1 = new MQQueue("Q1");
```

Esta instrução cria um objeto MQQueue com os valores padrão para todas as suas propriedades. O objeto representa uma fila do IBM MQ chamada Q1 que pertence ao gerenciador de filas local. Essa fila pode ser uma fila local, uma fila de alias ou uma definição de fila remota.

Uma forma alternativa do construtor MQQueue possui dois parâmetros, conforme mostrado no exemplo a seguir:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

O objeto MQQueue criado por esta instrução representa uma fila IBM MQ chamada Q2, que é de propriedade do gerenciador de filas QM2. O gerenciador de filas identificado dessa forma pode ser o gerenciador de filas locais ou um gerenciador de filas remotas. Se for um gerenciador de filas remotas, o IBM MQ deverá ser configurado de forma que, quando o aplicativo enviar uma mensagem a esse destino, o WebSphere MQ possa rotear a mensagem do gerenciador de filas locais para o gerenciador de filas remotas.

O construtor MQQueue também pode aceitar um Identificador Uniforme de Recurso (URI) de fila como um único parâmetro. Um URI de fila é uma sequência que especifica o nome de uma fila do IBM MQ e, opcionalmente, o nome do gerenciador de filas que possui a fila, e uma ou mais propriedades do objeto MQQueue. A instrução a seguir contém um exemplo de um URI de fila:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

O objeto MQQueue criado por esta instrução representa uma fila IBM MQ chamada Q3 que é de propriedade do gerenciador de filas QM3, e todas as mensagens enviadas para este destino são persistentes e têm prioridade de 5. Para obter mais informações sobre URIs de filas, consulte “Identificadores uniformes de recursos (URIs)” na página 226. Para obter uma maneira alternativa de configurar as propriedades de um objeto MQQueue, consulte “Configurando as propriedades de destinos” na página 220.

Para criar um objeto Tópico, um aplicativo pode usar o construtor MQTopic, conforme mostrado no exemplo a seguir:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

Essa instrução cria um objeto MQTopic com os valores padrão para todas as suas propriedades. O objeto representa um tópico chamado Sport/Football/Results.

O construtor MQTopic também pode aceitar um URI de tópico como um parâmetro. Um URI de tópico é uma sequência que especifica o nome de um tópico e, opcionalmente, uma ou mais propriedades do objeto MQTopic. A instrução a seguir contém um exemplo de um URI de tópico:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

O objeto MQTopic criado por esta instrução representa um tópico chamado Esporte/Tênis/Resultados, e todas as mensagens enviadas para este destino são não persistentes e têm uma prioridade de 0. Para obter mais informações sobre URIs de tópicos, consulte “Identificadores uniformes de recursos (URIs)” na página 226. Para obter uma maneira alternativa de configurar as propriedades de um objeto MQTopic, consulte “Configurando as propriedades de destinos” na página 220.

Configurando as propriedades de destinos

Um aplicativo pode configurar as propriedades de um destino chamando os métodos apropriados do destino. O destino pode ser um objeto administrado ou um objeto criado dinamicamente no tempo de execução.

Considere o seguinte código, por exemplo:

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

Este código cria um objeto MQQueue e, em seguida, configura duas propriedades do objeto. O efeito de configurar estas propriedades é que todas as mensagens enviadas ao destino são persistentes e têm uma prioridade 5.

Um aplicativo pode configurar as propriedades de objeto MQTopic de maneira semelhante, conforme mostrado no exemplo a seguir:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

Este código cria um objeto MQTopic e, em seguida, configura duas propriedades do objeto. O efeito de configurar estas propriedades é que todas as mensagens enviadas ao destino são não persistentes e têm prioridade de 0.

Para obter mais informações sobre as propriedades de um destino, além dos métodos usados para configurar as propriedades dele, consulte Propriedades de objetos do IBM MQ classes for JMS.

Linux AIX *Conectando ao IBM MQ a partir de um aplicativo JMS*

Para construir uma conexão, um aplicativo JMS usa um objeto **ConnectionFactory** para criar um objeto **Connection** , em seguida, inicia a conexão.

Para JMS 2.0 e posterior, os aplicativos geralmente se conectam a um provedor de sistemas de mensagens usando um objeto **ConnectionFactory** e o método `createContext()` .

Em versões anteriores do JMS, primeiro você tinha que usar `createConnection` para criar um objeto **Connection** e, em seguida, iniciar a chamada de conexão `getSession()` para criar um objeto **Session** que poderia executar operações do sistema de mensagens

Um objeto **JMSContext** efetivamente encapsula os objetos **Connection** e **Session** . Se você desejar usar a abordagem tradicional e criar a conexão e os objetos de sessão diretamente, consulte “Construindo uma conexão em um aplicativo JMS” na página 222 e “Criando uma sessão em um aplicativo JMS” na página 222

Para criar um objeto **JMSContext** , um aplicativo usa o método `createContext()` de um objeto **ConnectionFactory** , conforme mostrado no exemplo a seguir:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

Quando uma conexão JMS é criada, o IBM MQ classes for JMS cria uma manipulação de conexões (Hconn) e inicia uma conversa com o gerenciador de filas.

Nota: Observe que o ID do processo do aplicativo é usado como a identidade do usuário padrão a ser transmitida para o gerenciador de filas. Se o aplicativo estiver em execução no modo de transporte do cliente, esse ID de processo deve existir, com as autorizações relevantes, no servidor. Se desejar que uma identidade diferente seja usada, use o método `createConnection(username,password)` .

V 9.4.0 Esse mecanismo também pode ser usado para fornecer um token de autenticação, consulte Obtendo um token de autenticação de seu emissor de token escolhido

JMS 1.0 *Construindo uma conexão em um aplicativo JMS*

Para construir uma conexão no JMS 1.0, um aplicativo JMS usa um objeto ConnectionFactory para criar um objeto Connection e, em seguida, inicia a conexão.

Para criar um objeto Connection, um aplicativo usa o método createConnection() de um objeto ConnectionFactory, conforme mostrado no exemplo a seguir:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

Quando uma conexão do JMS é criada, o IBM MQ classes for JMS cria uma manipulação de conexões (Hconn) e inicia uma conversa com o gerenciador de filas.

A interface QueueConnectionFactory e a interface TopicConnectionFactory herda, cada uma, o método createConnection() da interface ConnectionFactory. Portanto, é possível usar o método createConnection() para criar um objeto específico do domínio, conforme mostrado no exemplo a seguir:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

Esse fragmento de código cria um objeto QueueConnection. Um aplicativo pode agora executar uma operação independente de domínio nesse objeto ou uma operação que é aplicável apenas ao domínio ponto a ponto. Entretanto, se o aplicativo tenta executar uma operação que é aplicável apenas ao domínio de publicação/assinatura, uma exceção IllegalStateException é emitida com a seguinte mensagem:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

Isso ocorre porque a conexão foi criada a partir de um connection factory específico do domínio.

Nota: Observe que o ID do processo do aplicativo é usado como a identidade do usuário padrão a ser transmitida para o gerenciador de filas. Se o aplicativo estiver em execução no modo de transporte do cliente, esse ID de processo deve existir, com as autorizações relevantes, no servidor. Se você deseja usar uma identidade, use o método createConnection (nome do usuário, senha).

A especificação do JMS afirma que uma conexão é criada no estado `stopped`. Até que uma conexão é iniciada, um consumidor de mensagem que está associado à conexão não pode receber nenhuma mensagem. Para iniciar uma conexão, um aplicativo usa o método start() de um objeto Connection, conforme mostrado no exemplo a seguir:

```
connection.start();
```

V 9.4.0 Esse mecanismo também pode ser usado para fornecer um token de autenticação, consulte Obtendo um token de autenticação de seu emissor de token escolhido

JMS 1.0 *Criando uma sessão em um aplicativo JMS*

Para criar uma sessão no JMS 1.0, um aplicativo JMS usa o método createSession() de um objeto de Conexão

O método createSession() possui dois parâmetros:

1. Um parâmetro que especifica se a sessão está transacionada ou não transacionada
2. Um parâmetro que especifica o modo de confirmação para a sessão

Por exemplo, o código a seguir cria uma sessão que não está transacionada e tem um modo de confirmação de AUTO_ACKNOWLEDGE:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Quando uma sessão JMS é criada, o IBM MQ classes for JMS cria uma manipulação de conexões (Hconn) e inicia uma conversação com o gerenciador de filas.

Um objeto Session, e qualquer objeto MessageProducer ou MessageConsumer criado a partir dele, não pode ser usado simultaneamente por diferentes encadeamentos de um aplicativo multiencadeado. A maneira mais simples de garantir que esses objetos não sejam usados simultaneamente é criar um objeto Session separado para cada encadeamento.

V 9.4.0 Esse mecanismo também pode ser usado para fornecer um token de autenticação, consulte Obtendo um token de autenticação de seu emissor de token escolhido

Sessões transacionadas em aplicativos JMS
Os aplicativos JMS podem executar transações locais primeiro criando uma sessão transacionada. Um aplicativo pode confirmar ou retroceder uma transação.

Os aplicativos JMS podem executar transações locais. Uma transação local é uma transação que envolve mudanças apenas para os recursos do gerenciador de filas ao qual o aplicativo está conectado. Para executar transações locais, um aplicativo deve primeiro criar uma sessão transacionada chamando o método createSession() de um objeto Connection, especificando como um parâmetro que a sessão é transacionada. Em seguida, todas as mensagens enviadas e recebidas dentro da sessão são agrupadas em uma sequência de transações. Uma transação terminará quando o aplicativo confirmar ou recuperar as mensagens que ele enviou e recebeu desde o início da transação.

Para confirmar uma transação, um aplicativo chama o método commit() do objeto Session. Quando uma transação for confirmada, todas as mensagens enviadas dentro da transação se tornarão disponíveis para entrega para outros aplicativos e todas as mensagens recebidas dentro da transação serão confirmadas para que o servidor de sistema de mensagens não tente entregá-las ao aplicativo novamente. No domínio ponto a ponto, o servidor de sistema de mensagens também remove as mensagens recebidas a partir de suas filas.

Para retroceder uma transação, um aplicativo chama o método rollback() do objeto Session. Quando uma transação for retrocedida, todas as mensagens enviadas dentro da transação serão descartadas pelo servidor de sistema de mensagens e todas as mensagens recebidas dentro da transação se tornarão disponíveis para entrega novamente. No domínio de ponto a ponto, as mensagens que foram recebidas são colocadas de volta em suas filas e se tornam visíveis a outros aplicativos novamente.

Uma nova transação será iniciada automaticamente quando um aplicativo criar uma sessão transacionada ou chamar o método commit() ou rollback(). Portanto, uma sessão transacionada sempre possui uma transação ativa.

Quando um aplicativo fechar uma sessão transacionada, um retrocesso implícito ocorrerá. Quando um aplicativo fechar uma conexão, um retrocesso implícito ocorrerá para todas as sessões transacionadas da conexão.

Se um aplicativo terminar sem fechar uma conexão, um retrocesso implícito também ocorrerá para todas as sessões transacionadas da conexão.

Uma transação é completamente contida dentro de uma sessão transacionada. Uma transação não pode abranger as sessões. Isso significa que não é possível para um aplicativo enviar e receber mensagens em duas ou mais sessões transacionadas e, em seguida, confirmar ou retroceder todas estas ações como uma única transação.

Confirmação dos modos de sessões do JMS
Cada sessão que não é transacionada possui um modo de confirmação que determina como as mensagens recebidas pelo aplicativo são confirmadas. Três modos de confirmação estão disponíveis e a opção de modo de confirmação afeta o design do aplicativo.

Se uma sessão não for transacionada, a maneira que as mensagens recebidas pelo aplicativo são confirmadas será determinada pelo modo de confirmação da sessão. Três modos de confirmação são descritos nos parágrafos a seguir:

AUTO_ACKNOWLEDGE

A sessão confirma automaticamente cada mensagem recebida pelo aplicativo.

Se as mensagens forem entregues de forma síncrona para o aplicativo, a sessão confirmará o recebimento de uma mensagem toda vez que uma chamada Receive for concluída com sucesso. Se as mensagens forem entregues de forma assíncrona, a sessão confirmará o recebimento de uma mensagem toda vez que uma chamada ao método onMessage() de um listener de mensagens for concluída com sucesso.

Se o aplicativo receber uma mensagem com sucesso, mas uma falha evitar a ocorrência de confirmação, a mensagem se tornará disponível para a entrega novamente. O aplicativo deve, portanto, ser capaz de manipular uma mensagem que é entregue novamente.

DUPS_OK_ACKNOWLEDGE

A sessão confirma as mensagens recebidas pelo aplicativo em momentos que ele seleciona.

O uso desse modo de confirmação reduz a quantidade de trabalho que a sessão deve fazer, mas uma falha que evita a confirmação de mensagens pode fazer com que mais de uma mensagem se torne disponível para entrega novamente. O aplicativo deve, portanto, ser capaz de manipular mensagens que são entregues novamente.

Restrição: Nos modos AUTO_ACKNOWLEDGE e DUPS_OK_ACKNOWLEDGE, o JMS não suporta um aplicativo que lança uma exceção não manipulada em um listener de mensagem. Isso significa que as mensagens serão sempre confirmadas quando o listener de mensagem retornar, independentemente de se ele foi processado com sucesso (contanto que quaisquer falhas não sejam fatais e não evitem que o aplicativo continue). Se você requerer melhor controle de confirmação da mensagem, use o CLIENT_ACKNOWLEDGE ou modos transacionados, que fornecem ao aplicativo controle total das funções de confirmação.

CLIENT_ACKNOWLEDGE

O aplicativo confirma as mensagens que ele recebe chamando o método Acknowledge da classe Message.

O aplicativo pode confirmar o recebimento de cada mensagem individualmente ou receber um lote de mensagens e chamar o método Acknowledge apenas para a última mensagem que ele recebe. Quando o método Acknowledge for chamado, todas as mensagens recebidas desde a última vez que o método foi chamado serão confirmadas.

Juntamente com qualquer um destes modos de confirmação, um aplicativo pode interromper e reiniciar a entrega de mensagens em uma sessão chamando o método Recover da classe Session. As mensagens recebidas, mas não confirmadas anteriormente são entregues novamente. No entanto, elas não podem ser entregues na mesma sequência em que foram entregues anteriormente. No entretanto, mensagens de prioridade superior podem ter chegado e algumas das mensagens originais podem ter expirado. No domínio ponto a ponto, algumas das mensagens originais podem ter sido consumidas por outro aplicativo.

Um aplicativo pode determinar se uma mensagem está sendo entregue novamente examinando o conteúdo do campo de cabeçalho JMSRedelivered da mensagem. O aplicativo faz isso chamando o método getJMSRedelivered() da classe Message.

Criando destinos em um aplicativo JMS

Em vez de recuperar destinos como objetos administrados a partir de um namespace Java Naming and Directory Interface (JNDI), um aplicativo JMS pode usar uma sessão para criar destinos dinamicamente no tempo de execução. Um aplicativo pode usar um identificador uniforme de recursos (URI) para identificar uma fila do IBM MQ ou um tópico e, como opção, especificar uma ou mais propriedades de um objeto Queue ou Topic.

Usando uma sessão para criar objetos Queue

Para criar um objeto Queue, um aplicativo pode usar o método createQueue() de um objeto Session, conforme mostrado no exemplo a seguir:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

Esse código cria um objeto Queue com os valores padrão para todas as suas propriedades. O objeto representa uma fila do IBM MQ chamada Q1 que pertence ao gerenciador de filas local. Essa fila pode ser uma fila local, uma fila de alias ou uma definição de fila remota.

O método createQueue() também aceita um URI de URI como um parâmetro. Um URI de fila é uma sequência que especifica o nome de uma fila do IBM MQ e, como opção, o nome do gerenciador de filas que possui a fila e uma ou mais propriedades do objeto Queue. A instrução a seguir contém um exemplo de um URI de fila:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

O objeto Fila criado por esta instrução representa uma fila IBM MQ chamada Q2 que é de propriedade de um gerenciador de filas chamado QM2, e todas as mensagens enviadas para este destino são persistentes e têm prioridade de 5. O gerenciador de filas identificado dessa forma pode ser o gerenciador de filas locais ou um gerenciador de filas remotas. Se for um gerenciador de filas remotas, o IBM MQ deverá ser configurado de forma que, quando o aplicativo enviar uma mensagem a esse destino, o WebSphere MQ possa rotear a mensagem do gerenciador de filas locais para o gerenciador de filas QM2. Para obter mais informações sobre URIs, consulte “Identificadores uniformes de recursos (URIs)” na página 226.

Observe que o parâmetro no método createQueue() contém informações específicas do provedor. Portanto, usar o método createQueue() para criar um objeto Queue, em vez de recuperar um objeto Queue como um objeto administrado de um namespace JNDI, pode tornar seu aplicativo menos móvel.

Um aplicativo pode criar um objeto TemporaryQueue usando o método createTemporaryQueue() de um objeto Session, conforme mostrado no exemplo a seguir:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

Embora uma sessão seja usada para criar uma fila temporária, o escopo de uma fila temporária é a conexão que foi usada para criar a sessão. Qualquer uma das sessões da conexão pode criar produtores e consumidores de mensagens para a fila temporária. A fila temporária permanece até o término da conexão ou até o aplicativo excluir explicitamente a fila temporária usando o método TemporaryQueue.delete(), o que ocorrer primeiro.

Quando um aplicativo cria uma fila temporária, o IBM MQ classes for JMS cria uma fila dinâmica no gerenciador de filas ao qual o aplicativo está conectado. A propriedade TEMPMODEL do connection factory especifica o nome da fila modelo usada para criar a fila dinâmica e a propriedade TEMPQPREFIX do connection factory especifica o prefixo usado para formar o nome da fila dinâmica.

Usando uma sessão para criar objetos Topic

Para criar um objeto Topic, um aplicativo pode usar o método createTopic() de um objeto Session, conforme mostrado no exemplo a seguir:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

Este código cria um objeto Topic com os valores padrão para todas as suas propriedades. O objeto representa um tópico chamado Sport/Football/Results.

O método createTopic() também aceita um URI de tópico como um parâmetro. Um URI de tópico é uma sequência que especifica o nome de um tópico e, como opção, uma ou mais propriedades do objeto Topic. O código a seguir contém um exemplo de um URI de tópico:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

O objeto Tópico criado por este código representa um tópico chamado Esporte/Tênis/Resultados, e todas as mensagens enviadas para este destino são não persistentes e têm prioridade de 0. Para obter mais informações sobre URIs de tópicos, consulte “Identificadores uniformes de recursos (URIs)” na página 226.

Observe que o parâmetro no método createTopic() contém informações específicas do provedor. Portanto, usar o método createTopic() para criar um objeto Topic, em vez de recuperar um objeto Topic como um objeto administrado a partir de um namespace JNDI, pode tornar seu aplicativo menos móvel.

Um aplicativo pode criar um objeto TemporaryTopic usando o método createTemporaryTopic() de um objeto Session, conforme mostrado no exemplo a seguir:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

Embora uma sessão seja usada para criar um tópico temporário, o escopo de um tópico temporário é a conexão que foi usada para criar a sessão. Qualquer uma das sessões da conexão pode criar produtores e consumidores de mensagens para o tópico provisório. O tópico provisório permanece até o término da conexão ou até o aplicativo excluir explicitamente o tópico provisório usando o método TemporaryTopic.delete(), o que ocorrer primeiro.

Quando um aplicativo cria um tópico temporário, IBM MQ classes for JMS cria um tópico com um nome que começa com os caracteres TEMP/*tempTopicPrefix*, em que *tempTopicPrefix* é o valor da propriedade TEMPTOPICPREFIX do connection factory.

Identificadores uniformes de recursos (URIs)

Um URI de fila é uma sequência que especifica o nome de uma fila do IBM MQ e, como opção, o nome do gerenciador de filas que possui a fila e uma ou mais propriedades do objeto Queue criado pelo aplicativo. Um URI de tópico é uma sequência que especifica o nome de um tópico e, como opção, uma ou mais propriedades do objeto Topic criado pelo aplicativo.

Um URI de fila tem o formato a seguir:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Um URI de tópico tem o formato a seguir:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

As variáveis nesses formatos têm os significados a seguir:

qMgrName
O nome do gerenciador de filas que possui a fila identificada pelo URI.

O gerenciador de filas pode ser o gerenciador de filas locais ou um gerenciador de filas remotas. Se for um gerenciador de filas remotas, o IBM MQ deverá ser configurado para que, quando um aplicativo enviar uma mensagem para a fila, o WebSphere MQ possa rotear a mensagem do gerenciador de filas locais para o gerenciador de filas remotas.

Se nenhum nome for especificado, o gerenciador de filas locais será assumido.

qName
O nome da fila do IBM MQ.

A fila pode ser uma fila local, uma fila de alias ou uma definição de fila remota.

Para as regras para criação de nomes de filas, consulte Regras para nomenclatura de objetos IBM MQ.

topicName
O nome do tópico.

Para as regras para criação de nomes de tópicos, consulte Regras para nomenclatura de objetos IBM MQ. Evite o uso dos caracteres curingas +, #, * e ? em nomes de tópicos. Nomes de tópicos contendo esses caracteres podem causar resultados inesperados quando assinados. Consulte Combinando sequências de tópicos.

***propertyName1*, *propertyName2*, ...**
Os nomes das propriedades do objeto Queue ou Topic criado pelo aplicativo. Tabela 35 na página 227 lista os nomes de propriedades válidos que podem ser usados em um URI.

Se nenhuma propriedade for especificada, o objeto Queue ou Topic terá os valores padrão para todas as suas propriedades.

***propertyValue1*, *propertyValue2*, ...**
Os valores das propriedades do objeto Queue ou Topic criado pelo aplicativo. Tabela 35 na página 227 lista os valores de propriedades válidos que podem ser usados em um URI.

Os colchetes ([]) denotam um componente opcional e as reticências (...) significam que a lista de pares de nome-valor de propriedades, se presente, pode conter um ou mais pares de nome-valor.

Tabela 35 na página 227 lista os nomes de propriedades válidos e os valores válidos que podem ser usados em URIs de fila e de tópico. Embora a ferramenta de administração do IBM MQ JMS use constantes simbólicas para os valores de propriedades, os URIs não podem conter constantes simbólicas.

Tabela 35. Nomes de propriedades e valores válidos para uso nos URIs de fila e tópico

Nome da Propriedade	**Descrição**	**Valores Válidos**
CCSID	Como os dados de caracteres no corpo de uma mensagem são representados quando o IBM MQ classes for JMS encaminha a mensagem para o destino	• Qualquer identificador de conjunto de caracteres codificado suportado pelo IBM MQ.
codificação	Como os dados numéricos no corpo de uma mensagem são representados quando o IBM MQ classes for JMS encaminha a mensagem ao destino	• Qualquer valor válido para o campo *Codificação* em um descritor de mensagem do IBM MQ.

Tabela 35. Nomes de propriedades e valores válidos para uso nos URIs de fila e tópico (continuação)

Nome da Propriedade	**Descrição**	**Valores Válidos**
expiração	O tempo de vida das mensagens enviadas ao destino	• -2 - Conforme especificado na chamada send() ou, se não especificado na chamada send(), o tempo de vida padrão do produtor de mensagem. • 0 - Uma mensagem enviada ao destino nunca expira. • Um número inteiro positivo que especifica o tempo de vida em milissegundos.
multicast	A configuração de multicast para um tópico ao usar uma conexão em tempo real com um broker	A lista a seguir contém os valores válidos. Está associado a cada valor o valor correspondente da propriedade MULTICAST conforme usado na ferramenta de administração do IBM MQ JMS. Para obter uma descrição da propriedade MULTICAST e dos respectivos valores válidos, consulte Propriedades de objetos do IBM MQ classes for JMS. • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED
persistence	A persistência de mensagens enviadas ao destino	• -2 - Conforme especificado na chamada send() ou, se não especificado na chamada send(), a persistência padrão do produtor de mensagem. • -1 - Conforme especificado pelo atributo *DefPersistence* da fila ou do tópico do IBM MQ. • 1 - Não persistente. • 2 - Persistente. • 3 - Equivalente ao valor HIGH para a propriedade PERSISTENCE conforme usada na ferramenta de administração do IBM MQ JMS. Para obter uma explicação desse valor, consulte “Mensagens persistentes do JMS” na página 258.

Tabela 35. Nomes de propriedades e valores válidos para uso nos URIs de fila e tópico (continuação)		
Nome da Propriedade	**Descrição**	**Valores Válidos**
priority	A prioridade de mensagens enviadas ao destino	• -2 - Conforme especificado na chamada send() ou, se não especificado na chamada send(), a prioridade padrão do produtor de mensagem. • -1 - Conforme especificado pelo atributo *DefPriority* da fila ou do tópico do IBM MQ. • Um número inteiro no intervalo de 0 a 9 que especifica a prioridade de mensagens enviadas ao destino.
targetClient	Se as mensagens enviadas ao destino contêm um cabeçalho MQRFH2	• 0 - As mensagens contêm um cabeçalho MQRFH2. • 1 - As mensagens não contêm um cabeçalho MQRFH2.

Por exemplo, o URI a seguir identifica uma fila do IBM MQ chamada Q1 que pertence ao gerenciador de filas locais. Um objeto Queue criado usando esse URI tem os valores padrão para todas as suas propriedades.

```
queue:///Q1
```

O URI a seguir identifica uma fila do IBM MQ chamada Q2 que é de propriedade de um gerenciador de filas chamado QM2. Todas as mensagens enviadas para este destino têm uma prioridade de 6. As propriedades remanescentes do objeto Fila criado usando esta URI possuem seus valores padrão.

```
queue://QM2/Q2?priority=6
```

O URI a seguir identifica um tópico chamado Sport/Athletics/Results. Todas as mensagens enviadas para este destino são não persistentes e têm prioridade de 0. As propriedades remanescentes do objeto Tópico criado usando esta URI possuem seus valores padrão.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

Enviando Mensagens em um aplicativo JMS

Antes de um aplicativo JMS poder enviar mensagens para um destino, ele deve primeiro criar um objeto MessageProducer para o destino. Para enviar uma mensagem para o destino, o aplicativo cria um objeto de mensagem e, em seguida, chama o método send() do objeto MessageProducer.

Um aplicativo usa um objeto MessageProducer para enviar mensagens. Um aplicativo normalmente cria um objeto MessageProducer para um destino específico, que pode ser uma fila ou um tópico, para que todas as mensagens enviadas usando o produtor de mensagens sejam enviadas ao mesmo destino. Portanto, antes de um aplicativo poder criar um objeto MessageProducer, ele deve primeiro criar um objeto Queue ou Topic. Para obter informações sobre como criar um objeto Queue ou Topic, consulte os tópicos a seguir:

- “Usando JNDI para recuperar objetos administrados em um aplicativo JMS ou Jakarta Messaging” na página 209
- “usando as extensões IBM JMS” na página 211

- “Usando as extensões IBM MQ JMS” na página 218
- “Criando destinos em um aplicativo JMS” na página 225

Para criar um objeto MessageProducer, um aplicativo usa o método createProducer() de um objeto de sessão conforme mostrado no exemplo a seguir:

```
MessageProducer producer = session.createProducer(destination);
```

O parâmetro `destination` é um objeto Queue ou Topic que o aplicativo criou anteriormente.

Antes de um aplicativo poder enviar uma mensagem, ele deve criar um objeto de mensagem. O corpo de uma mensagem contém os dados do aplicativo, e o JMS define cinco tipos de corpo da mensagem:

- Bytes
- Mapa
- Object
- Fluxo
- Texto

Cada tipo de corpo da mensagem tem sua própria interface do JMS, que é uma sub-interface da interface de mensagem, e um método na interface de sessão para criar uma mensagem com esse tipo de corpo. Por exemplo, a interface para uma mensagem de texto é chamada TextMessage, e um aplicativo usa o método createTextMessage() de um objeto de sessão para criar uma mensagem de texto, conforme mostrado na seguinte instrução:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Para obter mais informações sobre mensagens e corpos de mensagem, consulte “Mensagens JMS” na página 146.

Para enviar uma mensagem, um aplicativo usa o método send() de um objeto MessageProducer conforme mostrado no exemplo a seguir:

```
producer.send(outMessage);
```

Um aplicativo pode usar o método send() para enviar mensagens em qualquer domínio de mensagens. A natureza do destino determina qual domínio de mensagens será usado. No entanto, TopicPublisher, a sub-interface do MessageProducer que é específica para o domínio de publicação/assinatura, também tem um método publish(), que pode ser usado no lugar do método send(). Os dois métodos são funcionalmente os mesmos.

Um aplicativo pode criar um objeto MessageProducer com nenhum destino especificado. Neste caso, o aplicativo deve especificar o destino ao chamar o método send().

Se um aplicativo enviar uma mensagem em uma transação, a mensagem não será entregue ao seu destino até que a transação seja confirmada. Isso significa que um aplicativo não pode enviar uma mensagem e receber uma resposta para a mensagem na mesma transação.

Um destino pode ser configurado de forma que quando um aplicativo enviar mensagens para ele, o IBM MQ classes for JMS encaminha a mensagem e retorna o controle de volta ao aplicativo sem determinar se o gerenciador de filas recebeu a mensagem com segurança. Isto é, às vezes, referido como *postagem assíncrona*. Para obter mais informações, consulte “Colocando mensagens de forma assíncrona no IBM MQ classes for JMS” na página 326.

Recebendo mensagens em um aplicativo JMS

Um aplicativo usa um consumidor de mensagens para receber mensagens. Um assinante de tópico permanente é um consumidor de mensagens que recebe todas as mensagens enviadas a um destino, incluindo aquelas enviadas enquanto o consumidor está inativo. Um aplicativo pode selecionar quais

mensagens deseja receber usando um seletor de mensagens e pode receber mensagens de forma assíncrona usando um listener de mensagem.

Um aplicativo usa um objeto MessageConsumer para receber mensagens. Um aplicativo cria um objeto MessageConsumer para um destino específico, que pode ser uma fila ou um tópico, para que todas as mensagens recebidas usando o consumidor de mensagens sejam recebidas do mesmo destino. Portanto, antes de um aplicativo poder criar um objeto MessageConsumer, ele deverá primeiramente criar um objeto Fila ou Tópico. Para obter informações sobre como criar um objeto Queue ou Topic, consulte os tópicos a seguir:

- "Usando JNDI para recuperar objetos administrados em um aplicativo JMS ou Jakarta Messaging" na página 209
- "usando as extensões IBM JMS" na página 211
- "Usando as extensões IBM MQ JMS" na página 218
- "Criando destinos em um aplicativo JMS" na página 225

Para criar um objeto MessageConsumer, um aplicativo usa o método createConsumer() de um objeto de Sessão, conforme mostrado no exemplo a seguir:

```
MessageConsumer consumer = session.createConsumer(destination);
```

O parâmetro `destination` é um objeto Queue ou Topic que o aplicativo criou anteriormente.

O aplicativo, então, usa o método receive() do objeto MessageConsumer para receber uma mensagem do destino, conforme mostrado no exemplo a seguir:

```
Message inMessage = consumer.receive(1000);
```

O parâmetro na chamada receive() especifica quanto tempo em milissegundos o método aguarda uma mensagem adequada chegar, se nenhuma mensagem estiver disponível imediatamente. Se você omitir esse parâmetro, a chamada é bloqueada indefinidamente até que uma mensagem adequada chegue. Se você não quiser que o aplicativo aguarde uma mensagem, use o método receiveNoWait() no lugar.

O método receive() retorna uma mensagem de um tipo específico. Por exemplo, quando um aplicativo recebe uma mensagem de texto, o objeto retornado pela chamada receive() é um objeto TextMessage.

No entanto, o tipo declarado do objeto retornado por uma chamada receive() é um objeto de Mensagem. Portanto, para extrair os dados do corpo de uma mensagem que acabou de ser recebida, o aplicativo deve lançar da classe de Mensagem para a subclasse mais específica, como TextMessage. Se o tipo da mensagem não for conhecido, o aplicativo poderá usar o operador `instanceof` para determinar o tipo. É sempre uma boa prática para um aplicativo determinar o tipo de uma mensagem antes do casting para que os erros possam ser manipulados com êxito.

O código a seguir usa o operador `instanceof` e mostra como extrair os dados do corpo de uma mensagem de texto:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Se um aplicativo enviar uma mensagem em uma transação, a mensagem não será entregue ao seu destino até que a transação seja confirmada. Isso significa que um aplicativo não pode enviar uma mensagem e receber uma resposta para a mensagem na mesma transação.

Se um consumidor de mensagens receber mensagens de um destino que esteja configurado para leitura antecipada, quaisquer mensagens não persistentes que estejam no buffer de leitura antecipada quando o aplicativo terminar serão descartadas.

No domínio de publicação/assinatura, o JMS identifica dois tipos de consumidores de mensagem, o assinante de tópico não durável e assinante de tópico durável, que são descritos nas duas seções a seguir.

Assinantes de tópico não duráveis

Um assinante de tópico não durável recebe apenas aquelas mensagens que são publicadas enquanto o assinante está ativo. Uma assinatura não durável é iniciada quando um aplicativo cria um assinante de tópico não durável e é finalizado quando o aplicativo fecha o assinante ou quando o assinante está fora do escopo. Como uma extensão no IBM MQ classes for JMS, um assinante de tópico não durável também recebe publicações retidas.

Para criar um assinante de tópico não durável, um aplicativo pode usar o método createConsumer independente de domínio (), especificando um objeto Topic como o destino. Como alternativa, um aplicativo pode usar o método createSubscriber() específico de domínio, conforme mostrado no exemplo a seguir:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

O parâmetro `topic` é um objeto do Tópico que o aplicativo criou anteriormente.

Assinantes de tópico duráveis

Restrição: Um aplicativo não pode criar os assinantes do tópico durável ao usar uma conexão em tempo real com um broker.

Um assinante de tópico durável recebe todas as mensagens que são publicadas durante a vida útil de uma assinatura durável. Essas mensagens incluem todas aquelas que são publicadas enquanto o assinante não está ativo. Como uma extensão no IBM MQ classes for JMS, um assinante de tópico durável também recebe publicações retidas.

Para criar um assinante de tópico durável, um aplicativo usa o método createDurableSubscriber() de um objeto de Sessão, conforme mostrado no exemplo a seguir:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

Na chamada createDurableSubscriber(), o primeiro parâmetro é um objeto do Tópico que o aplicativo criou anteriormente, e o segundo parâmetro é um nome que é usado para identificar a assinatura durável.

A sessão usada para criar um assinante de tópico durável deve ter um identificador de cliente associado. O identificador de cliente associado a uma sessão é o mesmo que o identificador do cliente para a conexão que é usado para criar a sessão. O identificador de cliente pode ser especificado configurando a propriedade CLIENTID do objeto ConnectionFactory. Como alternativa, um aplicativo pode especificar o identificador de cliente chamando o método setClientID() do objeto Conexão.

O nome que é usado para identificar uma assinatura durável deve ser exclusivo somente no identificador de cliente e, portanto, o identificador de cliente faz parte do identificador integral exclusivo de uma assinatura durável. Para continuar a usar uma assinatura durável que foi criada anteriormente, um aplicativo deve criar um assinante de tópico durável usando uma sessão com o mesmo identificador de cliente que o associado à assinatura durável e usando o mesmo nome de assinatura.

Uma assinatura durável é iniciada quando um aplicativo cria um assinante de tópico durável usando um identificador de cliente e nome de assinatura para a qual nenhuma assinatura durável existe atualmente. No entanto, uma assinatura durável não termina quando o aplicativo fecha o assinante de tópico durável. Para finalizar uma assinatura durável, um aplicativo deve chamar o método unsubscribe() de um objeto de Sessão que possui o mesmo identificador de cliente que o associado à assinatura durável. O parâmetro na chamada unsubscribe() é o nome de assinatura, conforme mostrado no exemplo a seguir:

```
session.unsubscribe("D_SUB_000001");
```

O escopo de uma assinatura durável é um gerenciador de filas. Se uma assinatura durável existir em um gerenciador de filas e um aplicativo conectado a outro gerenciador de filas criar uma assinatura durável com o mesmo identificador de cliente e nome de assinatura, as duas assinaturas duráveis serão completamente independentes.

Seletores de mensagens

Um aplicativo pode especificar que apenas aquelas mensagens que atendam a determinados critérios sejam retornadas pelas chamadas receive() sucessivas. Ao criar um objeto MessageConsumer, o aplicativo pode especificar uma expressão de Linguagem de Consulta Estruturada (SQL) que determina quais mensagens são recuperadas. Esta expressão SQL é chamada de *seletor de mensagem*. O seletor de mensagem pode conter os nomes de campos de cabeçalho de mensagem do JMS e propriedades de mensagem. Para obter informações sobre como construir um seletor de mensagem, consulte “Seletores de mensagens no JMS” na página 146.

O exemplo a seguir mostra como um aplicativo pode selecionar mensagens com base em uma propriedade definida pelo usuário chamado myProp:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

A especificação JMS não permite que um aplicativo mude o seletor de mensagem de um consumidor de mensagem. Após um aplicativo criar um consumidor de mensagem com um seletor de mensagem, o seletor de mensagem permanecerá durante a existência desse consumidor. Se um aplicativo precisar de mais de um seletor de mensagem, o aplicativo deverá criar um consumidor de mensagem para cada seletor de mensagem.

Observe que, quando um aplicativo é conectado a um gerenciador de filas da Versão 7, a propriedade MSGSELECTION do connection factory não tem efeito. Para otimizar o desempenho, toda seleção de mensagens é feita pelo gerenciador de filas.

Suprimindo as publicações locais

Um aplicativo pode criar um consumidor de mensagem que ignora as publicações feitas na própria conexão do consumidor. O aplicativo faz isso configurando o terceiro parâmetro em uma chamada createConsumer() para `true`, conforme mostrado no exemplo a seguir:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

Em uma chamada createDurableSubscriber(), o aplicativo faz isso configurando o quarto parâmetro para `true`, conforme mostrado no exemplo a seguir

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

Entrega assíncrona de mensagens

Um aplicativo pode receber mensagens de forma assíncrona registrando um listener de mensagem com um consumidor de mensagens. O listener de mensagem tem um método chamado onMessage, que é

chamado de maneira assíncrona quando uma mensagem adequada está disponível e cuja finalidade é processar a mensagem. O código a seguir ilustra o mecanismo:

JMS 3.0

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

Um aplicativo pode usar uma sessão, para receber mensagens de forma síncrona usando chamadas receive() ou para receber mensagens de forma assíncrona usando listeners de mensagens, mas não para ambos. Se um aplicativo precisar receber mensagens de forma síncrona e assíncrona, ele deve criar sessões separadas.

Assim que uma sessão for configurada para receber mensagens de forma assíncrona, os métodos a seguir não poderão ser chamados na sessão ou em objetos criados a partir dessa sessão:

- MessageConsumer.receive()
- MessageConsumer.receive(long)
- MessageConsumer.receiveNoWait()

- Session.acknowledge()
- MessageProducer.send(Destination, Message)
- MessageProducer.send(Destination, Message, int, int, long)
- MessageProducer.send(Message)
- MessageProducer.send(Message, int, int, long)
- MessageProducer.send(Destination, Message, CompletionListener)
- MessageProducer.send(Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send(Message, CompletionListener)
- MessageProducer.send(Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(Queue)
- Session.createBrowser(Queue, String)
- Session.createBytesMessage()
- Session.createConsumer(Destination)
- Session.createConsumer(Destination, String, boolean)
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Destination)
- Session.createQueue(String)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

Se qualquer um desses métodos é chamado, uma JMSException contendo a mensagem:

```
JMSCC0033: uma chamada de método síncrona não é permitida quando uma sessão está sendo usada de forma assíncrona: 'method name'
```

é lançado.

Recebendo mensagens suspeitas

Um aplicativo pode receber uma mensagem que não pode ser processada. Pode haver várias razões pelas quais a mensagem não pode ser processada, por exemplo, a mensagem pode ter um formato incorreto. Essas mensagens são descritas como mensagens suspeitas e requerem tratamento especial para impedir a mensagem que está sendo processada recursivamente.

Para obter detalhes sobre como manipular mensagens suspeitas, consulte “Manipulando mensagens suspeitas no IBM MQ classes for JMS” na página 238.

Padronizando tamanhos de buffer para se adequar às mensagens que estão sendo recebidas

Quando uma mensagem é recebida do IBM MQ por um aplicativo nãoJMS , um buffer de mensagem deve ser fornecido pelo aplicativo para a mensagem ser gravada. Aplicativos JMS não precisam criar manualmente um buffer. O IBM MQ classes for JMS cria e dimensiona buffers de mensagens automaticamente para adequar os tamanhos das mensagens que estão sendo recebidas. Para a maioria dos aplicativos, os buffers gerenciados automaticamente fornecem um equilíbrio adequado de desempenho e conveniência para o desenvolvedor de aplicativos Em determinadas circunstâncias, pode ser benéfico especificar o tamanho inicial do buffer de mensagem manualmente O tamanho inicial padrão de um buffer de recebimento do IBM MQ JMS é de 4 KB Se um aplicativo sempre receberá mensagens de 256 KB de tamanho, pode ser preferível configurar o tamanho do buffer inicial para 256 KB. Isso pode evitar a necessidade de o IBM MQ classes for JMS tentar e falhar em receber a mensagem em um buffer de 4 KB antes de redimensioná-la para 256 KB e recebê-la com êxito Para um aplicativo conectado pelo cliente, isso pode evitar a necessidade de um roundtrip de rede potencialmente desperdiçado enquanto o IBM MQ classes for JMS determina o tamanho correto do buffer a ser usado

O tamanho do buffer inicial pode ser configurado configurando a propriedade com.ibm.mq.jmqi.defaultMaxMsgSize Java para seu valor escolhido, em bytes. Observe que essa propriedade afeta todos os aplicativos IBM MQ JMS que estão em execução dentro do Java Virtual Machine, portanto, cuidado para não afetar adversamente outros consumidores de mensagens que recebem mensagens de um tamanho diferente.

O IBM MQ classes for JMS ainda tenta reduzir automaticamente o tamanho do buffer se várias mensagens menores que o tamanho configurado forem recebidas. Por padrão, isso acontece se forem recebidas 10 mensagens que são todas menores que o tamanho do buffer. Por exemplo, se 10 mensagens forem recebidas em uma linha com 128 KB de tamanho, o buffer será reduzido de 256 KB para 128 KB. Ele é então aumentado novamente quando mensagens maiores são recebidas. É possível configurar o número de mensagens que devem ser recebidas, antes que um buffer seja reduzido em tamanho. Por exemplo, isso pode ser útil se o aplicativo for conhecido por receber cinco mensagens grandes seguidas por 10 mensagens menores e, em seguida, outras cinco mensagens grandes.. Com as configurações padrão, o buffer seria reduzido após as 10 mensagens menores terem sido recebidas e precisariam aumentar novamente para as mensagens maiores. A Java propriedade do sistema com.ibm.mq.jmqi.smallMsgBufferReductionThreshold pode ser definida para o número de mensagens que devem ser recebidas antes que o tamanho do buffer seja reduzido Neste exemplo, ele poderia ser configurado como 20 para evitar que 10 mensagens menores reduzam o tamanho do buffer

As propriedades podem ser configuradas independentemente umas das outras. Por exemplo, é possível optar por deixar o tamanho do buffer inicial para seu valor padrão de 4 KB, mas aumentar o valor de com.ibm.mq.jmqi.smallMsgBufferReductionThreshold para que, quando o buffer for aumentado em tamanho, ele permaneça esse tamanho por mais tempo.

Se um grande número de códigos de retorno MQRC_TRUNCATED_MSG_FAILED (2080) forem vistos para seus aplicativos JMS em registros de estatísticas de MQI, isso pode ser uma indicação de que você se beneficiaria de configurar um tamanho de buffer inicial mais alto para esses aplicativos ou reduzir a frequência com a qual os tamanhos de buffer são reduzidos. No entanto, é importante observar que para um aplicativo de longa execução é provável que você veja apenas um número muito pequeno de códigos de retorno MQRC_TRUNCATED_MSG_FAILED. Isso ocorre porque geralmente o buffer é aumentado para o tamanho correto imediatamente após a primeira mensagem grande ser recebida e não é reduzido no tamanho, a menos que um número de mensagens menores sejam recebidas. Portanto, é possível que um grande número de MQRC_TRUNCATED_MSG_FAILED indique outras práticas de aplicativos ruins, como conectar ao IBM MQ para receber apenas uma ou duas mensagens antes de desconectar.

Recuperação de dados do usuário da assinatura

Se as mensagens que um aplicativo IBM MQ classes for JMS está consumindo de uma fila são colocadas por uma assinatura durável definida administrativamente, o aplicativo precisa acessar as informações de

dados do usuário que estão associadas à assinatura. Essas informações são incluídas na mensagem como uma propriedade.

Quando uma mensagem é consumida de uma fila que contém um cabeçalho RFH2 com a pasta MQPS, o valor associado à chave Sud, se existir, será incluído como uma propriedade Sequência no objeto de Mensagem do JMS retornado para o aplicativo IBM MQ classes for JMS. Para ativar a recuperação dessa propriedade da mensagem, a constante JMS_IBM_SUBSCRIPTION_USER_DATA na interface JmsConstants pode ser usada com o método a seguir para obter os dados do usuário da assinatura:

- 

jakarta.jms.Message.getStringProperty(java.lang.String)
- 

javax.jms.Message.getStringProperty(java.lang.String)

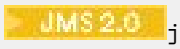
No exemplo a seguir, uma assinatura durável administrativa é definida usando o comando MQSC **DEFINE SUB**:

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

As cópias de mensagens que são publicadas na sequência de tópicos `PUBLIC` são colocadas na fila, `MY.SUBSCRIPTION.Q`. Os dados do usuário que estão associados à assinatura durável são então incluídos como uma propriedade na mensagem, que é armazenada na pasta MQPS do cabeçalho RFH2 com a chave Sud.

O aplicativo IBM MQ classes for JMS pode chamar:


```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```




```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

A Sequência a seguir é então retornada:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

Conceitos relacionados

"O cabeçalho MQRFH2 e JMS" na página 151

Tarefas relacionadas

Definindo uma assinatura administrativa

Referências relacionadas

DEFINE SUB

Interface JmsConstants

Fechando para um aplicativo IBM MQ classes for JMS

É importante para um aplicativo IBM MQ classes for JMS fechar certos objetos JMS explicitamente antes de parar. Finalizadores não podem ser chamados, portanto, não dependem deles para liberar recursos. Não permita que um aplicativo finalize com o rastreio compactado ativo.

A coleta de lixo sozinha não pode liberar todos os recursos do IBM MQ classes for JMS e IBM MQ de maneira oportuna, especialmente se um aplicativo criar muitos objetos de curta duração do JMS no nível da sessão ou inferior. É, portanto, importante que um aplicativo feche um objeto Conexão, Sessão, MessageConsumer ou MessageProducer quando ele não for mais necessário.

Se um aplicativo finalizar sem fechar uma Conexão, uma reversão implícita ocorre para todas as sessões transacionadas da conexão. Para assegurar que quaisquer mudanças feitas pelo aplicativo sejam confirmadas, feche a Conexão explicitamente antes de fechar o aplicativo.

Não use finalizadores em um aplicativo para fechar os objetos do JMS. Como os finalizadores podem não ser chamados, os recursos podem não ser liberados. Quando uma Conexão é fechada, ela fecha todas as Sessões que foram criadas a partir dela. Da mesma forma, MessageConsumers e MessageProducers criados a partir de uma Sessão são fechados quando a Sessão é fechada. No entanto, considere o

fechamento de Sessões, MessageConsumers e MessageProducers explicitamente para assegurar que os recursos sejam liberados de uma maneira oportuna.

Se a compactação de rastreio estiver ativada, encerramentos System.Halt() e términos JVM não controlados e anormais provavelmente resultarão em um arquivo de rastreio corrompido. Sempre que possível, desative o recurso de rastreio quando você tiver coletado as informações de rastreio de que você precisa. Se você estiver rastreando um aplicativo até uma finalização anormal, use a saída de rastreio descompactada.

Nota: Para se desconectar de um gerenciador de filas, um aplicativo JMS chama o método close() no objeto de conexão.

Manipulando mensagens suspeitas no IBM MQ classes for JMS

Uma mensagem suspeita é uma que não pode ser processada por um aplicativo de recebimento. Se uma mensagem suspeita for entregue a um aplicativo e retrocedida um número especificado de vezes, o IBM MQ classes for JMS poderá movê-la para uma fila de restauração.

Uma mensagem suspeita é uma mensagem que não pode ser processada por um aplicativo de recebimento. A mensagem pode ter um tipo inesperado ou conter informações que não podem ser tratadas pela lógica do aplicativo. Se uma mensagem suspeita for entregue a um aplicativo, o aplicativo será incapaz de processar e irá retrocedê-la para a fila de onde ela veio. Por padrão, o IBM MQ classes for JMS entregará novamente, repetidamente, a mensagem para o aplicativo. Isso poderá fazer com que o aplicativo seja travado em um loop continuamente tentando processar a mensagem suspeita e a retrocedendo.

Para evitar que isso aconteça, o IBM MQ classes for JMS pode detectar mensagens suspeitas e movê-las para um destino alternativo. Para fazer isso, o IBM MQ classes for JMS faz uso das propriedades a seguir:

- O valor do campo BackoutCount dentro do MQMD da mensagem que foi detectada.
- Os atributos da fila IBM MQ **BOTHRESH** (limite de restauração) e **BOQNAME** (fila de novo enfileiramento de restauração) da fila de entrada que contém a mensagem.

Sempre que uma mensagem for retrocedida por um aplicativo, o gerenciador de filas incrementará automaticamente o valor do campo BackoutCount para a mensagem.

Quando o IBM MQ classes for JMS detectar uma mensagem que tiver uma BackoutCount maior que zero, ele comparará o valor da BackoutCount com o valor do atributo **BOTHRESH**.

- Se a BackoutCount for menor que o valor do atributo **BOTHRESH**, o IBM MQ classes for JMS a entregará ao aplicativo para processamento.
- No entanto, se a BackoutCount for maior ou igual a **BOTHRESH**, então, a mensagem será considerada como uma mensagem suspeita. Nesta situação, o IBM MQ classes for JMS, em seguida, move a mensagem para a fila especificada pelo atributo **BOQNAME**. Se a mensagem não puder ser enviada para a fila de restauração, ela será movida para a fila de devoluções do gerenciador de filas ou descartada, dependendo das opções de relatório da mensagem.

Nota:

- Se o atributo **BOTHRESH** for deixado em seu valor padrão de `0`, então, a manipulação da mensagem suspeita será desativada. Isso significa que quaisquer mensagens suspeitas são colocadas de volta na fila de entrada.
- A outra coisa a ser observada é que o IBM MQ classes for JMS consulta os atributos **BOTHRESH** e **BOQNAME** para a fila na primeira vez em que ele detecta uma mensagem que tem uma BackoutCount maior que zero. Os valores desses atributos são, então, armazenados em cache e usados sempre que o IBM MQ classes for JMS encontrar uma mensagem que tenha uma BackoutCount maior que zero.

Configurando o seu sistema para executar a manipulação de mensagens suspeitas

A fila que o IBM MQ classes for JMS usa ao consultar os atributos **BOTHRESH** e **BOQNAME** depende do estilo do sistema de mensagens que estiver sendo executado:

- Para o sistema de mensagens ponto a ponto, é a fila local subjacente. Isso será importante quando um aplicativo JMS estiver consumindo mensagens de filas de alias ou filas de clusters.
- Para o sistema de mensagens de publicação/assinatura, uma fila gerenciada é criada para conter as mensagens para um aplicativo. O IBM MQ classes for JMS consulta a fila gerenciada para determinar os valores para os atributos **BOTHRESH** e **BOQNAME**.

  A fila gerenciada é criada por meio de uma fila modelo associada ao objeto Tópico que o aplicativo assinou e herda os valores dos atributos **BOTHRESH** e **BOQNAME** da fila modelo. A fila modelo que é usada depende de o aplicativo de recebimento ter obtido uma assinatura durável ou não durável:

  – A fila modelo usada para assinaturas duráveis é especificada pelo atributo **MDURMDL** do Tópico. O valor padrão desse atributo é `SYSTEM.DURABLE.MODEL.QUEUE`.

  – Para assinaturas não duráveis, a fila modelo que é usada é especificada pelo atributo **MNDURMDL**. O valor padrão do atributo **MNDURMDL** é `SYSTEM.NDURABLE.MODEL.QUEUE`.

Ao consultar os atributos **BOTHRESH** e **BOQNAME**, o IBM MQ classes for JMS:

- Abre a fila local ou a fila de destino para uma fila de alias.
- Consultar os atributos **BOTHRESH** e **BOQNAME**.
- Fecha a fila local ou a fila de destino para uma fila de alias.

As opções abertas que são usadas ao abrir a fila local ou a fila de destino para uma fila de alias, dependem da versão do IBM MQ classes for JMS que estiver sendo usada:

- Para o IBM MQ classes for JMS no IBM MQ 9.1.0 Fix Pack 1 e anterior ou o IBM MQ 9.1.1, se a fila local ou a fila de destino para uma fila de alias for uma fila de clusters, o IBM MQ classes for JMS abrirá a fila com as opções `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` e `MQOO_FAIL_IF_QUIESCING`. Isso significa que o usuário que estiver executando o aplicativo de recebimento deverá ter acesso de consulta e de obtenção à instância local da fila de clusters.

  O IBM MQ classes for JMS abrirá todos os outros tipos de fila local com as opções abertas `MQOO_INQUIRE` e `MQOO_FAIL_IF_QUIESCING`. Para que o IBM MQ classes for JMS consulte os valores dos atributos, o usuário que estiver executando o aplicativo de recebimento deverá ter acesso de consulta na fila local.
- Ao usar o IBM MQ classes for JMS no IBM MQ 9.1.0 Fix Pack 2 e mais recente ou para o IBM MQ 9.1.2 e mais recente, o usuário que executa o aplicativo de recebimento deve ter acesso à consulta na fila local, independentemente do tipo da fila.

Para mover mensagens suspeitas para uma fila de reenfileiramento de restauração ou para a fila de mensagens não entregues do gerenciador de filas, você deve conceder ao usuário que está executando as autoridades `put` e `passall` do aplicativo.

Processando mensagens suspeitas para aplicativos síncronos

Se um aplicativo receber mensagens de forma síncrona, chamando um dos métodos a seguir, o IBM MQ classes for JMS reenfileirará uma mensagem suspeita dentro da unidade de trabalho que estava ativa quando o aplicativo tentou obter a mensagem:

- JMSConsumer.receive()
- JMSConsumer.receive (tempo limite longo)
- JMSConsumer.receiveBody(Class<T> c)
- JMSConsumer.receiveBody(Class<T> c, long timeout)
- JMSConsumer.receiveBodyNoWait Class<T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive()
- MessageConsumer.receive (tempo limite longo)
- MessageConsumer.receiveNoWait()

- QueueReceiver.receive ()
- QueueReceiver.receive(tempo limite longo)
- QueueReceiver.receiveNoWait ()
- TopicSubscriber.receive ()
- TopicSubscriber.receive (tempo limite longo)
- TopicSubscriber.receiveNoWait ()

Isso significa que se o aplicativo estiver usando um contexto ou sessão do JMS transacionada, então, a movimentação da mensagem para a fila de restauração não será confirmada até que a transação seja confirmada.

Se o atributo **BOTHRESH** for configurado para um valor diferente de zero, o atributo **BOQNAME** também deverá ser configurado. Se o **BOTHRESH** for configurado para um valor maior que zero e o **BOQNAME** não tiver sido configurado, o comportamento será determinado pelas opções de relatório da mensagem:

- Se a mensagem tiver a opção de relatório MQRO_DISCARD_MSG configurada, a mensagem será descartada.
- Se a mensagem tiver a opção de relatório MQRO_DEAD_LETTER_Q especificada e o IBM MQ classes for JMS tentar mover a mensagem para a fila de mensagens não entregues do gerenciador de filas.
- Se a mensagem não tiver a MQRO_DISCARD_MSG ou MQRO_DEAD_LETTER_Q configurada, o IBM MQ classes for JMS tentará colocar a mensagem na fila de mensagens não entregues para o gerenciador de filas.

No caso de a tentativa de colocar a mensagem na fila de mensagens não entregues falhar por alguma razão, o que acontece com a mensagem será determinado pelo fato de o aplicativo de recebimento estar usando um contexto ou sessão transacionada ou não transacionada do JMS:

- Se o aplicativo de recebimento estiver usando um contexto ou sessão do JMS transacionada e a transação for confirmada, então, a mensagem será descartada.
- Se o aplicativo de recebimento estiver usando um contexto ou sessão do JMS transacionada e recuperar a transação, a mensagem será retornada para a fila de entrada.
- Se o aplicativo de recebimento tiver criado um contexto ou uma sessão do JMS não transacionada, a mensagem será descartada.

Processando mensagens suspeitas para aplicativos assíncronos

Se um aplicativo estiver recebendo mensagens de forma assíncrona por meio de um MessageListener, o IBM MQ classes for JMS reenfileirará mensagens suspeitas sem afetar a entrega da mensagem. O processo de reenfileiramento ocorre fora de qualquer unidade de trabalho associada à entrega de mensagem real para o aplicativo.

Se o **BOTHRESH** for configurado para um valor maior que zero e o **BOQNAME** não tiver sido configurado, o comportamento será determinado pelas opções de relatório da mensagem:

- Se a mensagem tiver a opção de relatório MQRO_DISCARD_MSG configurada, a mensagem será descartada.
- Se a mensagem tiver a opção de relatório MQRO_DEAD_LETTER_Q especificada e o IBM MQ classes for JMS tentar mover a mensagem para a fila de mensagens não entregues do gerenciador de filas.
- Se a mensagem não tiver a MQRO_DISCARD_MSG ou MQRO_DEAD_LETTER_Q configurada, o IBM MQ classes for JMS tentará colocar a mensagem na fila de mensagens não entregues para o gerenciador de filas.

Se a tentativa de colocar a mensagem na fila de mensagens não entregues falhar por alguma razão, o IBM MQ classes for JMS retornará a mensagem para a fila de entrada.

Para obter informações sobre como as especificações de ativação e os ConnectionConsumers manipulam mensagens suspeitas, consulte Removendo mensagens da fila no ASF.

O que acontece com uma mensagem quando ela é movida para a fila de restauração

Quando uma mensagem suspeita for reenfileirada para a fila de reenfileiramento de restauração, o IBM MQ classes for JMS incluirá um cabeçalho RFH2 nela (se ela já não tinha um) e atualizará alguns dos campos dentro do descritor de mensagens (MQMD).

Se a mensagem suspeita contiver um cabeçalho RFH2 (porque ela era uma mensagem do JMS, por exemplo), o IBM MQ classes for JMS mudará os campos a seguir dentro do MQMD ao mover a mensagem para a fila de reenfileiramento de restauração:

- O campo BackoutCount é reconfigurado para zero.
- O campo Expiração da mensagem é atualizado para refletir a expiração restante no momento em que a mensagem suspeita foi recebida pelo aplicativo JMS.

Se a mensagem suspeita não contiver um cabeçalho RFH2, o IBM MQ classes for JMS incluirá um e atualizará os campos a seguir no MQMD como parte do processamento de restauração:

- O campo BackoutCount é reconfigurado para zero.
- O campo Expiração da mensagem é atualizado para refletir a expiração restante no momento em que a mensagem suspeita foi recebida pelo aplicativo JMS.
- O campo Formato da mensagem é mudado para MQHRF2.
- O campo CCSID é mudado para 1208.
- O campo Codificação é modificado para ser 273.

Além disso, os campos CCSID e Codificação da mensagem suspeita são copiados para os campos CCSID e Codificação do cabeçalho RFH2, para assegurar que o encadeamento do cabeçalho da mensagem na fila de reenfileiramento de restauração esteja correto.

Conceitos relacionados

"Manipulando mensagens suspeitas no ASF" na página 343
No Application Server Facilities, a manipulação de mensagens suspeitas é tratada de modo ligeiramente diferente do que em outras partes no IBM MQ classes for JMS.

Exceções no IBM MQ classes for JMS

Um aplicativo IBM MQ classes for JMS deve manipular exceções lançadas pelas chamadas da API do JMS ou entregues a um manipulador de exceções.

IBM MQ classes for JMS relata problemas de tempo de execução lançando as exceções. O tipo de exceções que são lançadas, e a maneira como essas exceções devem ser manipuladas, depende da versão da especificação JMS que é usada pelo seu aplicativo:

- Os métodos nas interfaces que são definidas no JMS 1.1 e anterior lançam exceções verificadas. A classe base dessas exceções é `JMSException`. Para obter mais informações sobre como manipular exceções verificadas, consulte "Manipulando exceções verificadas" na página 242.
- Os métodos nas interfaces incluídas em JMS 2.0 lançam exceções desmarcadas A classe base para essas exceções é `JMSRuntimeException`. Para obter mais informações sobre como manipular exceções não verificadas, consulte "Manipulando exceções não verificadas" na página 245.

Também é possível registrar um `ExceptionListener` com um JMS `Connection` ou um `JMSContext`. Em seguida, as classes de MQ para JMS notificam o `ExceptionListener` quando um problema é detectado com uma conexão com o gerenciador de filas ou ao tentar entregar uma mensagem de forma assíncrona. Para obter mais informações, consulte "ExceptionListeners" na página 248.

Conceitos relacionados

Classes do IBM MQ for JMS

Referências relacionadas

ASYNCEXCEPTION

Manipulando exceções verificadas
Os métodos nas interfaces que são definidas no JMS 1.1 ou anterior lançam exceções verificadas. A classe base para essas exceções é `JMSException`. Portanto, a captura de `JMSExceptions` fornece uma maneira genérica de lidar com esses tipos de exceções.

Cada `JMSException` encapsula as seguintes informações:

- Uma mensagem de exceção específica do provedor, a qual o seu aplicativo pode obter chamando o método `Throwable.getMessage()`.
- Um código de erro específico do provedor, o qual o seu aplicativo pode obter chamando o método `JMSException.getErrorCode()`.
- Uma exceção vinculada. Uma exceção que é lançada por uma chamada da API de JMS 1.1 é frequentemente o resultado de um problema de nível inferior que é relatado por outra exceção que está vinculada a essa exceção. O seu aplicativo pode obter uma exceção vinculada chamando o método `JMSException.getLinkedException()` ou o método `Throwable.getCause()`.

Quando você usa a API JMS 1.1, a maioria das exceções que são lançadas pelo IBM MQ classes for JMS são instâncias de subclasses de `JMSException`. Essas subclasses implementam a interface `com.ibm.msg.client.jms.JmsExceptionDetail`, que fornece as seguintes informações adicionais:

- Uma explicação da mensagem de exceção. O seu aplicativo pode obter essa mensagem chamando o método `JmsExceptionDetail.getExplanation()`.
- Uma resposta do usuário recomendada à exceção. O seu aplicativo pode obter essa mensagem chamando o método `JmsExceptionDetail.getUserAction()`.
- As chaves para as inserções de mensagem na mensagem de exceção. O seu aplicativo pode obter um iterador para todas as chaves chamando o método `JmsExceptionDetail.getKeys()`.
- As inserções de mensagem na mensagem de exceção. Por exemplo, uma inserção de mensagem pode ser o nome da fila que causou a exceção e pode ser útil para o seu aplicativo acessar esse nome. O seu aplicativo pode obter a inserção da mensagem correspondente em uma chave especificada chamando o método `JmsExceptionDetail.getValue()`.

Todos os métodos na interface `JmsExceptionDetail` retornarão nulo se nenhum detalhe estiver disponível.

Por exemplo, se um aplicativo tentar criar um produtor de mensagem para uma fila do IBM MQ que não existe, uma exceção será lançada com as informações a seguir:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

A exceção lançada, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, é uma subclasse da classe a seguir e implementa a interface `com.ibm.msg.client.jms.JmsExceptionDetail`.

- JM 3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Exceções Vinculadas

Uma exceção vinculada fornece informações adicionais sobre um problema de tempo de execução. Portanto, para cada `JMSException` que é lançado, um aplicativo deve verificar a exceção vinculada.

A própria exceção vinculada pode ter outra exceção vinculada e, por isso, as exceções vinculadas formam uma cadeia que leva de volta para o problema subjacente original. Uma exceção vinculada é implementada usando o mecanismo de exceção encadeada da classe `java.lang.Throwable`, e seu

aplicativo pode obter uma exceção vinculada chamando o método `Throwable.getCause()`. Para um `JMSException`, o método `getLinkedException()` delega para o método `Throwable.getCause()`.

Por exemplo, se um aplicativo especificar um número de porta incorreto ao se conectar a um gerenciador de filas, as exceções formarão a seguinte cadeia:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Geralmente, cada exceção em uma cadeia é lançada a partir de uma camada diferente no código. Por exemplo, as exceções na cadeia anteriormente são lançadas pelas seguintes camadas:

- A primeira exceção, uma instância de uma subclasse de `JMSException`, é lançada pela camada comum em IBM MQ classes for JMS.
- A próxima exceção, uma instância de `com.ibm.mq.MQException`, é lançada pelo provedor de mensagens IBM MQ.
- As próximas duas exceções, ambas as quais são instâncias de `com.ibm.mq.jmqi.JmqiException`, são lançadas pela Java Message Queueing Interface (JMQI). A JMQI é o componente que é usado pelo IBM MQ classes for JMS para se comunicar com um gerenciador de filas.
- A exceção final, uma instância de `java.net.ConnectionException`, é lançada pela biblioteca de classes Java.

Para obter mais informações sobre a arquitetura em camadas de IBM MQ classes for JMS, consulte as classes do IBM MQ para arquitetura JMS.

É possível codificar seu aplicativo para iterar por meio dessa cadeia para extrair todas as informações apropriadas, conforme mostrado no exemplo a seguir:

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
```

```
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Observe que seu aplicativo deve sempre verificar o tipo de cada exceção em uma cadeia porque o tipo de exceção pode variar e exceções de tipos diferentes encapsulam informações diferentes.

Obtendo informações específicas do IBM MQ sobre um problema

As instâncias de `com.ibm.mq.MQException` e `com.ibm.mq.jmqi.JmqiException` encapsulam informações específicas de IBM MQ sobre um problema.

Um `MQException` encapsula as seguintes informações:

- Um código de conclusão, que o seu aplicativo pode obter chamando o método `getCompCode()`.
- Um código de razão, que o seu aplicativo pode obter chamando o método `getReason()`.

Para obter exemplos de como usar esses métodos, consulte o código de amostra em exceções vinculadas.

Um `JmqiException` também encapsula um código de conclusão e um código de razão. Além disso, um `JmqiException` contém as informações em uma mensagem AMQ *nnnn* ou CSQ *nnnn*, se uma estiver

associada à exceção. Seu aplicativo pode obter os vários componentes desta mensagem chamando os métodos a seguir:

- O método `getWmqMsgExplanation()` retorna a explicação da mensagem AMQ *nnnn* ou CSQ *nnnn*.
- O método `getWmqMsgSeverity()` retorna a gravidade da mensagem AMQ *nnnn* ou CSQ *nnnn*.
- O método `getWmqMsgSummary()` retorna o resumo da mensagem AMQ *nnnn* ou CSQ *nnnn*.
- O método `getWmqMsgUserResponse()` retorna a resposta do usuário que está associada à mensagem AMQ *nnnn* ou CSQ *nnnn*.

Manipulando exceções não verificadas
Os métodos nas interfaces que são definidas em JMS 2.0 lançam exceções não verificadas. A classe base para essas exceções é `JMSRuntimeException`. Portanto, a captura de `JMSRuntimeExceptions` fornece uma maneira genérica de lidar com esses tipos de exceções.

Cada `JMSRuntimeException` encapsula as seguintes informações:

- Uma mensagem de exceção específica do provedor, a qual o seu aplicativo pode obter chamando o método `JMSRuntimeException.getMessage()`.
- Um código de erro específico do provedor, o qual o seu aplicativo pode obter chamando o método `JMSRuntimeException.getErrorCode()`.
- Uma exceção vinculada. Uma exceção que é lançada por uma chamada da API de JMS 2.0 é frequentemente o resultado de um problema de nível inferior que é relatado por outra exceção que está vinculada a essa exceção. Seu aplicativo pode obter uma exceção vinculada chamando o método `JMSRuntimeException.getCause()`.

Ao chamar métodos nas interfaces que são fornecidas pela API JMS 2.0 , a maioria das exceções lançadas pelo IBM MQ classes for JMS são instâncias de subclasses de `JMSRuntimeException`. Essas subclasses implementam a interface `com.ibm.msg.client.jms.JmsExceptionDetail`, que fornece as informações adicionais a seguir:

- Uma explicação da mensagem de exceção. O seu aplicativo pode obter essa mensagem chamando o método `JmsExceptionDetail.getExplanation()`.
- Uma resposta do usuário recomendada à exceção. O seu aplicativo pode obter essa mensagem chamando o método `JmsExceptionDetail.getUserAction()`.
- As chaves para as inserções de mensagem na mensagem de exceção. O seu aplicativo pode obter um iterador para todas as chaves chamando o método `JmsExceptionDetail.getKeys()`.
- As inserções de mensagem na mensagem de exceção. Por exemplo, uma inserção de mensagem pode ser o nome da fila que causou a exceção e pode ser útil para o seu aplicativo acessar esse nome. O seu aplicativo pode obter a inserção da mensagem correspondente em uma chave especificada chamando o método `JmsExceptionDetail.getValue()`.

Todos os métodos na interface `JmsExceptionDetail` retornarão nulo se nenhum detalhe estiver disponível.

Por exemplo, se um aplicativo tentar criar um `JMSProducer` para uma fila IBM MQ que não existe, uma exceção será lançada com as seguintes informações:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

A exceção lançada, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, é uma subclasse da classe a seguir e implementa a interface `com.ibm.msg.client.jms.JmsExceptionDetail` .

- JM 3.0 `jakarta.jms.InvalidDestinationException`

- JMS 2.0 `javax.jms.InvalidDestinationException`

Exceções encadeadas

Geralmente, as exceções são causadas por outras exceções. Portanto, para cada `JMSRuntimeException` que é lançado, seu aplicativo deve verificar a exceção vinculada.

A causa do `JMSRuntimeException` pode ser outra exceção. Essas exceções formam uma cadeia que leva de volta para o problema subjacente original. A causa de uma exceção ser implementada usando o mecanismo de exceção encadeada da classe `java.lang.Throwable`, e seu aplicativo pode obter uma exceção vinculada chamando o método `Throwable.getCause()`.

Por exemplo, se um aplicativo especificar um número de porta incorreto ao se conectar a um gerenciador de filas, as exceções formarão a seguinte cadeia:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Geralmente, cada exceção em uma cadeia é lançada a partir de uma camada diferente no código. Por exemplo, as exceções na cadeia anteriormente são lançadas pelas seguintes camadas:

- A primeira exceção, uma instância de uma subclasse de `JMSRuntimeException`, é lançada pela camada comum em IBM MQ classes for JMS.
- A próxima exceção, uma instância de `com.ibm.mq.MQException`, é lançada pelo provedor de mensagens IBM MQ.
- As próximas duas exceções, ambas as quais são instâncias de `com.ibm.mq.jmqi.JmqiException`, são lançadas pela Java Message Queueing Interface (JMQI). A JMQI é o componente que é usado pelo IBM MQ classes for JMS para se comunicar com um gerenciador de filas.
- A exceção final, uma instância de `java.net.ConnectionException`, é lançada pela biblioteca de classes Java.

Para obter mais informações sobre a arquitetura em camadas de IBM MQ classes for JMS, consulte as classes do IBM MQ para arquitetura JMS.

É possível codificar seu aplicativo para iterar por meio dessa cadeia para extrair todas as informações apropriadas, conforme mostrado no exemplo a seguir:

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
```

```
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Observe que seu aplicativo deve sempre verificar o tipo de cada exceção em uma cadeia porque o tipo de exceção pode variar e exceções de tipos diferentes encapsulam informações diferentes.

Obtendo informações específicas do IBM MQ sobre um problema

As instâncias de `com.ibm.mq.MQException` e `com.ibm.mq.jmqi.JmqiException` encapsulam informações específicas de IBM MQ sobre um problema.

Um `MQException` encapsula as seguintes informações:

- Um código de conclusão, que o seu aplicativo pode obter chamando o método `getCompCode()`.
- Um código de razão, que o seu aplicativo pode obter chamando o método `getReason()`.

Para obter exemplos de como usar esses métodos, consulte o código de amostra em exceções encadeadas.

Um `JmqiException` também encapsula um código de conclusão e um código de razão. Além disso, um `JmqiException` contém as informações em uma mensagem AMQ *nnnn* ou CSQ *nnnn*, se uma estiver associada à exceção. Seu aplicativo pode obter os vários componentes desta mensagem chamando os métodos a seguir:

- O método `getWmqMsgExplanation()` retorna a explicação da mensagem AMQ *nnnn* ou CSQ *nnnn*.
- O método `getWmqMsgSeverity()` retorna a gravidade da mensagem AMQ *nnnn* ou CSQ *nnnn*.
- O método `getWmqMsgSummary()` retorna o resumo da mensagem AMQ *nnnn* ou CSQ *nnnn*.
- O método `getWmqMsgUserResponse()` retorna a resposta do usuário que está associada à mensagem AMQ *nnnn* ou CSQ *nnnn*.

ExceptionListeners
Os objetos JMS `Connection` e `JMSContext` possuem uma conexão associada a um gerenciador de filas. Seu aplicativo pode registrar um `ExceptionListener` com um JMS `Connection` ou `JMSContext`. Se ocorrer um problema que torne a conexão que está associada ao `Connection` ou `JMSContext` inutilizável, o IBM MQ classes for JMS entregará uma exceção para o `ExceptionListener` chamando seu método `onException()`. Em seguida, seu aplicativo tem a oportunidade de restabelecer a conexão.

IBM MQ classes for JMS também pode entregar uma exceção ao listener de exceção, se ocorrer um problema durante a tentativa de entregar uma mensagem de maneira assíncrona.

Listeners de Exceção

Em IBM MQ 8.0.0 Fix Pack 2, para manter o comportamento para aplicativos JMS atuais que configuram um JMS `MessageListener` e um JMS `ExceptionListener` e para assegurar que os IBM MQ classes for JMS sejam consistentes com a especificação de JMS, o valor padrão para a propriedade `ConnectionFactory` ASYNCEXCEPTION será alterado para SYNC_EXCEPTIONS_CONNECTIONBROKEN. Como resultado, apenas as exceções que correspondem aos códigos de erro de conexão interrompidos são entregues ao `ExceptionListener` de um aplicativo.

O APAR IT14820, incluído do IBM MQ 9.0.0 Fix Pack 1, atualiza o IBM MQ classes for JMS para que:

- Um `ExceptionListener` que é registrado por um aplicativo é chamado para quaisquer exceções de conexão interrompidas, independentemente se o aplicativo está usando consumidores de mensagens síncronas ou assíncronas.
- As exceções de não conexão interrompidas (por exemplo MQRC_GET_INHIBITED) que surgem durante a entrega de mensagens são entregues ao `ExceptionListener` de um aplicativo, quando o aplicativo está usando consumidores de mensagens assíncronas e o JMS ConnectionFactory que é usado pelo aplicativo tem o conjunto de propriedade ASYNC_EXCEPTION configurado para o valor ASYNC_EXCEPTIONS_ALL.

Nota: Um `ExceptionListener` é chamado apenas uma vez para uma exceção de conexão interrompida, mesmo que duas conexões TCP/IP (uma usada pela Conexão JMS e uma usada por uma Sessão JMS) sejam interrompidas.

Para qualquer outro tipo de problema, uma exceção é lançada pela chamada da API do JMS atual. O tipo de exceção que é lançado depende da versão da API do JMS que o aplicativo está usando:

- Se o aplicativo estiver usando as interfaces que são fornecidas pela especificação do JMS 1.1, a exceção será um `JMSException`. Para obter mais informações sobre como manipular essas exceções, consulte “Manipulando exceções verificadas” na página 242.

- Se o aplicativo estiver usando as interfaces do JMS 2.0, a exceção será um `JMSRuntimeException`. Para obter mais informações sobre como manipular essas exceções, consulte “Manipulando exceções não verificadas” na página 245.

Se um aplicativo não registrar um listener de exceção com um `Connection` ou `JMSContext`, quaisquer exceções que forem entregues ao listener de exceção serão gravadas no log IBM MQ classes for JMS.

Acessando recursos do IBM MQ a partir de um aplicativo IBM MQ classes for JMS

IBM MQ classes for JMS fornece recursos para explorar um número de recursos do IBM MQ.

Atenção: Esses recursos estão fora da especificação de JMS ou, em determinados casos, violam a especificação de JMS. Se você usá-los, seu aplicativo será improvavelmente compatível com outros provedores JMS. Esses recursos que não estão em conformidade com a especificação de JMS são rotulados com um aviso de Atenção.

Lendo e gravando o descritor de mensagens a partir de um aplicativo IBM MQ classes for JMS

Controle a capacidade de acessar o descritor de mensagens (MQMD) configurando as propriedades em um Destino e uma Mensagem.

Alguns aplicativos IBM MQ requerem valores específicos para serem configurados no MQMD de mensagens enviadas a eles. IBM MQ classes for JMS fornece atributos de mensagens que permitem que os aplicativos JMS configurem os campos MQMD e, portanto, permitam que os aplicativos JMS "conduzem" os aplicativos IBM MQ.

Deve-se configurar a propriedade do objeto Destination WMQ_MQMD_WRITE_ENABLED para true para que a configuração de propriedades do MQMD tenham qualquer efeito. Então é possível usar os métodos de configuração de propriedades da mensagem (por exemplo, setStringProperty) para designar valores para os campos MQMD. Todos os campos MQMD são expostos, exceto StrucId e Version; BackoutCount pode ser lido, mas não gravado.

Este exemplo resulta em uma mensagem sendo colocada em uma fila ou em um tópico com MQMD.UserIdentifier configurado como "JoeBloggs".

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

É necessário configurar WMQ_MQMD_MESSAGE_CONTEXT antes de configurar JMS_IBM_MQMD_UserIdentifier. Para obter mais informações sobre o uso do WMQ_MQMD_MESSAGE_CONTEXT, consulte “Propriedades de objeto de mensagem do JMS” na página 252.

De forma semelhante, será possível extrair o conteúdo dos campos MQMD configurando WMQ_MQMD_READ_ENABLED como true antes de receber uma mensagem e, em seguida, usar os métodos get da mensagem, como getStringProperty. As propriedades recebidas são somente leitura.

Este exemplo resulta no campo *value* que mantém o valor do campo MQMD.ApplIdentityData de uma mensagem obtida a partir de uma fila ou de um tópico.

```
// Create a ConnectionFactory, connection, session, consumer
// ...
```

```
// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

Propriedades do objeto de destino doJMS
Duas propriedades do objeto Destination controlam acesso ao MQMD a partir do JMS e uma terceira controla o contexto de mensagem.

Tabela 36. Nomes e descrições das propriedades

Propriedade	**Formato curto**	**Descrição**
WMQ_MQMD_WRITE_ENABLED	MDW	Se um aplicativo JMS pode configurar os valores dos campos MQMD
WMQ_MQMD_READ_ENABLED	MDR	Se um aplicativo JMS pode extrair os valores dos campos MQMD
WMQ_MQMD_MESSAGE _CONTEXT	MDCTX	Qual nível de contexto da mensagem deve ser configurado pelo aplicativo JMS. O aplicativo deve estar em execução com autoridade de contexto apropriada para que essa propriedade entre em vigor

Tabela 37. Nomes de propriedades, valores e métodos configurados

Propriedade	**Valores válidos na ferramenta de administração (padrões em negrito)**	**Valores válidos em programas**	**Método configurado**
WMQ_MQMD_WRITE _ENABLED	• **Não** Todas as propriedades JMS_IBM_MQMD* são ignoradas e seus valores não são copiados para a estrutura MQMD subjacente. • SIM As propriedades JMS_IBM_MQMD* são processadas. Seus valores serão copiados para a estrutura do MQMD subjacente.	• **falso** • True	setMQMDWriteEnabled

Tabela 37. Nomes de propriedades, valores e métodos configurados (continuação)

Propriedade	Valores válidos na ferramenta de administração (padrões em negrito)	Valores válidos em programas	Método configurado
WMQ_MQMD_READ _ENABLED	• **Não** Ao enviar mensagens, as propriedades JMS_IBM_MQMD* em uma mensagem enviada não são atualizadas para refletir os valores de campos atualizados no MQMD. Ao receber mensagens, nenhuma das propriedades JMS_IBM_MQMD* estarão disponíveis em uma mensagem recebida, mesmo que o emissor tenha configurado algumas ou todas elas. • SIM Ao enviar mensagens, todas as propriedades JMS_IBM_MQMD* em uma mensagem enviada são atualizadas para refletir os valores de campos atualizados no MQMD, incluindo aquelas que o emissor não configurou explicitamente. Ao receber mensagens, todas as propriedades JMS_IBM_MQMD* estão disponíveis em uma mensagem recebida, incluindo aquelas que o emissor não configurou explicitamente.	• **falso** • True	setMQMDReadEnabled
WMQ_MQMD _MESSAGE_CONTEXT	• **DEFAULT** A chamada da API MQOPEN e a estrutura de MQPMO não especificam nenhuma opção de contexto de mensagem explícita • SET_IDENTITY_CONTEXT A chamada da API MQOPEN especifica a opção de contexto de mensagem MQOO_SET_IDENTITY_CONTEXT e a estrutura de MQPMO especifica MQPMO_SET_IDENTITY_CONTEXT • SET_ALL_CONTEXT A chamada da API MQOPEN especifica a opção de contexto de mensagem MQOO_SET_ALL_CONTEXT e a estrutura de MQPMO especifica MQPMO_SET_ALL_CONTEXT	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

Propriedades de objeto de mensagem do JMS
Propriedades do objeto de mensagem prefixadas com JMS_IBM_MQMD permitem definir ou ler o campo MQMD correspondente.

Enviando mensagens

Todos os campos MQMD, exceto StrucId e Version, são representados. Essas propriedades referem-se apenas aos campos MQMD; quando uma propriedade ocorre tanto no MQMD quanto no cabeçalho MQRFH2, a versão no MQRFH2 não é configurada nem extraída.

Qualquer uma dessas propriedades pode ser configurada, exceto JMS_IBM_MQMD_BackoutCount. Qualquer valor configurado para JMS_IBM_MQMD_BackoutCount é ignorado.

Se uma propriedade tiver um comprimento máximo e você fornecer um valor que é muito longo, o valor será truncado.

Para determinadas propriedades, também deve-se definir a propriedade WMQ_MQMD_MESSAGE_CONTEXT no objeto de Destino. O aplicativo deve estar em execução com autoridade de contexto apropriado para esta propriedade entrar em vigor. Se você não configurar WMQ_MQMD_MESSAGE_CONTEXT para um valor apropriado, o valor da propriedade será ignorado. Se você configura WMQ_MQMD_MESSAGE_CONTEXT para um valor apropriado, mas você não tem autoridade de contexto suficiente para o gerenciador de filas, uma JMSException é emitida. Propriedades que requerem valores específicos de WMQ_MQMD_MESSAGE_CONTEXT são conforme a seguir.

As propriedades a seguir requerem que WMQ_MQMD_MESSAGE_CONTEXT seja configurado para WMQ_MDCTX_SET_IDENTITY_CONTEXT ou WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

As propriedades a seguir requerem WMQ_MQMD_MESSAGE_CONTEXT seja configurado para WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_PutApplType
- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOriginData

Como receber mensagens

Todas estas propriedades estão disponíveis em uma mensagem recebida se a propriedade WMQ_MQMD_READ_ENABLED estiver configurada como true, não importa quais propriedades reais estão definidas para o aplicativo de produção. Um aplicativo não pode modificar as propriedades de uma mensagem recebida, a menos que todas as propriedades sejam limpas primeiro, de acordo com a especificação de JMS. A mensagem recebida pode ser transmitida sem modificar as propriedades.

Atenção: Se o seu aplicativo recebe uma mensagem de um destino com a propriedade WMQ_MQMD_READ_ENABLED definida como true, e a encaminha para um destino com WMQ_MQMD_WRITE_ENABLED configurado como true, isto resulta em todos os valores de campo MQMD da mensagem recebida copiados na mensagem encaminhada.

Tabela de propriedades

Esta tabela lista as propriedades do objeto de mensagem que representa os campos MQMD. Consulte os links para obter descrições completas dos campos e seus valores permitidos.

Tabela 38. Propriedade nomes, descrições e tipos

Propriedade	Descrição	Tipo de Java	Link para descrição completa
JMS_IBM_MQMD_Report	Opções para as mensagens de relatório	Integer	Relatório
JMS_IBM_MQMD_MsgType	Tipo de mensagem	Integer	MsgType
JMS_IBM_MQMD_Expiry	Tempo de vida da mensagem	Integer	Expiração
JMS_IBM_MQMD_Feedback	Feedback ou código de razão	Integer	Feedback
JMS_IBM_MQMD_Encoding	Codificação numérica de dados da mensagem	Integer	Codificação
JMS_IBM_MQMD_CodedCharSetId	Identificador do conjunto de caracteres de dados da mensagem	Integer	CodedCharSetId
JMS_IBM_MQMD_Format	Nome do formato dos dados da mensagem	Sequência	Formato
JMS_IBM_MQMD_Priority [1]	Prioridade da mensagem	Integer	Prioridade
JMS_IBM_MQMD_Persistence	Persistência de mensagem	Integer	Persistência
JMS_IBM_MQMD_MsgId [2]	ID da Mensagem	Objeto (byte[]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	Identificador de correlação	Objeto (byte[]) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	contador de backout	Integer	BackoutCount
JMS_IBM_MQMD_ReplyToQ	Nome da fila de resposta	Sequência	ReplyToQ
JMS_IBM_MQMD_ReplyToQMgr	Nome do gerenciador de filas de resposta	Sequência	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	Identificador de usuários	Sequência	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Símbolo de contabilidade	Objeto (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	dados do aplicativo relacionados à identidade	Sequência	ApplIdentityData
JMS_IBM_MQMD_PutApplType	Tipo de aplicativo que coloca a mensagem	Integer	PutApplType
JMS_IBM_MQMD_PutApplName	Nome do aplicativo que coloca a mensagem	Sequência	PutApplName
JMS_IBM_MQMD_PutDate	Data quando a mensagem foi colocada	Sequência	PutDate
JMS_IBM_MQMD_PutTime	Hora quando a mensagem foi colocada	Sequência	PutTime
JMS_IBM_MQMD_ApplOriginData	Os dados do aplicativo relacionados à origem	Sequência	ApplOriginData

Tabela 38. Propriedade nomes, descrições e tipos (continuação)			
Propriedade	**Descrição**	**Tipo de Java**	**Link para descrição completa**
JMS_IBM_MQMD_GroupId	Identificador de grupo	Objeto (byte[]) [4]	GroupId
JMS_IBM_MQMD_MsgSeqNumber	Número de sequência de mensagem lógica dentro do grupo	Integer	MsgSeqNumber
JMS_IBM_MQMD_Offset	Deslocamento dos dados na mensagem física a partir do início da mensagem lógica	Integer	offset
JMS_IBM_MQMD_MsgFlags	Sinalizadores de mensagem	Integer	MsgFlags
JMS_IBM_MQMD_OriginalLength	Comprimento da mensagem original	Integer	OriginalLength

1. **Atenção:** Se você designa um valor para JMS_IBM_MQMD_Priority que não está dentro do intervalo 0-9, isso viola a especificação do JMS.
2. **Atenção:** A especificação JMS indica que o ID de mensagem deve ser configurado pelo provedor JMS e que ele deve ser exclusivo ou nulo. Se você designar um valor para JMS_IBM_MQMD_MsgId, esse valor será copiado para o JMSMessageID. Portanto, não é configurado pelo provedor JMS e pode não ser exclusivo: isso viola a especificação de JMS.
3. **Atenção:** Se você designa um valor para JMS_IBM_MQMD_CorrelId que inicia com a sequência 'ID:', isso viola a especificação do JMS.
4. **Atenção:** O uso de propriedades de matriz de bytes em uma mensagem viola a especificação de JMS.

Acessando dados de mensagens do IBM MQ a partir de um aplicativo usando o IBM MQ classes for JMS
É possível acessar os dados concluídos de mensagens IBM MQ de dentro de um aplicativo usando IBM MQ classes for JMS. Para acessar todos os dados, a mensagem deve ser uma `JMSBytesMessage`. O corpo do `JMSBytesMessage` inclui qualquer cabeçalho `MQRFH2`, quaisquer outros cabeçalhos do IBM MQ e os dados de mensagens a seguir.

Configure a propriedade `WMQ_MESSAGE_BODY` do destino como `WMQ_MESSAGE_BODY_MQ` para receber todos os dados do corpo da mensagem no `JMSBytesMessage`.

Se `WMQ_MESSAGE_BODY` está configurado para `WMQ_MESSAGE_BODY_JMS` ou `WMQ_MESSAGE_BODY_UNSPECIFIED`, o corpo da mensagem é retornado sem o cabeçalho JMS `MQRFH2` e as propriedades `JMSBytesMessage` refletem o conjunto de propriedades no `RFH2`.

Alguns aplicativos não podem usar as funções descritas neste tópico. Se um aplicativo for conectado a um gerenciador de filas do IBM MQ V6 ou se ele configurou `PROVIDERVERSION` como 6, as funções não estarão disponíveis.

Enviando uma mensagem

Ao enviar mensagens a propriedade de destino, `WMQ_MESSAGE_BODY`, terá precedência sobre `WMQ_TARGET_CLIENT`.

Se `WMQ_MESSAGE_BODY` está configurado para `WMQ_MESSAGE_BODY_JMS`, IBM MQ classes for JMS gera automaticamente um cabeçalho `MQRFH2` baseado nas configurações das propriedades `JMSMessage` e campos de cabeçalho.

Se WMQ_MESSAGE_BODY for configurado como WMQ_MESSAGE_BODY_MQ, nenhum cabeçalho adicional será incluído no corpo da mensagem

Se WMQ_MESSAGE_BODY for configurado como WMQ_MESSAGE_BODY_UNSPECIFIED, IBM MQ classes for JMS enviará um cabeçalho MQRFH2, a menos que WMQ_TARGET_CLIENT seja configurado como WMQ_TARGET_DEST_MQ. No recebimento, configurando WMQ_TARGET_CLIENT como WMQ_TARGET_DEST_MQ resulta em nenhum MQRFH2 que está sendo removido do corpo da mensagem.

Nota: JMSBytesMessage e JMSTextMessage não requerem um MQRFH2, enquanto que o JMSStreamMessage, JMSMapMessage e JMSObjectMessage requerem.

WMQ_MESSAGE_BODY_UNSPECIFIED é a configuração padrão para WMQ_MESSAGE_BODY e WMQ_TARGET_DEST_JMS é a configuração padrão para WMQ_TARGET_CLIENT.

Se você enviar uma JMSBytesMessage, será possível substituir as configurações padrão para o corpo da mensagem do JMS quando mensagem do IBM MQ for construída. Use as seguintes propriedades:

- JMS_IBM_Format ou JMS_IBM_MQMD_Format: essa propriedade especificará o formato do cabeçalho ou carga útil do aplicativo IBM MQ que inicia o corpo da mensagem do JMS se não houver nenhum cabeçalho do WebSphere MQ precedente.
- JMS_IBM_Character_Set ou JMS_IBM_MQMD_CodedCharSetId: essa propriedade especificará o CCSID do cabeçalho ou carga útil do aplicativo IBM MQ que inicia o corpo da mensagem do JMS se não houver nenhum cabeçalho do WebSphere MQ precedente.
- JMS_IBM_Encoding ou JMS_IBM_MQMD_Encoding: essa propriedade especificará a codificação do cabeçalho ou carga útil do aplicativo IBM MQ que inicia o corpo da mensagem do JMS se não houver nenhum cabeçalho do WebSphere MQ precedente.

Se ambos os tipos de propriedade estão especificados, as propriedades JMS_IBM_MQMD_* substituem as propriedades JMS_IBM_* correspondentes, contanto que a propriedade de destino WMQ_MQMD_WRITE_ENABLED seja configurada como true.

As diferenças em vigor entre as propriedades de mensagem de configuração usando JMS_IBM_MQMD_* e JMS_IBM_* são significativas:

1. As propriedades JMS_IBM_MQMD_* são específicas para o provedor do IBM MQ JMS.
2. As propriedades JMS_IBM_MQMD_* são configuradas apenas no MQMD. As propriedades JMS_IBM_* são configuradas no MQMD apenas se a mensagem não tem um cabeçalho MQRFH2 JMS. Caso contrário, elas são configuradas no cabeçalho do JMS RFH2.
3. As propriedades JMS_IBM_MQMD_* não tem efeito na codificação de texto e nos números gravados em um JMSMessage.

   Um aplicativo de recebimento que presume provavelmente os valores de MQMD.Encoding e MQMD.CodedCharSetId corresponde ao conjunto de codificação e caracteres de números e textos no corpo da mensagem. Se as propriedades JMS_IBM_MQMD_* forem usadas, será responsabilidade do aplicativo de envio fazer isso. A codificação e o conjunto de caracteres de números e texto no corpo da mensagem são configurados pelas propriedades JMS_IBM_*.

   O fragmento codificado de forma inválida no Figura 39 na página 256 envia uma mensagem codificada no conjunto de caracteres 1208, com MQMD.CodedCharSetId configurado para 37.

a. Enviar mensagem codificada de forma errada

```
TextMessage tmo = session.createTextMessage();
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQDestination)destination).setMQMDWriteEnabled(true);
tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
tmo.setText("String one");
producer.send(tmo);
```

b. Recebendo a mensagem, contando com o valor de JMS_IBM_CHARACTER_SET configurado pelo valor de MQMD.CodedCharSetId:

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

c. Saída resultante:

```
Message is "éÈÊ'>...??>?"
```

Figura 39. MQMD e dados de mensagens inconsistentemente codificados

Um dos snippets de código em Figura 40 na página 256 resulta em uma mensagem sendo colocada em uma fila ou em um tópico com seu corpo que contém a carga útil do aplicativo sem um cabeçalho MQRFH2 gerado automaticamente que está sendo incluído.

1. Configurando WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Configurando WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

Figura 40. Envie uma mensagem com um corpo da mensagem MQ.

Recebendo uma mensagem

Se WMQ_MESSAGE_BODY estiver configurado como WMQ_MESSAGE_BODY_JMS, o tipo e o corpo de mensagem de entrada do JMS serão determinados pelos conteúdos da mensagem do WebSphere MQ recebida. O tipo e o corpo de mensagem serão determinados pelos campos no cabeçalho MQRFH2 ou no MQMD, se não houver nenhum MQRFH2.

Se WMQ_MESSAGE_BODY estiver configurado como WMQ_MESSAGE_BODY_MQ, o tipo de mensagem de entrada do JMS será JMSBytesMessage. O corpo da mensagem JMS são os dados da mensagem retornados pela chamada de API subjacente MQGET. O comprimento do corpo da mensagem é o comprimento retornado pela chamada MQGET. O conjunto de caracteres e a codificação dos dados no corpo da mensagem é determinado pelos campos CodedCharSetId e Encoding do MQMD. O formato dos dados no corpo da mensagem é determinado pelo campo Format do MQMD

Se WMQ_MESSAGE_BODY estiver configurado como WMQ_MESSAGE_BODY_UNSPECIFIED, o valor padrão, o IBM MQ classes for JMS, configurará o WMQ_MESSAGE_BODY_JMS.

Ao receber um JMSBytesMessage, será possível decodificá-lo por referência às seguintes propriedades:

- JMS_IBM_Format ou JMS_IBM_MQMD_Format: essa propriedade especificará o formato do cabeçalho ou carga útil do aplicativo IBM MQ que inicia o corpo da mensagem do JMS se não houver nenhum cabeçalho do WebSphere MQ precedente.
- JMS_IBM_Character_Set ou JMS_IBM_MQMD_CodedCharSetId: essa propriedade especificará o CCSID do cabeçalho ou carga útil do aplicativo IBM MQ que inicia o corpo da mensagem do JMS se não houver nenhum cabeçalho do WebSphere MQ precedente.
- JMS_IBM_Encoding ou JMS_IBM_MQMD_Encoding: essa propriedade especificará a codificação do cabeçalho ou carga útil do aplicativo IBM MQ que inicia o corpo da mensagem do JMS se não houver nenhum cabeçalho do WebSphere MQ precedente.

O fragmento de código a seguir resulta em uma mensagem recebida que é um JMSBytesMessage. Independentemente do conteúdo da mensagem recebida e do campo de formato do MQMD recebido, a mensagem será uma JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Propriedade de destino WMQ_MESSAGE_BODY
WMQ_MESSAGE_BODY determina se um aplicativo JMS processa o MQRFH2 de uma mensagem do IBM MQ como parte da carga útil da mensagem (ou seja, como parte do corpo da mensagem do JMS).

Tabela 39. Nomes e descrições das propriedades

Propriedade	**Formato curto**	**Descrição**
WMQ_MESSAGE_BODY	MBODY	Se um aplicativo JMS processa o MQRFH2 de uma mensagem do IBM MQ como parte da carga útil da mensagem (ou seja, como parte do corpo da mensagem do JMS).

Tabela 40. Nomes de propriedades, valores e métodos configurados

Propriedade	Valores válidos na ferramenta de administração (padrões em negrito)	Valores válidos em programas	Método configurado
WMQ_MESSAGE_BODY	• **UNSPECIFIED** Ao enviar, o IBM MQ classes for JMS gera e inclui um cabeçalho MQRFH2, ou não, dependendo do valor de WMQ_TARGET_CLIENT. Ao receber, age como JMS do valor. • JMS Ao enviar, o IBM MQ classes for JMS gera automaticamente um cabeçalho MQRFH2 e o inclui na mensagem do IBM MQ. Ao receber, o IBM MQ classes for JMS configura as propriedades de mensagem do JMS de acordo com os valores no MQRFH2 (se presente); ele não apresenta o MQRFH2 como parte do corpo da mensagem do JMS. • MQ Ao enviar, o IBM MQ classes for JMS não gera um MQRFH2. Ao receber, o IBM MQ classes for JMS apresenta o MQRFH2 como parte do corpo da mensagem do JMS.	• **WMQ_MESSAGE_BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

Mensagens persistentes do JMS
Os aplicativos IBM MQ classes for JMS podem usar o atributo de fila **NonPersistentMessageClass** para fornecer melhor desempenho para mensagens persistentes do JMS às custas de alguma confiabilidade.

Uma fila do IBM MQ tem um atributo chamado **NonPersistentMessageClass**. O valor desse atributo determina se mensagens não persistentes na fila serão descartadas quando o gerenciador de filas for reiniciado.

É possível configurar o atributo para uma fila local usando o comando IBM MQ Script (MQSC), DEFINE QLOCAL, com um dos parâmetros a seguir:

NPMCLASS(NORMAL)
Mensagens não persistentes na fila são descartadas quando o gerenciador de filas for reiniciado. Esse é o valor-padrão.

NPMCLASS(HIGH)
Mensagens não persistentes na fila não são descartadas quando o gerenciador de filas é reiniciado após um encerramento em modo quiesce ou imediato. Mensagens não persistentes podem ser descartadas, no entanto, após um encerramento prioritário ou uma falha.

Este tópico descreve como aplicativos IBM MQ classes for JMS podem usar esse atributo de fila para fornecer melhor desempenho para mensagens persistentes do JMS.

A propriedade PERSISTENCE de um objeto Queue ou Topic pode ter o valor HIGH. É possível usar a ferramenta de administração do IBM MQ JMS para configurar esse valor ou um aplicativo pode chamar o método Destination.setPersistence() passando o valor WMQConstants.WMQ_PER_NPHIGH como um parâmetro.

Se um aplicativo envia uma mensagem persistente do JMS ou uma mensagem não persistente do JMS para um destino no qual a propriedade PERSISTENCE tem o valor HIGH e a fila subjacente do IBM MQ for configurada para NPMCLASS(HIGH), a mensagem será colocada na fila como uma mensagem não persistente do IBM MQ. Se a propriedade PERSISTENCE do destino não tiver o valor HIGH ou se a fila subjacente for configurada para NPMCLASS(NORMAL), uma mensagem persistente do JMS será colocada na fila como uma mensagem persistente do IBM MQ e uma mensagem não persistente do JMS será colocada na fila como uma mensagem não persistente do IBM MQ.

Se uma mensagem persistente do JMS for colocada em uma fila como uma mensagem não persistente do IBM MQ e você deseja assegurar que a mensagem não seja descartada após um encerramento em modo quiesce ou imediato de um gerenciador de filas, todas as filas pelas quais a mensagem pode ser roteada deverão ser configuradas para NPMCLASS(HIGH). No domínio de publicar/assinar, essas filas incluem filas de assinantes. Como um auxílio para aplicar essa configuração, o IBM MQ classes for JMS emitirá uma InvalidDestinationException se um aplicativo tentar criar um consumidor de mensagem para um destino no qual a propriedade PERSISTENCE tem o valor HIGH e a fila subjacente do IBM MQ está configurada para NPMCLASS(NORMAL).

Configurar a propriedade PERSISTENCE de um destino para HIGH não afeta como uma mensagem é recebida desse destino. Uma mensagem enviada como uma mensagem persistente do JMS é recebida como uma mensagem persistente do JMS e uma mensagem enviada como uma mensagem não persistente do JMS é recebida como uma mensagem não persistente do JMS.

Quando um aplicativo envia a primeira mensagem para um destino no qual a propriedade PERSISTENCE tem o valor HIGH ou quando um aplicativo cria o primeiro consumidor de mensagens para um destino no qual a propriedade PERSISTENCE tem o valor HIGH, o IBM MQ classes for JMS emite uma chamada MQINQ para determinar se NPMCLASS(HIGH) está configurado na fila subjacente do IBM MQ. O aplicativo deve, portanto, ter a autoridade para consultar na fila. Além disso, o IBM MQ classes for JMS preserva o resultado da chamada MQINQ até que o destino seja excluído e não emite mais chamadas MQINQ. Portanto, se você mudar a configuração NPMCLASS na fila subjacente enquanto o aplicativo ainda estiver usando o destino, o IBM MQ classes for JMS não observa a nova configuração.

Permitindo que mensagens persistentes do JMS sejam colocadas em filas do IBM MQ como mensagens não persistentes do IBM MQ, você está ganhando desempenho ao custo de alguma confiabilidade. Se você precisar de confiabilidade máxima para mensagens persistentes do JMS, não envie as mensagens a um destino no qual a propriedade PERSISTENCE tem o valor HIGH.

A JMS Camada pode usar SYSTEM.JMS.TEMPQ.MODEL, em vez de SYSTEM.DEFAULT.MODEL.QUEUE. SYSTEM.JMS.TEMPQ.MODEL cria filas dinâmicas permanentes que aceitam mensagens persistentes, porque SYSTEM.DEFAULT.MODEL.QUEUE não pode aceitar mensagens persistentes. Deve-se, portanto, usar SYSTEM.JMS.TEMPQ.MODEL ou mudar a fila modelo para uma fila alternativa de sua escolha para usar filas provisórias para aceitar mensagens persistentes.

Usando TLS com o IBM MQ classes for JMS

Os aplicativos IBM MQ classes for JMS podem usar a criptografia de Segurança da Camada de Transporte (TLS). Para fazer isso, eles requerem um provedor JSSE.

As conexões do IBM MQ classes for JMS usando TRANSPORT(CLIENT) suportam a criptografia TLS. O TLS fornece criptografia de comunicação, autenticação e integridade da mensagem. É geralmente usada para comunicações seguras entre qualquer dois pontos na Internet ou em uma intranet.

O IBM MQ classes for JMS usa o Java Secure Socket Extension (JSSE) para manipular a criptografia TLS e, portanto, requer um provedor JSSE. JVMs JSE v1.4 têm um provedor JSSE integrado. Detalhes sobre como gerenciar e armazenar certificados podem variar de provedor para provedor. Para obter informações sobre isso, consulte a documentação do seu provedor JSSE.

Esta seção supõe que seu provedor JSSE esteja instalado e configurado corretamente e que os certificados adequados tenham sido instalados e disponibilizados para seu provedor JSSE. Agora é possível usar JMSAdmin para configurar várias propriedades administrativas.

Se o seu aplicativo IBM MQ classes for JMS usar uma tabela de definição de canal do cliente (CCDT) para conectar a um gerenciador de filas, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS” na página 289.

Propriedade de objeto SSLCIPHERSUITE
Configure SSLCIPHERSUITE para ativar a criptografia TLS em um objeto ConnectionFactory.

Para ativar a criptografia TLS em um objeto ConnectionFactory, use JMSAdmin para configurar a propriedade SSLCIPHERSUITE para um CipherSuite suportado pelo provedor JSSE. Deve corresponder ao CipherSpec configurado no canal de destino. No entanto, CipherSuites são distintos dos CipherSpecs e, portanto, têm nomes diferentes. O “CipherSpecs e CipherSuites TLS no IBM MQ classes for JMS” na página 263 contém uma tabela mapeando os CipherSpecs suportados pelo IBM MQ para seus CipherSuites equivalentes, conforme conhecidos pelo JSSE. Para obter mais informações sobre CipherSpecs e CipherSuites com o IBM MQ, consulte Protegendo IBM MQ.

Por exemplo, para configurar um objeto ConnectionFactory que pode ser usado para criar uma conexão por meio de um canal MQI ativado para TLS com um CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA, emita o seguinte comando para JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Também pode ser configurado a partir de um aplicativo, usando o método setSSLCipherSuite() em um objeto MQConnectionFactory.

Por conveniência, se um CipherSpec for especificado na propriedade SSLCIPHERSUITE, JMSAdmin tentará mapear o CipherSpec para um CipherSuite apropriado e emitirá um aviso. Essa tentativa de mapear não será feita se a propriedade for especificada por um aplicativo.

Como alternativa, use o Client Channel Definition Table (CCDT). Para obter informações adicionais, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS” na página 289.

Propriedade de objeto SSLFIPSREQUIRED
Se precisar de uma conexão para usar um CipherSuite que seja suportado pelo IBM Java provedor JSSE FIPS (IBMJSSEFIPS), configure a propriedade SSLFIPSREQUIRED do connection factory como YES.

Nota: No AIX, Linux, and Windows, IBM MQ fornece conformidade FIPS 140-2 por meio do módulo criptográfico IBM Crypto for C (ICC) . O certificado deste módulo foi movido para o status Histórico. Os clientes devem visualizar o IBM Crypto for C (ICC) certificado e estar ciente de qualquer aviso fornecido pelo NIST Um módulo FIPS 140-3 de substituição está atualmente em andamento e seu status pode ser visualizado procurando por ele na NIST CMVP modules in process list.

O IBM MQ Operator 3.2.0 e a imagem do contêiner do gerenciador de filas 9.4.0.0 em diante são baseados no UBI 9 A conformidade do FIPS 140-3 está pendente atualmente e seu status pode ser visualizado procurando "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" no NIST CMVP modules in process list.

O valor padrão desta propriedade é NO, o que significa que uma conexão pode usar qualquer CipherSuite que seja suportado pelo IBM MQ.

Se um aplicativo usar mais de uma conexão, o valor de SSLFIPSREQUIRED que é usado quando o aplicativo cria a primeira conexão determina o valor que é usado quando o aplicativo cria qualquer conexão subsequente. Isso significa que o valor da propriedade SSLFIPSREQUIRED do connection factory que é usado para criar uma conexão subsequente é ignorado. Deve-se reiniciar o aplicativo se você quiser usar um valor diferente de SSLFIPSREQUIRED.

Um aplicativo pode configurar esta propriedade chamando o método setSSLFipsRequired() de um objeto ConnectionFactory. A propriedade será ignorada se nenhum CipherSuite estiver configurado.

Tarefas relacionadas

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Referências relacionadas

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Propriedade de objeto SSLPEERNAME

Use SSLPEERNAME para especificar um padrão de nome distinto para assegurar que seu aplicativo JMS se conecte ao gerenciador de filas correto.

Um aplicativo JMS pode assegurar que se conecta ao gerenciador de filas correto especificando um padrão de nome distinto (DN). A conexão será bem-sucedida somente se o gerenciador de filas apresentar um DN que corresponda ao padrão. Para obter mais detalhes sobre o formato desse padrão, consulte os tópicos relacionados.

O DN é configurado usando a propriedade SSLPEERNAME de um objeto ConnectionFactory. Por exemplo, o comando JMSAdmin a seguir configura um objeto ConnectionFactory para esperar que o gerenciador de filas para se identificar com um Nome Comum que começa com os caracteres `QMGR.` e com pelo menos dois nomes de Unidades Organizacionais, o primeiro dos quais deve ser IBM e o segundo WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

A verificação não faz distinção entre maiúsculas e minúsculas e pontos-e-vírgulas podem ser usados no lugar de vírgulas. SSLPEERNAME também pode ser configurado a partir de um aplicativo usando o método setSSLPeerName() em um objeto MQConnectionFactory. Se essa propriedade não estiver configurada, nenhuma verificação será executada no Nome distinto fornecido pelo gerenciador de filas. Essa propriedade será ignorada se nenhum CipherSuite estiver configurado.

Propriedade de objeto SSLCERTSTORES

Utilize SSLCERTSTORES para especificar uma lista de servidores LDAP a serem usados para verificação da lista de revogação de certificado (CRL).

É comum usar a lista de revogação de certificado (CRL) para identificar os certificados que não são mais confiáveis. CRLs geralmente são hospedadas em servidores LDAP. O JMS permite que um servidor LDAP seja especificado para verificação de CRL sob Java 2 v1.4 ou posterior. O exemplo a seguir JMSAdmin direciona JMS a usar um CRL hospedado em um servidor LDAP chamado crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Nota: Para usar um CertStore com sucesso com uma CRL hospedada em um servidor LDAP, certifique-se de que o Java Software Development Kit (SDK) seja compatível com a CRL. Alguns SDKs requerem que a CRL esteja em conformidade com o RFC 2587, que define um esquema para o LDAP v2. A maioria dos servidores LDAP v3 usa RFC 2256 em vez disso.

Se seu servidor LDAP não estiver em execução na porta padrão 389, será possível especificar a porta anexando dois pontos (:) e o número da porta ao nome do host. Se o certificado apresentado pelo gerenciador de filas estiver presente na CRL hospedada em crl1.ibm.com, a conexão não será concluída. Para evitar um ponto único de falha, o JMS permite que vários servidores LDAP sejam fornecidos, apresentando uma lista de servidores LDAP delimitada pelo caractere de espaço. Aqui está um exemplo:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

Quando vários servidores LDAP são especificados, o JMS tenta cada um por vez até localizar um servidor com o qual pode verificar com sucesso o certificado do gerenciador de filas. Cada servidor deve conter informações idênticas.

Uma sequência nesse formato pode ser fornecida por um aplicativo no método MQConnectionFactory.setSSLCertStores(). Como alternativa, o aplicativo pode criar um ou mais objetos java.security.cert.CertStore, coloque-os em um objeto Collection apropriado e forneça esse objeto

Collection ao método setSSLCertStores(). Dessa maneira, o aplicativo pode customizar a verificação da CRL. Consulte a documentação de JSSE para obter detalhes sobre como construir e usar objetos CertStore.

O certificado apresentado pelo gerenciador de filas quando uma conexão que está sendo configurada for validada da seguinte forma:

1. O primeiro objeto CertStore no Collection identificado por sslCertStores é usado para identificar um servidor de CRL.
2. Uma tentativa é feita para contatar o servidor de CRL.
3. Se a tentativa for bem-sucedida, uma correspondência do certificado é procurada no servidor.
   a. Se o certificado tiver sido revogado, o processo de procura terminou e a solicitação de conexão falhará com o código de razão MQRC_SSL_CERTIFICATE_REVOKED.
   b. Se o certificado não for localizado, o processo de procura termina e a conexão tem permissão para continuar.
4. Se a tentativa de contatar o servidor não for bem-sucedida, o próximo objeto CertStore será usado para identificar um servidor CRL e o processo é repetido a partir da etapa 2.

   Se esse era o último CertStore no Collection ou se o Collection não contiver nenhum objeto CertStore, o processo de procura falhou e a solicitação de conexão falhará com o código de razão MQRC_SSL_CERT_STORE_ERROR.

O objeto Collection determina a ordem na qual CertStores são usados.

Se seu aplicativo usar setSSLCertStores() para configurar um Collection de objetos CertStore, o MQConnectionFactory não poderá mais ser ligado a namespace JNDI. A tentativa de fazer isso causa uma exceção. Se a propriedade sslCertStores não estiver configurada, nenhuma verificação de revogação será executada no certificado fornecido pelo gerenciador de filas. Essa propriedade será ignorada se nenhum CipherSuite estiver configurado.

Propriedade de objeto SSLRESETCOUNT
Essa propriedade representa o número total de bytes enviados e recebidos por uma conexão antes da chave secreta usada para criptografia ser renegociada.

O número de bytes enviados é o número antes da criptografia e o número de bytes recebidos é o número após a decriptografia. O número de bytes também inclui informações de controle enviadas e recebidas pelo IBM MQ classes for JMS.

Por exemplo, para configurar um objeto ConnectionFactory que pode ser usado para criar uma conexão por meio de um canal MQI ativado para TLS com uma chave secreta que seja renegociada após 4 MB de dados fluírem, emita o seguinte comando para JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Um aplicativo pode configurar essa propriedade chamando o método setSSLResetCount() de um objeto ConnectionFactory.

Se o valor dessa propriedade for zero, que é o valor padrão, a chave secreta nunca será renegociada. A propriedade será ignorada se nenhum CipherSuite estiver configurado.

Propriedade de objeto SSLSocketFactory
Para customizar outros aspectos da conexão TLS para um aplicativo, crie um SSLSocketFactory e configure o JMS para usá-lo.

Você pode desejar customizar outros aspectos da conexão TLS para um aplicativo. Por exemplo, você pode desejar inicializar o hardware criptográfico ou mudar o keystore e o armazenamento confiável em uso. Para fazer isso, o aplicativo deve primeiramente criar um objeto javax.net.ssl.SSLSocketFactory customizado de acordo. Consulte sua documentação do JSSE para obter informações sobre como fazer isso, porque os recursos customizáveis variam de provedor para provedor. Após um objeto

SSLSocketFactory adequado ser obtido, use o método MQConnectionFactory.setSSLSocketFactory() para configurar o JMS para usar o objeto SSLSocketFactory customizado.

Se seu aplicativo usar o método setSSLSocketFactory() para configurar um objeto SSLSocketFactory customizado, o objeto MQConnectionFactory não poderá mais ser ligado a um namespace JNDI. A tentativa de fazer isso causa uma exceção. Se esta propriedade não for configurada, o objeto SSLSocketFactory padrão será usado. Consulte a documentação do JSSE para obter detalhes sobre o comportamento do objeto padrão SSLSocketFactory. Essa propriedade será ignorada se nenhum CipherSuite estiver configurado.

Importante: Não presuma que o uso das propriedades SSL garantirá a segurança quando um objeto ConnectionFactory for recuperado a partir de namespace JNDI que não é seguro em si. Especificamente, a implementação de LDAP padrão do JNDI não é segura. Um invasor pode imitar o servidor LDAP, enganando um aplicativo JMS a conectar-se ao servidor errado sem perceber. Com o acordo de segurança adequada no lugar, outras implementações de JNDI (como a implementação fscontext) estão seguras.

Fazendo mudanças no keystore ou no armazenamento confiável de JSSE
Se você fizer mudanças no keystore ou no armazenamento confiável, deverá tomar determinadas ações para que as mudanças sejam captadas.

Se você mudar o conteúdo do keystore ou do armazenamento confiável JSSE ou o local do arquivo de keystore ou de armazenamento confiável, os aplicativos IBM MQ classes for JMS que estão em execução no momento não selecionarão automaticamente as mudanças. Para que as mudanças entrem em vigor, as seguintes ações devem ser executadas:

- Os aplicativos devem fechar todas as suas conexões e destruir quaisquer conexões não usadas em conjuntos de conexões.
- Se seu provedor JSSE armazenar em cache informações do keystore e do armazenamento confiável, essas informações deverão ser atualizadas.

Após essas ações terem sido executadas, os aplicativos poderão, então, recriar suas conexões.

Dependendo de como você projeta seus aplicativos e sobre a função fornecida pelo seu provedor JSSE, pode ser possível executar estas ações sem parar e reiniciar seus aplicativos. No entanto, parar e reiniciar o aplicativo poderá ser a solução mais simples.

CipherSpecs e CipherSuites TLS no IBM MQ classes for JMS
A capacidade de aplicativos IBM MQ classes for JMS para estabelecer conexões com um gerenciador de filas, depende do CipherSpec especificado na extremidade do servidor do canal MQI e o CipherSuite especificado na extremidade do cliente.

A tabela a seguir lista os CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes.

Deprecated É necessário revisar o tópico CipherSpecs descontinuados para ver se algum dos CipherSpecs, listados na tabela a seguir, foi descontinuado pelo IBM MQ e, se sim, em qual atualização o CipherSpec foi descontinuado.

Importante: Os CipherSuites listados são aqueles suportados pelo IBM Java Runtime Environment (JRE) fornecido com IBM MQ. Os CipherSuites listados incluem aqueles suportados pelo Oracle Java JRE. Para obter mais informações sobre como configurar seu aplicativo para usar um JRE Oracle Java, consulte Configurando seu aplicativo para usar mapeamentos de CipherSuite do IBM Java ou do Oracle Java.

A tabela também indica o protocolo usado para a comunicação e se o CipherSuite está em conformidade com o padrão FIPS 140-2 ou não.

Nota: No AIX, Linux, and Windows, IBM MQ fornece conformidade FIPS 140-2 por meio do módulo criptográfico IBM Crypto for C (ICC) . O certificado deste módulo foi movido para o status Histórico. Os clientes devem visualizar o IBM Crypto for C (ICC) certificado e estar ciente de qualquer aviso fornecido pelo NIST Um módulo FIPS 140-3 de substituição está atualmente em andamento e seu status pode ser visualizado procurando por ele na NIST CMVP modules in process list.

O IBM MQ Operator 3.2.0 e a imagem do contêiner do gerenciador de filas 9.4.0.0 em diante são baseados no UBI 9 A conformidade do FIPS 140-3 está pendente atualmente e seu status pode ser

visualizado procurando "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" no NIST CMVP modules in process list.

Ciphersuites denotados como compatíveis com FIPS 140-2 podem ser usados se o aplicativo não tiver sido configurado para impor a conformidade com FIPS 140-2, mas se a conformidade com FIPS 140-2 tiver sido configurada para o aplicativo (consulte as seguintes notas sobre configuração), apenas aqueles que estiverem marcados como compatíveis com FIPS 140-2 CipherSuites poderão ser configurados; tentar usar outro CipherSuites resulta em erro.

Nota: Cada JRE pode ter vários provedores de segurança criptográficos, com cada um podendo contribuir com uma implementação do mesmo CipherSuite. No entanto, nem todos os provedores de segurança são certificados pelo FIPS 140-2. Se a conformidade com FIPS 140-2 não é imposta para um aplicativo, então, é possível que uma implementação não certificada do CipherSuite possa ser usada. As implementações podem não funcionar em conformidade com FIPS 140-2, mesmo que o CipherSuite teoricamente atenda ao nível de segurança mínimo exigido pelo padrão. Consulte as notas a seguir para obter mais informações sobre a configuração do cumprimento FIPS 140-2 nos aplicativos IBM MQ JMS.

Para obter mais informações sobre conformidade de FIPS 140-2 e Conjunto B para CipherSpecs e CipherSuites, veja Especificando CipherSpecs. Também pode ser necessário conhecer as informações referentes às Normas Federais de Processamento de Informações dos EUA.

Para usar o conjunto completo de CipherSuites e para operar com FIPS 140-2 e/ou conformidade Suite-B, um JRE adequado é necessário.IBM Java 7 Service Refresh 4 Fix Pack 2 ou um nível superior de IBM JRE fornece o suporte apropriado para o TLS 1.2 CipherSuites listado em Tabela 41 na página 265.

Para poder usar Cifras TLS 1.3 , o JRE em execução em seu aplicativo deve suportar TLS 1.3.

Nota: Para usar alguns CipherSuites, o 'irrestrito' política de arquivos precisam ser configurados no JRE. Para obter mais detalhes sobre como os arquivos de políticas são configurados em um SDK ou JRE, consulte o tópico *Arquivos de políticas do IBM SDK* na *Referência de segurança para o IBM SDK, Java Technology Edition* para a versão que você está usando.

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	no
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sim

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sim
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	no

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no
TLS_RSA_WITH_3DES_EDE_CBC_SHA "2" na página 284	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	não "4" na página 284

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	não "4" na página 284
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	não "4" na página 284

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	não "4" na página 284
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	não "4" na página 284

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	não "4" na página 284
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	não "4" na página 284

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	no
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	no

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	no
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	sim
TLS_AES_128_GCM_SHA256 "3" na página 284	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	no

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_AES_256_GCM_SHA384 "3" na página 284	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	no
TLS_CHACHA20_POLY1305_SHA256 "3" na página 284	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	no

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_AES_128_CCM_SHA256 "3" na página 284	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	no
TLS_AES_128_CCM_8_SHA256 "3" na página 284	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	no
QUALQUER "3" na página 284	*ANY	*ANY	Múltiplos	no
ANY_TLS13 "3" na página 284	*TLS13	*TLS13	TLS V13	no

Tabela 41. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)				
CipherSpec **"1" na página 284**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ANY_TLS12_OR_HIGHER "3" na página 284	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 e superior	no
ANY_TLS13_OR_HIGHER "3" na página 284	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 e acima	no

Notas:

1. Este é o valor configurado em um canal no IBM MQ, incluindo em um CCDT (binário ou JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.
3. Para poder usar Cifras TLS v1.3 , o Java runtime environment (JRE) que está executando seu aplicativo deve suportar TLS v1.3.
4. V 9.4.0 V 9.4.0 No IBM MQ 9.4.0, o JRE do IBM Java 8 remove o suporte para a troca de chave RSA ao operar no modo FIPS.

Configurando Ciphersuites e a conformidade com FIPS em um aplicativo IBM MQ classes for JMS

- Um aplicativo que usa IBM MQ classes for JMS pode usar um dos dois métodos para configurar o CipherSuite para uma conexão:

  - Chame o método setSSLCipherSuite de um objeto ConnectionFactory.
  - Use a ferramenta de administração do IBM MQ JMS para configurar a propriedade SSLCIPHERSUITE de um objeto ConnectionFactory.
- Um aplicativo que usa o IBM MQ classes for JMS pode usar um dos dois métodos para cumprir a conformidade com FIPS 140-2:
  - Chame o método setSSLFipsRequired de um objeto ConnectionFactory.
  - Use a ferramenta de administração do IBM MQ JMS para configurar a propriedade SSLFIPSREQUIRED de um objeto ConnectionFactory.

Configurando seu aplicativo para usar mapeamentos do IBM Java ou Oracle Java CipherSuite

V 9.4.0 No IBM MQ 9.4.0, um Cipher pode ser definido como o nome CipherSpec ou CipherSuite e é manipulado corretamente pelo IBM MQ.

Nota: Removed A Java Propriedade do sistema `com.ibm.mq.cfg.useIBMCipherMappings`, que controlava quais mapeamentos eram usados em versões anteriores de IBM MQ, não é mais necessária, e é removida do produto em IBM MQ 9.4.0

limitações de interoperabilidade

Determinados CipherSuites podem ser compatíveis com mais de um CipherSpec IBM MQ, dependendo do protocolo em uso. No entanto, apenas a combinação CipherSuite/CipherSpec que usa a versão de TLS especificada na Tabela 1 é suportada. Tentando usar as combinações não suportadas de CipherSuites e CipherSpecs falhará com uma exceção apropriada. Instalações usando qualquer uma dessas combinações CipherSuite/CipherSpec deve mover para uma combinação suportada.

A tabela a seguir mostra os CipherSuites para o qual essa limitação aplica-se.

Tabela 42. CipherSuites e suas CipherSpecs suportados e não suportados

CipherSuite	**Suportado TLS CipherSpec**	**CipherSpec SSL não suportado**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na página 285	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Nota:

1. Deprecated Esse CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.

Gravando saídas de canal no Java for IBM MQ classes for JMS
Cria-se as saídas de canal definindo as classes do Java que implementam interfaces especificadas.

Para obter uma introdução às saídas de segurança, inicie com o tópico Programas de saída de segurança do canal

Três interfaces estão definidas no pacote com.ibm.mq.exits:

- WMQSendExit, para uma saída de envio
- WMQReceiveExit, para uma saída de recebimento
- WMQSecurityExit, para uma saída de segurança

O seguinte código de amostra define uma classe que implementa todas as três interfaces:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the security exit here
  }
}
```

Cada saída recebe como parâmetros um objeto MQCXP e um objeto MQCD. Estes objetos representam as estruturas MQCXP e MQCD definidas na interface processual.

Quando uma saída de envio é chamada, o parâmetro agentBuffer conterá os dados que estiverem prestes a ser enviados ao gerenciador de filas do servidor. Um parâmetro de comprimento não é necessário porque a expressão agentBuffer.limit() fornece o comprimento dos dados. A saída de envio retorna em seu valor os dados a serem enviados ao gerenciador de filas do servidor. No entanto, se a saída de envio não for a última saída de envio em uma sequência de saídas de envio, os dados retornados serão passados em vez da próxima saída de envio da sequência. A saída de envio pode retornar uma versão modificada dos dados que ele recebe no parâmetro agentBuffer ou pode retornar dados inalterados. O corpo de saída mais simples possível é portanto:

```
{ return agentBuffer; }
```

Quando uma saída de recebimento é chamada, o parâmetro agentBuffer conterá os dados que foram recebidos do gerenciador de filas do servidor. A saída de recebimento retorna em seu valor os dados a serem transmitidas para o aplicativo pelo IBM MQ classes for JMS. No entanto, se a saída de recebimento não for a última saída de recebimento em uma sequência de saídas de recebimento, os dados retornados serão passados para próxima saída de recebimento na sequência.

Quando uma saída de segurança for chamada, o parâmetro agentBuffer conterá os dados que foram recebidos em um fluxo de segurança na saída de segurança ao final do servidor da conexão. A saída de segurança retorna em seu valor os dados a serem enviados em um fluxo de segurança para a saída de segurança do servidor.

Saídas de canal são chamadas com um buffer que tem uma matriz auxiliar. Para melhor desempenho, a saída deve retornar um buffer com uma matriz auxiliar.

Até 32 de dados do usuário podem ser transmitidos para uma saída de canal quando é chamada. A saída acessa os dados do usuário chamando o método getExitData() do objeto MQCXP. Embora a saída possa mudar os dados do usuário ao chamar o método setExitData(), os dados do usuário são atualizados sempre que a saída é chamada. Quaisquer mudanças feitas nos dados do usuário são, portanto, perdidas. No entanto, a saída pode passar os dados de uma chamada para a próxima usando a área do usuário de

saída do objeto MQCXP. A saída acessa a área do usuário de saída por referência, chamando o método getExitUserArea().

Cada classe de saída deve ter um construtor. O construtor pode ser o construtor padrão, conforme mostrado no exemplo anterior ou um construtor com um parâmetro de sequência. O construtor é chamado para criar uma instância da classe de saída para cada saída definida na classe. Portanto, no exemplo anterior, uma instância da classe MyMQExits é criada para a saída de envio, outra instância é criada para a saída de recebimento e uma terceira instância é criada para a saída de segurança. Quando um construtor com um parâmetro de sequência for chamado, o parâmetro conterá os mesmos dados de usuário que são passados para a saída de canal para a qual a instância está sendo criada. Se uma classe de saída tiver um construtor padrão e um único construtor de parâmetro, esse único construtor de parâmetro terá precedência.

Não feche a conexão de dentro de uma saída do canal.

Quando os dados são enviados para a extremidade do servidor de uma conexão, a criptografia TLS é executada *após* quaisquer saídas de canal serem chamadas. De modo semelhante, quando os dados são recebidos da extremidade do servidor de uma conexão, a criptografia TLS é executada *antes* de quaisquer saídas de canal serem chamadas.

Em versões do IBM MQ classes for JMS anteriores a IBM WebSphere MQ 7.0, saídas de canal foram implementados usando as interfaces MQSendExit, MQReceiveExit, e MQSecurityExit. Ainda é possível usar essas interfaces, mas as novas interfaces são preferenciais para melhor função e desempenho.

Configurando o IBM MQ classes for JMS para usar saídas de canal
Um aplicativo IBM MQ classes for JMS pode usar a segurança do canal, enviar e receber saídas no canal MQI que é iniciado quando o aplicativo se conecta a um gerenciador de filas. O aplicativo pode usar saídas por escrito em Java, C ou C++. O aplicativo também pode usar uma sequência de envio ou receber saídas que são executadas em sucessão.

As propriedades a seguir são usadas para especificar uma saída de envio ou uma sequência de saídas de envio, usada por uma conexão JMS:

- A propriedade **SENDEXIT** de um objeto MQConnectionFactory.
- A propriedade **sendexit** em uma especificação de ativação usada pelo adaptador de recursos do IBM MQ para comunicação de entrada.
- A propriedade **sendexit** em um objeto ConnectionFactory usado pelo adaptador de recursos do IBM MQ para comunicação de saída.

O valor da propriedade é uma sequência que consiste em um ou mais itens separados por vírgulas. Cada item identifica uma saída de envio de uma das maneiras a seguir:

- O nome de uma classe que implementa a interface `WMQSendExit` para uma saída de envio escrita em Java.
- Uma sequência no formato *libraryName* (*entryPointName*) para uma saída de envio escrita em C ou C++.

De maneira semelhante, as propriedades a seguir especificam a saída de recebimento, ou sequência de saídas de recebimento, usada por uma conexão:

- A propriedade **RECEXIT** de um objeto MQConnectionFactory.
- A propriedade **receiveexit** em uma especificação de ativação usada pelo adaptador de recursos do IBM MQ para comunicação de entrada.
- A propriedade **receiveexit** em um objeto ConnectionFactory usado pelo adaptador de recursos do IBM MQ para comunicação de saída.

As propriedades a seguir especificam a saída de segurança usada por uma conexão:

- A propriedade **SECEXIT** de um objeto MQConnectionFactory.
- A propriedade **securityexit** em uma especificação de ativação usada pelo adaptador de recursos do IBM MQ para comunicação de entrada.

- A propriedade **securityexit** em um objeto ConnectionFactory usado pelo adaptador de recursos do IBM MQ para comunicação de saída.

Para MQConnectionFactories, é possível configurar as propriedades **SENDEXIT**, **RECEXIT** e **SECEXIT** usando a ferramenta de administração IBM MQ JMS ou IBM MQ Explorer. Como alternativa, um aplicativo pode configurar as propriedades chamando os métodos `setSendExit()`, `setReceiveExit()` e `setSecurityExit()`.

As saídas de canal são carregadas por seu próprio carregador de classe. Para localizar uma saída do canal, o carregador de classes procura os locais a seguir na ordem especificada.

1. O caminho de classe especificado pela propriedade **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** ou pelo atributo **JavaExitsClassPath** na sub-rotina Channels do arquivo de configuração do cliente IBM MQ.
2. Deprecated O caminho de classe especificado pela propriedade de sistema Java **com.ibm.mq.exitClasspath**. Observe que essa propriedade agora está descontinuada.
3. O diretório de saídas do IBM MQ, conforme mostrado em Tabela 43 na página 288. O carregador de classes primeiro procura no diretório os arquivos de classe que não são compactados em arquivos Java archive (JAR). Se a saída do canal não for encontrada, o carregador de classes, em seguida, procurará os arquivos JAR no diretório.

Tabela 43. O diretório de saídas do IBM MQ

Plataforma	Diretório
Linux AIX AIX and Linux	`/var/mqm/exits` (32-bit saídas de canal) `/var/mqm/exits64` (64-bit saídas de canal)
Windows Windows	*install_data_dir* \exits em que *install_data_dir* é o diretório que você escolheu para os arquivos de dados do IBM MQ durante a instalação. O diretório padrão é C:\ProgramData \IBM \MQ.

Nota: Se uma saída de canal existir em mais de um local, o IBM MQ classes for JMS carregará a primeira instância que ele encontrar.

O pai do carregador de classes é o carregador de classes que é usado para carregar o IBM MQ classes for JMS. Portanto, é possível que o carregador de classes pai carregue uma saída do canal se ela não puder ser localizada em nenhum dos locais precedentes. No entanto, quando você estiver usando o IBM MQ classes for JMS em um ambiente como um servidor de aplicativos JEE, provavelmente não poderá influenciar na escolha do carregador de classes-pai e, portanto, o carregador de classes deverá ser definido configurando a propriedade de sistema Java **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** no servidor de aplicativos.

Se seu aplicativo estiver sendo executado com o Java security manager ativado, o arquivo de configuração de política usado pelo ambiente de tempo de execução Java no qual o aplicativo está em execução deverá ter as permissões para carregar uma classe de saída de canal. Para obter informações sobre como fazer isso, consulte Executando IBM MQ classes para aplicativos JMS no Java Security Manager.

As interfaces `MQSendExit`, `MQReceiveExit` e `MQSecurityExit` fornecidas com versões anteriores ao IBM WebSphere MQ 7.0 ainda são suportadas. Se você usar saídas de canal que implementam essas interfaces, `com.ibm.mq.jar` deve estar presente no caminho de classe.

Para obter informações sobre como gravar as saídas de canal em C, consulte “Programas de Saída de Canal para Canais de Mensagens” na página 973. Deve-se armazenar programas de saída de canal gravados em C ou C++ no diretório mostrado em Tabela 43 na página 288.

Se o seu aplicativo usar uma tabela de definição do canal do cliente (CCDT) para conectar a um gerenciador de filas, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS” na página 289.

Especificando os dados do usuário a serem transmitidos para as saídas de canal ao usar o IBM MQ classes for JMS
Até 32 de dados do usuário podem ser transmitidos para uma saída de canal quando é chamada.

A propriedade SENDEXITINIT de um objeto MQConnectionFactory especifica os dados do usuário que são transmitidos para cada saída de envio quando ele é chamado. O valor da propriedade é uma sequência que consiste em um ou mais itens de dados do usuário separados por vírgulas. A posição de cada item de dados do usuário dentro da sequência determina para qual saída de envio, em uma sequência de saídas de envio, os dados do usuário são transmitidos. Por exemplo, o primeiro item de dados do usuário na sequência é transmitido para a primeira saída de envio em uma sequência de saídas de envio.

É possível configurar a propriedade SENDEXITINIT usando a ferramenta de administração do IBM MQ JMS ou IBM MQ Explorer. Como alternativa, um aplicativo pode configurar a propriedade chamando o método setSendExitInit().

De forma semelhante, a propriedade RECEXITINIT de um objeto ConnectionFactory especifica os dados do usuário que são transmitidos para cada saída de recebimento e a propriedade SECEXITINIT especifica os dados do usuário transmitidos para uma saída de segurança. É possível configurar essas propriedades usando a ferramenta de administração do IBM MQ JMS ou do IBM MQ Explorer. Como alternativa, um aplicativo pode definir as propriedades chamando os métodos setReceiveExitInit() e setSecurityExitInit().

Observe as regras a seguir ao especificar dados do usuário que são transmitidos para as saídas de canal:

- Se o número de itens de dados do usuário em uma sequência for maior que o número de saídas em uma sequência, os itens em excesso de dados do usuário são ignorados.
- Se o número de itens de dados do usuário em uma sequência for menor que o número de saídas em uma sequência, cada item não especificado de dados do usuário será configurado para uma sequência vazia. Duas vírgulas em sucessão dentro de uma sequência ou uma vírgula no início de uma sequência, também denotam um item não especificado de dados do usuário.

Se um aplicativo usar uma tabela de definição de canal do cliente (CCDT) para se conectar a um gerenciador de filas, quaisquer dados do usuário especificados em uma definição de canal de conexão do cliente serão transmitidos para as saídas de canal quando forem chamados. Para obter mais informações sobre como usar uma tabela de definição de canal do cliente, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS” na página 289.

Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS
Um aplicativo IBM MQ classes for JMS pode usar definições de canal de conexão do cliente armazenadas em uma tabela de definição de canal do cliente (CCDT). Configure um objeto ConnectionFactory para usar a CCDT. Existem algumas restrições sobre seu uso.

Como uma alternativa para criar uma definição de canal de conexão do cliente configurando determinadas propriedades de um objeto ConnectionFactory, um aplicativo IBM MQ classes for JMS pode usar definições de canal de conexão do cliente armazenadas em uma tabela de definição de canal do cliente. Essas definições são criadas pelos comandos do IBM MQ Script (MQSC) ou os comandos do IBM MQ Programmable Command Format (PCF). Quando o aplicativo criar um objeto Connection, IBM MQ classes for JMS irá procurar a tabela de definição de canal do cliente para uma definição de canal de conexão do cliente adequada e usará a definição de canal para iniciar um canal MQI. Para obter mais informações sobre tabelas de definição de canal do cliente e como construir um, consulte Client Channel Definition Table.

Para usar uma tabela de definição de canal do cliente, a propriedade CCDTURL de um objeto ConnectionFactory deve ser configurada para um objeto de URL. O IBM MQ classes for JMS não lê as informações sobre a CCDT no arquivo de configuração do IBM MQ MQI client, embora alguns outros valores sejam usados daí (veja “O arquivo de configuração IBM MQ classes for JMS/Jakarta Messaging” na página 100 para saber qual valor se aplica). O objeto de URL contém um localizador uniforme de recursos (URL) que identifica o nome e o local do arquivo contendo a tabela de definição de canal do cliente e especifica como o arquivo pode ser acessado. É possível configurar a propriedade CCDTURL usando a ferramenta de administração do IBM MQ JMS ou um aplicativo pode configurar a propriedade criando um objeto de URL e chamando o método setCCDTURL() do objeto ConnectionFactory.

Por exemplo, se o ccdt1.tab do arquivo contiver uma tabela de definição de canal do cliente e for armazenado no mesmo sistema no qual o aplicativo está em execução, o aplicativo pode configurar a propriedade CCDTURL da seguinte maneira:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Como outro exemplo, suponha que o arquivo ccdt2.tab contenha uma tabela de definição de canal do cliente e esteja armazenada em um sistema diferente daquele no qual o aplicativo está em execução. Se o arquivo puder ser acessado usando o protocolo FTP, o aplicativo poderá configurar a propriedade CCDTURL da seguinte maneira:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

Além da configuração da propriedade CCDTURL do objeto ConnectionFactory, a propriedade QMANAGER do mesmo objeto deve ser configurada como um dos seguintes valores:

- O nome de um gerenciador de filas
- Um asterisco (*) seguido pelo nome de um grupo de gerenciadores de filas

Estes são os mesmos valores que podem ser usados para o parâmetro **QMgrName** em uma chamada MQCONN emitida por um aplicativo cliente que está usando o Message Queue Interface (MQI). Para obter mais informações sobre o significado desses valores, portanto, consulte MQCONN. É possível configurar a propriedade QMANAGER usando a ferramenta de administração do IBM MQ JMS ou IBM MQ Explorer. Como alternativa, um aplicativo pode configurar a propriedade chamando o método setQueueManager() do objeto ConnectionFactory.

Se um aplicativo então criar um objeto Connection a partir do objeto ConnectionFactory, IBM MQ classes for JMS acessará a tabela de definição de canal do cliente identificada pela propriedade CCDTURL, usará a propriedade QMANAGER para procurar a tabela de uma definição de canal de conexão do cliente adequado e, em seguida, usará a definição de canal para iniciar um canal MQI para um gerenciador de filas.

Observe que as propriedades CCDTURL e CHANNEL de um objeto ConnectionFactory não poderão ser ambas configuradas quando o aplicativo chamar o método createConnection(). Se ambas propriedades forem configuradas, o método lançará uma exceção. A propriedade CCDTURL ou CHANNEL será considerada para ser configurada ou se seu valor for algo diferente de nulo, uma sequência vazia ou uma sequência que contém todos os caracteres em branco.

Quando o IBM MQ classes for JMS localizar uma definição de canal de conexão do cliente apropriada na tabela de definição de canal do cliente, ele usará somente as informações extraídas da tabela para iniciar um canal MQI. Quaisquer propriedades relacionadas ao canal do objeto ConnectionFactory são ignoradas.

Em particular, observe os pontos a seguir se você estiver usando TLS:

- Um canal MQI usará TLS somente se a definição de canal extraída da tabela de definição de canal de cliente especificar o nome de um CipherSpec suportado pelo IBM MQ classes for JMS.
- Uma tabela de definição de canal do cliente também contém informações sobre o local dos servidores Lightweight Directory Access Protocol (LDAP) que mantém as listas de revogação de certificado (CRLs). IBM MQ classes for JMS usa apenas essas informações para acessar os servidores LDAP que mantém CRLs.
- Uma tabela de definição de canal do cliente também pode conter o local de um respondente do OCSP. IBM MQ classes for JMS não pode usar as informações do OCSP em um arquivo da tabela de definição de canal do cliente. No entanto, é possível configurar o OCSP, conforme descrito na seção Online Certificate Status Protocol (OCSP) nos aplicativos clientes Java e JMS.

Para obter mais informações sobre como usar o TLS com uma tabela de definição de canal de cliente, veja Usando o cliente transacional estendido com canais TLS.

Observe também os pontos a seguir se estiver usando saídas de canal:

- Um canal MQI usa somente as saídas de canal e dados do usuário associados especificados pela definição de canal extraída da tabela de definição de canal do cliente.
- Uma definição de canal extraída de uma tabela de definição de canal do cliente pode especificar as saídas do canal gravadas em Java. Isso significa, por exemplo, que o parâmetro SCYEXIT no comando DEFINIR CANAL para criar uma nova definição de canal de conexão do cliente pode especificar o nome de uma classe que implementa a interface WMQSecurityExit. De forma semelhante, o parâmetro SENDEXIT pode especificar o nome de uma classe que implementa a interface WMQSendExit e o parâmetro RCVEXIT pode especificar o nome de uma classe que implementa a interface WMQReceiveExit. Para obter mais informações sobre como gravar uma saída de canal no Java, consulte"Gravando saídas de canal no Java for IBM MQ classes for JMS" na página 285.

  O uso de saídas de canais gravadas em uma linguagem diferente do Java também é suportado. Para obter informações sobre como especificar os parâmetros SCYEXIT, SENDEXIT e RCVEXIT no comando DEFINE CHANNEL para saídas do canais gravadas em outra linguagem, consulte DEFINE CHANNEL.

Reconexão automática do cliente JMS
Configure o cliente JMS para se reconectar automaticamente seguindo uma rede, um gerenciador de filas ou uma falha do servidor.

Normalmente, se um aplicativo IBM MQ classes for JMS independente é conectado a um gerenciador de filas usando o transporte do cliente e o gerenciador de filas se torna indisponível por alguma razão (devido a uma indisponibilidade de rede, uma falha do gerenciador de filas ou o gerenciador de filas está sendo interrompido, por exemplo), o IBM MQ classes for JMS lança uma JMSException da próxima vez que o aplicativo tenta se comunicar com o gerenciador de filas. O aplicativo deve capturar a JMSException e tentar se reconectar ao gerenciador de filas. É possível simplificar o design do aplicativo, ativando a reconexão automática do cliente. Quando o gerenciador de filas se tornar indisponível, o IBM MQ classes for JMS tentará se reconectar ao gerenciador de filas automaticamente em nome do aplicativo. Isso significa que o aplicativo não precisa conter lógica para se reconectar.

O uso dessa implementação de reconexão automática do cliente não é suportado dentro dos servidores de aplicativos do Java Platform, Enterprise Edition. Consulte "Usando a reconexão automática do cliente em ambientes Java EE" na página 297 para obter uma implementação alternativa.

Usando a reconexão automática do cliente JMS
Se um aplicativo do IBM MQ classes for JMS independente usar uma Connection Factory que tenha o conjunto de propriedades CONNECTIONNAMELIST ou CCDTURL, o aplicativo será elegível para usar a reconexão automática do cliente.

A reconexão do cliente automática pode ser usada para reconectar aos gerenciadores de filas, incluindo aqueles que são parte de uma configuração de alta disponibilidade (HA). As configurações de HA incluem gerenciadores de filas de várias instâncias, gerenciadores de filas do RDQM ou gerenciadores de filas de HA em um dispositivo IBM MQ.

O comportamento da funcionalidade de reconexão automática do cliente que é fornecido pelo IBM MQ classes for JMS depende das propriedades a seguir:

A propriedade TRANSPORT (nome abreviado TRAN) do JMS Connection Factory

TRANSPORT especifica como os aplicativos que usam o Connection Factory se conectam a um gerenciador de filas. Essa propriedade deve ser configurada para o valor CLIENT para que a reconexão automática do cliente seja usada. A reconexão automática do cliente não está disponível para aplicativos que se conectam a um gerenciador de filas que usa um Connection Factory que tem a propriedade TRANSPORT configurada para BIND, DIRECT ou DIRECTHTTP.

A propriedade QMANAGER (nome abreviado QMGR) do JMS Connection Factory

A propriedade QMANAGER especifica o nome do gerenciador de filas ao qual o Connection Factory se conecta.

A propriedade CONNECTIONNAMELIST (nome abreviado CRHOSTS) do JMS Connection Factory

A propriedade CONNECTIONNAMELIST é uma lista separada por vírgula, em que cada entrada contém informações sobre o nome do host e a porta que devem ser usados para conectar-se ao

gerenciador de filas especificado pela propriedade QMANAGER quando estiver usando o transporte CLIENT. A lista tem o formato a seguir: nome do host(porta), nome do host(porta).

A propriedade CCDTURL (nome abreviado CCDT) do JMS Connection Factory

A propriedade CCDTURL aponta para a tabela de definição de canal do cliente que o IBM MQ classes for JMS usa quando se conecta a um gerenciador de filas usando uma CCDT.

A propriedade CLIENTRECONNECTOPTIONS (nome abreviado CROPT) do JMS Connection Factory

CLIENTRECONNECTOPTIONS controla se o IBM MQ classes for JMS tentará se conectar automaticamente a um gerenciador de filas em nome de um aplicativo se um gerenciador de filas for disponibilizado.

O atributo DefRecon na sub-rotina de Canais do arquivo de configuração do cliente

O atributo DefRecon fornece uma opção administrativa para ativar todos os aplicativos para se reconectarem automaticamente ou para desativar a reconexão automática para aplicativos que são escritos para se reconectarem automaticamente.

A reconexão automática do cliente está disponível somente quando um aplicativo se conecta com sucesso a um gerenciador de filas.

Quando um aplicativo se conecta a um gerenciador de filas que usa o transporte CLIENT, o IBM MQ classes for JMS usa o valor da propriedade do Connection Factory CLIENTRECONNECTOPTIONS para determinar se a reconexão automática do cliente deve ser usada, caso o gerenciador de filas ao qual o aplicativo está conectado ficar indisponível. A Tabela 1 mostra os valores possíveis para a propriedade CLIENTRECONNECTOPTIONS e o comportamento do IBM MQ classes for JMS para cada um desses valores:

Tabela 44. Valores possíveis da propriedade CLIENTRECCECTOPTIONS.

CLIENTRECONNECTOPTIONS	**Comportamento do IBM MQ classes for JMS**
QUALQUER	Se CONNECTIONNAMELIST estiver configurado, use o valor da propriedade CONNECTIONNAMELIST para abrir uma conexão com uma combinação de nome de host e porta e se conectar a qualquer gerenciador de filas. Para usar essa opção de reconexão automática do cliente, a propriedade QMANAGER deve ser configurada para o valor padrão ou "*". Se CCDTURL for configurado, abra a tabela de definição de canal do cliente especificada pela propriedade CCDTURL, selecione uma entrada na tabela e, em seguida, use essa entrada para iniciar um canal de conexão do cliente com um gerenciador de filas. Para usar essa opção reconexão automática do cliente, a propriedade QMANAGER deve ser configurada para um dos seguintes: • Um asterisco (*) • Um asterisco (*) seguido pelo nome de um grupo de gerenciadores de filas • Uma sequência de caracteres vazia ou uma sequência que contém todos os caracteres em branco
ASDEF	Usar o valor de DefRecon para determinar se a reconexão automática do cliente está disponível.

Tabela 44. Valores possíveis da propriedade CLIENTRECCECTOPTIONS. (continuação)	
CLIENTRECONNECTOPTIONS	**Comportamento do IBM MQ classes for JMS**
DISABLED	Não execute nenhuma reconexão automática do cliente e retorne uma JMSException para o aplicativo.
QMGR	Especifica que o cliente deve se reconectar ao mesmo gerenciador de filas. Essa opção deve ser usada para soluções de alta disponibilidade, em que a reconexão com outra instância do mesmo gerenciador de filas é necessária. Se CONNECTIONNAMELIST estiver configurado, use o valor da propriedade CONNECTIONNAMELIST para abrir uma conexão com uma combinação de nome do host e porta e conecte-se ao gerenciador de filas especificado pela propriedade QMANAGER. Se CCDTURL for configurado, abra a tabela de definição de canal do cliente especificada pela propriedade CCDTURL, localize as entradas na tabela que correspondem ao nome do gerenciador de filas especificado pela propriedade QMANAGER e, em seguida, use essas entradas para iniciar um canal de conexão do cliente para esse gerenciador de filas.

Se CONNECTIONNAMELIST estiver configurado, ao executar a reconexão automática do cliente, o IBM MQ classes for JMS usará as informações na propriedade CONNNECTIONNAMELIST da Connection Factory para determinar a qual sistema se reconectar.

O IBM MQ classes for JMS tenta inicialmente se reconectar usando o nome do host e a porta especificados na primeira entrada em CONNECTIONNAMELIST. Se uma conexão for feita, o IBM MQ classes for JMS tenta, então, se conectar ao gerenciador de filas que tem o nome especificado na propriedade QMANAGER. Se uma conexão com o gerenciador de filas puder ser estabelecida, o IBM MQ classes for JMS reabre todos os objetos do IBM MQ que o aplicativo tinha aberto antes da reconexão automática do cliente e eles continuam a execução como antes.

Se uma conexão não puder ser estabelecida com o gerenciador de filas requerido usando a primeira entrada em CONNECTIONNAMELIST, o IBM MQ classes for JMS tenta a segunda entrada em CONNECTIONNAMELIST e assim por diante.

Quando o IBM MQ classes for JMS tiver tentado todas as entradas em CONNECTIONNAMELIST, ele espera um período de tempo antes de tentar reconectar novamente. Para executar a nova tentativa de reconexão, o IBM MQ classes for JMS inicia com a primeira entrada em CONNECTIONNAMELIST. Em seguida, tenta cada entrada em CONNECTIONNAMELIST até que uma reconexão ocorra ou que o fim de CONNECTIONNAMELIST seja atingido, quando o IBM MQ classes for JMS espera um período de tempo antes de tentar novamente.

Se o CCDTURL for configurado, ao executar a reconexão automática do cliente, o IBM MQ classes for JMS usará a tabela de definição de canal do cliente especificada na propriedade CCDTURL para determinar a qual sistema se reconectar.

O IBM MQ classes for JMS analisa inicialmente a tabela de definição de canal do cliente e localiza uma entrada apropriada que corresponda ao valor da propriedade QMANAGER. Quando uma entrada for localizada, o IBM MQ classes for JMS tentará se reconectar ao gerenciador de filas requerido usando essa entrada. Se uma conexão com o gerenciador de filas puder ser estabelecida, o IBM MQ classes for

JMS reabre todos os objetos do IBM MQ que o aplicativo tinha aberto antes da reconexão automática do cliente e eles continuam a execução como antes.

Se uma conexão não puder ser estabelecida com o gerenciador de filas requerido, o IBM MQ classes for JMS procura outra entrada apropriada na tabela de definição de canal do cliente e tenta usar essa e assim por diante.

Quando o IBM MQ classes for JMS ter tentado todas as entradas apropriadas na tabela de definição de canal do cliente, ele espera um período de tempo antes de tentar reconectar novamente. Para executar a nova tentativa de reconexão, o IBM MQ classes for JMS analisa a tabela de definição de canal do cliente novamente e tenta a primeira entrada adequada. Ele tentará, então, cada entrada apropriada na tabela de definição de canal do cliente até que uma reconexão ocorra ou a última entrada apropriada na tabela de definição de canal do cliente tenha sido tentada, quando o IBM MQ classes for JMS espera um período de tempo antes de tentar novamente.

Se estiver usando CONNECTIONNAMELIST ou CCDTURL, o processo de reconexão automática do cliente continuará até que o IBM MQ classes for JMS seja reconectado com êxito ao gerenciador de filas especificado pela propriedade QMANAGER.

Por padrão, as tentativas de reconexão ocorrem nos intervalos a seguir:

- A primeira tentativa é feita após um atraso inicial de 1 segundo mais um elemento aleatório de até 250 milissegundos.
- A segunda tentativa é feita 2 segundos mais um intervalo aleatório de até 500 milissegundos após a primeira tentativa falhar.
- A terceira tentativa é feita 4 segundos mais um intervalo aleatório de até 1 segundo após a segunda tentativa falhar.
- A quarta tentativa é feita 8 segundos mais um intervalo aleatório de até 2 segundos após a terceira tentativa falhar.
- A quinta tentativa é feita 16 segundos mais um intervalo aleatório de até 4 segundos após a quarta tentativa falhar.
- A sexta tentativa e todas as tentativas subsequentes são feitas 25 segundos mais um intervalo aleatório de até 6 segundos e 250 milissegundos após a tentativa anterior falhar.

As tentativas de reconexão são atrasadas por intervalos que são parcialmente fixos e em parcialmente aleatórios. Isso é para evitar que todos os aplicativos IBM MQ classes for JMS que estavam conectados a um gerenciador de filas que não está mais disponível se reconectem simultaneamente.

Se precisar aumentar os valores padrão para refletir mais precisamente o tempo necessário para um gerenciador de filas se recuperar ou para um gerenciador de filas em espera ser ativado, modifique o atributo ReconDelay na sub-rotina Channel do arquivo de configuração do cliente. Para obter mais informações, consulte Sub-rotina CHANNELS do arquivo de configuração do cliente.

Se um aplicativo IBM MQ classes for JMS continuará a funcionar corretamente após ser reconectado automaticamente depende de seu design. Leia os tópicos relacionados para entender como projetar aplicativos que podem usar a funcionalidade de reconexão automática.

Códigos de razão que indicam que um gerenciador de filas não está mais disponível
Quais códigos de razão indicam que um gerenciador de filas não está mais disponível ou não pode ser alcançado ao tentar a reconexão automática do IBM MQ classes for JMS.

"Reconexão automática do cliente JMS" na página 291 fornece uma visão geral de JMSExceptions e como os seus aplicativos podem ser reiniciados automaticamente, além das informações em "Usando a reconexão automática do cliente JMS" na página 291 que detalham os requisitos para reconexão automática do cliente.

As informações a seguir listam os códigos de razão do IBM MQ que o seu aplicativo deve verificar:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

A maioria das JMSExceptions lançadas de volta para aplicativos corporativos contêm uma MQException vinculada que contém o código de razão. Para implementar a lógica de nova tentativa para os códigos de razão na lista anterior, os seus aplicativos corporativos devem verificar essa exceção vinculada usando código semelhante ao exemplo a seguir:

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

Conceitos relacionados
Classes do IBM MQ for JMS

Usando a reconexão automática do cliente em ambientes Java SE e Java EE
É possível fazer uso da reconexão do cliente automática do IBM MQ para facilitar várias soluções de alta disponibilidade (HA) e recuperação de desastre (DR) dentro de um ambiente Java SE e Java EE.

Várias soluções de HA e DR estão disponíveis em diferentes plataformas:

- Multi Gerenciadores de filas de várias instâncias são instâncias do mesmo gerenciador de filas configurado em servidores diferentes (consulte Gerenciadores de filas de várias instâncias). Uma instância do gerenciador de filas é definida como a instância ativa e outra instância é definida como a instância em espera. Se a instância ativa falhar, o gerenciador de filas de várias instâncias será reiniciado automaticamente no servidor de espera.

  Os gerenciadores de filas ativo e de espera possuem o mesmo identificador de gerenciador de filas (QMID). Os aplicativos clientes do IBM MQ que se conectam a um gerenciador de filas de várias instâncias podem ser configurados para se reconectarem automaticamente a uma instância em espera de um gerenciador de filas usando a reconexão automática do cliente.
- Linux O RDQM (gerenciador de filas de dados replicados) é uma solução de alta disponibilidade que está disponível nas plataformas Linux (consulte Alta disponibilidade do RDQM). Uma configuração do RDQM consiste em três servidores configurados em um grupo de alta disponibilidade (HA), cada um com uma instância do gerenciador de filas. Uma instância é o gerenciador de filas em execução, que sincronicamente replica os seus dados para as outras duas instâncias. Se o servidor que estiver

executando esse gerenciador de filas falhar, outra instância do gerenciador de filas será iniciada e terá dados atuais com os quais operar. As três instâncias do gerenciador de filas compartilham um endereço IP flutuante, de modo que os clientes precisam apenas ser configurados com um único endereço IP. Os aplicativos clientes que se conectam a um gerenciador de filas do RDQM podem ser configurados para se reconectarem automaticamente a uma instância em espera de um gerenciador de filas usando a reconexão automática do cliente.

- MQ Appliance Uma solução de alta disponibilidade também pode ser fornecida por um par de IBM MQ Appliances (veja Alta disponibilidade e Recuperação de desastre na documentação do IBM MQ Appliance). Um gerenciador de filas de HA é executado em um dos dispositivos, enquanto replica sincronicamente os dados para a instância de espera do gerenciador de filas no outro dispositivo. Se o dispositivo primário falhar, o gerenciador de filas será iniciado automaticamente e será executado no outro dispositivo. As duas instâncias do gerenciador de filas podem ser configuradas para compartilharem um endereço IP flutuante, para que os clientes precisem apenas ser configurados com um único endereço IP. Os aplicativos clientes que se conectam a um gerenciador de filas de HA em um IBM MQ Appliance podem ser configurados para se reconectarem automaticamente à instância de espera de um gerenciador de filas usando a reconexão automática do cliente.

Nota: Dentro de ambientes Java EE, como WebSphere Application Server, a reconexão automática do cliente com especificações de ativação usando a funcionalidade fornecida por IBM MQ classes for JMS não é suportada. O adaptador de recursos do IBM MQ fornecerá seu próprio mecanismo para reconectar as especificações de ativação se o gerenciador de filas que a especificação de ativação foi conectado se tornar indisponível. Para obter mais informações, consulte “Suporte para reconexão automática do cliente em ambientes Java EE” na página 298.

Conceitos relacionados

Gerenciadores de Filas de Várias Instâncias

Reconexão automática do cliente

Referências relacionadas

alta disponibilidade do RDQM

Usando a reconexão automática do cliente em ambientes Java SE

Aplicativos que usam o IBM MQ classes for JMS em execução em ambientes Java SE podem usar a funcionalidade de reconexão automática do cliente por meio da propriedade do connection factory **CLIENTRECONNECTOPTIONS**.

A propriedade do connection factory **CLIENTRECONNECTOPTIONS** usa duas propriedades de connection factory adicionais, **CONNECTIONNAMELIST** e **CCDTURL**, para determinar como se conectar ao servidor no qual o gerenciador de filas está em execução.

Propriedade CONNECTIONNAMELIST

A propriedade **CONNECTIONNAMELIST** é uma lista separada por vírgula que contém as informações de nome do host e da porta a serem usados para se conectar a um gerenciador de filas no modo cliente. Essa propriedade é usada com os valores **QMANAGER** e **CHANNEL**. Quando um aplicativo usa a propriedade **CONNECTIONNAMELIST** para criar uma conexão do cliente, o IBM MQ classes for JMS tenta se conectar a cada host na ordem da lista. Se o primeiro host do gerenciador de filas estiver indisponível, o IBM MQ classes for JMS tentará se conectar ao próximo host na lista. Se o final da lista de nomes de conexão for alcançado sem criar uma conexão, o IBM MQ classes for JMS lançará o código de razão `MQRC_QMGR_NOT_AVAILABLE` IBM MQ.

Se o gerenciador de filas ao qual o aplicativo está conectado falhar, todos os aplicativos que usavam uma **CONNECTIONNAMELIST** para se conectar a esse gerenciador de filas receberão uma exceção indicando que o gerenciador de filas não está disponível. O aplicativo deve capturar a exceção e limpar todos os recursos que estava usando. Para criar uma conexão, o aplicativo deve usar o connection factory. O connection factory tenta se conectar a cada host na ordem da lista novamente; o gerenciador de filas que falhou não está disponível agora. O connection factory tenta se conectar a outro host na lista.

Propriedade CCDTURL

A propriedade **CCDTURL** contém um Localizador Uniforme de Recursos (URL) que aponta para uma Client Channel Definition Table (CCDT); essa propriedade é usada com a propriedade **QMANAGER**. O CCDT contém uma lista de canais do cliente que são usados para se conectar a um gerenciador de filas definido em um sistema IBM MQ. Para obter informações sobre como os CCDTs são usados pelo IBM MQ classes for JMS, consulte "Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS" na página 289.

Usando a propriedade CLIENTRECONNECTOPTIONS para ativar a reconexão automática do cliente dentro do IBM MQ classes for JMS

A propriedade **CLIENTRECONNECTOPTIONS** é usada para ativar a reconexão automática do cliente dentro do IBM MQ classes for JMS. Os valores possíveis para essa propriedade são como a seguir:

ASDEF
O comportamento de reconexão automática do cliente é definido pelo valor padrão especificado na sub-rotina do canal do arquivo de configuração do cliente IBM MQ (`mqclient.ini`).

Desativado
A reconexão automática do cliente está desativada.

QMGR
A tentativa de conexão do IBM MQ classes for JMS a um gerenciador de filas com o mesmo identificador de gerenciador de filas do gerenciador de filas ao qual ele estava conectado, usando uma das opções a seguir:

- A propriedade **CONNECTIONNAMELIST** e o canal que está definido na propriedade **CHANNEL**.
- A CCDT definida na propriedade **CCDTURL**.

qualquer um
A tentativa de reconexão do IBM MQ classes for JMS a um gerenciador de filas de mesmo nome com o uso da propriedade **CONNECTIONNAMELIST** ou da **CCDTURL**.

Informações relacionadas
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente

Usando a reconexão automática do cliente em ambientes Java EE
O adaptador de recursos do IBM MQ, que pode ser implementado em ambientes do Java EE (Java Platform, Enterprise Edition) e o provedor de sistemas de mensagens do WebSphere Application Server IBM MQ usam o IBM MQ classes for JMS para se comunicar com gerenciadores de filas do IBM MQ. O adaptador de recursos IBM MQ e o provedor de mensagens WebSphere Application Server IBM MQ fornecem uma série de mecanismos para permitir especificações de ativação, portas de listener WebSphere Application Server e aplicativos sendo executados dentro de contêineres de clientes para se reconectar automaticamente a um gerenciador de filas. Os aplicativos Enterprise JavaBeans (EJBs) e aqueles baseados na web precisam implementar sua própria lógica de reconexão.

Nota: A reconexão do cliente automática com especificações de ativação usando a funcionalidade fornecida por IBM MQ classes for JMS não é suportada (consulte "Reconexão automática do cliente JMS" na página 291). O adaptador de recursos do IBM MQ fornecerá seu próprio mecanismo para reconectar as especificações de ativação se o gerenciador de filas que a especificação de ativação foi conectado se tornar indisponível.

O mecanismo que o adaptador de recursos fornece é controlado por:

- A propriedade do adaptador de recursos do IBM MQ **reconnectionRetryCount**.
- A propriedade do adaptador de recursos do IBM MQ **reconnectionRetryInterval**.
- A propriedade de especificação de ativação **connectionNameList**.

Para obter mais informações sobre essas propriedades, consulte"Configuração para propriedades do objeto ResourceAdapter" na página 458.

O uso da reconexão automática do cliente dentro de um método onMessage() do aplicativo bean acionado por mensagens ou qualquer outro aplicativo que esteja em execução no ambiente Java Platform, Enterprise Edition não é suportado. O aplicativo precisará implementar sua própria lógica de reconexão se o gerenciador de filas que estava conectado se tornar indisponível. Para obter informações adicionais, consulte “Implementando a lógica de reconexão em um aplicativo Java EE” na página 305.

Suporte para reconexão automática do cliente em ambientes Java EE
Dentro de ambientes Java EE, como WebSphere Application Server, o adaptador de recursos IBM MQ, e o provedor de mensagens WebSphere Application Server IBM MQ fornecem uma série de mecanismos que permitem que os aplicativos se reconectem a um gerenciador de filas automaticamente. No entanto, em alguns casos, são aplicadas restrições a esse suporte.

O adaptador de recursos do IBM MQ que pode ser implementado em ambientes Java EE e o provedor de sistemas de mensagens do WebSphere Application Server IBM MQ usam o IBM MQ classes for JMS para se comunicar com os gerenciadores de filas do IBM MQ.

A tabela a seguir resume o suporte que o adaptador de recursos do IBM MQ e o provedor de sistemas de mensagens do WebSphere Application Server IBM MQ fornecem para a reconexão automática do cliente.

Tabela 45. Resumo do suporte para opções de reconexão automática do cliente em ambientes do Java EE

Opções para a reconexão automática	**Propriedade CONNECTIONNAMELIST**	**Propriedade CCDTURL**	**Propriedade CLIENTRECONNECTOPTIONS**	**Abordagem alternativa para reconexão automática do cliente**
Especificações de ativação	Suportado com restrições	Suportado com restrições	Não Suportado	O ambiente do Java EE e as especificações de ativação fornecem seu próprio mecanismo de reconexão
Portas do listener do WebSphere Application Server	Suportado com restrições	Suportado com restrições	Não Suportado	O WebSphere Application Server fornece o seu próprio mecanismo de reconexão
Enterprise JavaBeans e aplicativos baseados na web	Suportado com restrições	Suportado com restrições	Não Suportado	O aplicativo deve implementar sua própria lógica de reconexão
Os aplicativos em execução dentro de contêineres do cliente	Suportado	Suportado	Suportado	Não-aplicável

Os aplicativos de bean acionado por mensagem que são instalados em um ambiente Java EE, como IBM MQ classes for JMS, podem usar especificações de ativação para processar mensagens em um sistema IBM MQ. As especificações de ativação são usadas para detectar mensagens que chegam em um sistema IBM MQ e entregá-las para os beans acionados por mensagens para processamento. Os beans acionados por mensagens também podem fazer mais conexões com os sistemas IBM MQ de dentro de seu método **onMessage()**. Para obter mais informações sobre como essas conexões podem usar a reconexão automática do cliente, consulte Aplicativos Enterprise JavaBeans e baseados na web.

Especificações de ativação
Para especificações de ativação, as propriedades **CONNECTIONNAMELIST** e **CCDTURL** são suportadas com restrições e a propriedade **CLIENTRECONNECTOPTIONS** não é suportada.

Aplicativos MDB (message-driven bean) instalados em um ambiente Java EE, como WebSphere Application Server, podem usar especificações de ativação para processar mensagens em um sistema IBM MQ.

As especificações de ativação são usadas para detectar mensagens que chegam em um sistema IBM MQ e, em seguida, entregá-las para os MDBs para processamento. Esta seção trata como a especificação de ativação monitora o sistema IBM MQ.

Os MDBs também podem fazer conexões adicionais com sistemas do IBM MQ de dentro de seu método `onMessage()`.

Os detalhes sobre como essas conexões podem usar a reconexão automática do cliente podem ser localizados no "Enterprise JavaBeans e aplicativos baseados na web" na página 303.

Propriedade CONNECTIONNAMELIST

Ao iniciar, a especificação de ativação tenta se conectar ao gerenciador de filas usando o:

- Um especificado na propriedade **QMANAGER**
- Canal mencionado na propriedade **CHANNEL**
- Informações de nome do host e da porta da primeira entrada na **CONNECTIONNAMELIST**

Se a especificação de ativação não puder se conectar ao gerenciador de filas usando a primeira entrada na lista, ela será movida para a segunda entrada, e assim por diante, até que uma conexão com o gerenciador de filas tenha sido feita ou que o final da lista tenha sido alcançado.

Se a especificação de ativação não puder se conectar ao gerenciador de filas especificado usando qualquer uma das entradas na **CONNECTIONNAMELIST**, ela será interrompida e deverá ser reiniciada.

Assim que a especificação de ativação estiver em execução, ela obterá mensagens do sistema do IBM MQ e as entregará para um MDB para processamento.

Se o gerenciador de filas falhar enquanto uma mensagem estiver sendo processada, o ambiente do Java EE detectará a falha e tentará reconectar a especificação de ativação.

A especificação de ativação usa as informações na propriedade **CONNECTIONNAMELIST** como antes, ao executar as tentativas de reconexão.

Se a especificação de ativação tentar todas as entradas no **CONNECTIONNAMELIST** e ainda não conseguir se conectar ao gerenciador de fila, a especificação de ativação aguardará o período especificado pela IBM MQ propriedade do adaptador de recursos **reconnectionRetryInterval** antes de tentar novamente.

A propriedade do adaptador de recursos do IBM MQ **reconnectionRetryCount** define o número de tentativas de reconexão consecutivas que são feitas antes de uma especificação de ativação ser interrompida e requer uma reinicialização manual

Assim que a especificação de ativação tiver sido reconectada a um sistema do IBM MQ, o ambiente do Java EE executará qualquer limpeza transacional que for necessária e continuará entregando mensagens para MDBs para processamento.

Para que a limpeza transacional funcione corretamente, o ambiente do Java EE deverá ser capaz de acessar os logs para o gerenciador de filas que falhou.

Se as especificações de ativação estiverem sendo usadas com MDBs transacionais que participam de transações XA e estiverem se conectando a um gerenciador de filas de várias instâncias, a **CONNECTIONNAMELIST** deverá conter uma entrada para ambas as instâncias do gerenciador de filas, ativa e em espera.

Isso significa que o ambiente do Java EE poderá acessar os logs do gerenciador de filas se o ambiente precisar executar a recuperação da transação, independentemente de qual gerenciador de filas o ambiente reconecta para seguir uma falha.

Se os MDBs transacionais estiverem sendo usados com gerenciadores de filas independentes, a propriedade **CONNECTIONNAMELIST** deverá conter uma única entrada, para assegurar que a especificação de ativação sempre se reconecte ao mesmo gerenciador de filas em execução no mesmo sistema após uma falha

Propriedade CCDTURL

Ao iniciar, a especificação de ativação tenta se conectar ao gerenciador de filas especificado na propriedade **QMANAGER** usando a primeira entrada na tabela de definições de canais do cliente (CCDT).

Se a especificação de ativação não puder se conectar ao gerenciador de filas usando a primeira entrada na tabela, ela será movida para a segunda entrada, e assim por diante, até que uma conexão com o gerenciador de filas tenha sido feita ou que o final da tabela tenha sido alcançado.

Se a especificação de ativação não puder se conectar ao gerenciador de filas especificado usando qualquer uma das entradas na CCDT, ela será interrompida e deverá ser reiniciada.

Assim que a especificação de ativação estiver em execução, ela obterá mensagens do sistema do IBM MQ e as entregará para um MDB para processamento.

Se o gerenciador de filas falhar enquanto uma mensagem estiver sendo processada, o ambiente do Java EE detectará a falha e tentará reconectar a especificação de ativação.

A especificação de ativação usa as informações na propriedade CCDT como antes, ao executar as tentativas de reconexão.

Se a especificação de ativação tentar todas as entradas na CCDT e ainda não conseguir se conectar ao gerenciador de filas, a especificação de ativação aguardará o período de tempo especificado pela IBM MQ propriedade do adaptador de recurso **reconnectionRetryInterval** antes de tentar novamente

A propriedade do adaptador de recursos do IBM MQ **reconnectionRetryCount** define o número de tentativas de reconexão consecutivas que são feitas antes de uma especificação de ativação ser interrompida e requer uma reinicialização manual

Assim que a especificação de ativação tiver sido reconectada a um sistema do IBM MQ, o ambiente do Java EE executará qualquer limpeza transacional que for necessária e continuará entregando mensagens para MDBs para processamento.

Para que a limpeza transacional funcione corretamente, o ambiente do Java EE deverá ser capaz de acessar os logs para o gerenciador de filas que falhou.

Se as especificações de ativação estiverem sendo usadas com MDBs transacionais que participam de transações XA e estiverem se conectando a um gerenciador de filas de várias instâncias, a CCDT deverá conter uma entrada para ambas as instâncias do gerenciador de filas, ativa e em espera.

Isso significa que o ambiente do Java EE poderá acessar os logs do gerenciador de filas se o ambiente precisar executar a recuperação da transação, independentemente de qual gerenciador de filas o ambiente reconecta para seguir uma falha.

Se os MDBs transacionais estiverem sendo usados com gerenciadores de filas independentes, a CCDT deverá conter uma única entrada para assegurar que a especificação de ativação sempre se reconecte ao mesmo gerenciador de filas em execução no mesmo sistema após uma falha.

Assegure-se de que tenha configurado o valor padrão de *PREFERRED* para a propriedade **AFFINITY** nas CCDTs, usado com especificações de ativação, para que as conexões sejam feitas com o mesmo gerenciador de filas ativo.

Propriedade CLIENTRECONNECTOPTIONS

As especificações de ativação fornecem sua própria funcionalidade de reconexão. A funcionalidade fornecida permitirá que as especificações sejam reconectadas automaticamente a um sistema IBM MQ se o gerenciador de filas ao qual elas estavam conectadas falhar.

Por causa disso, a funcionalidade de reconexão automática do cliente fornecida pelo IBM MQ classes for JMS não é suportada.

Deve-se configurar a propriedade **CLIENTRECONNECTOPTIONS** como *DISABLED* para todas as especificações de ativação usadas no Java EE.

Portas do listener do WebSphere Application Server
Aplicativos bean acionados por mensagens (MDB) instalados no WebSphere Application Server também podem usar portas do listener para processar mensagens em um sistema IBM MQ.

As portas do listener são usadas para detectar mensagens que chegam em um sistema IBM MQ e, em seguida, entregá-las aos MDBs para processamento. Este tópico explica como a porta do listener monitora o sistema IBM MQ.

Os MDBs também podem fazer conexões adicionais com sistemas do IBM MQ de dentro de seu método `onMessage()`.

Consulte "Enterprise JavaBeans e aplicativos baseados na web" na página 303 para obter mais informações sobre como essas conexões podem usar a reconexão automática do cliente

Para as portas do listener WebSphere Application Server:

- **CONNECTIONNAMELIST** e **CCDTURL** são suportados com restrições
- **CLIENTRECONNECTOPTIONS** não é suportado

Propriedade CONNECTIONNAMELIST

As portas do listener usam conjuntos de conexões JMS ao se conectarem ao IBM MQ, portanto, estão sujeitas às implicações do uso de conjuntos de conexões. Consulte "Especificações de ativação" na página 299 para obter informações adicionais.

Se não houver conexões livres e o número máximo de conexões ainda não tiver sido criado por meio desse connection factory, a propriedade **CONNECTIONNAMELIST** será usada para tentar e criar uma nova conexão com o IBM MQ.

Se todos os sistemas IBM MQ na **CONNECTIONNAMELIST** não estiverem acessíveis, a porta do listener parará.

A porta listener, então, espera a propriedade customizada do serviço de listener de mensagens **RECOVERY.RETRY.INTERVAL** pelo tempo especificado e tenta se reconectar novamente.

Essa tentativa de reconexão verificará se há alguma conexão livre no conjunto de conexões, apenas no caso de uma ser retornada entre as tentativas de conexão. Se não houver uma disponível, a porta do listener usará a **CONNECTIONNAMELIST** como antes.

Assim que a porta do listener tiver sido reconectada a um sistema do IBM MQ, o ambiente do Java EE executará qualquer limpeza transacional que for necessária e, em seguida, continuará entregando mensagens para MDBs para processamento.

Para que a limpeza transacional funcione corretamente, o ambiente do Java EE deverá ser capaz de acessar os logs para o gerenciador de filas que falhou.

Se as portas do listener estiverem sendo usadas com MDBs transacionais que participam de transações XA e estiverem se conectando a um **gerenciador de filas de várias instâncias**, a **CONNECTIONNAMELIST** deverá conter uma entrada para ambas as instâncias do gerenciador de filas, ativa e em espera.

Isso significa que o ambiente do Java EE poderá acessar os logs do gerenciador de filas se o ambiente precisar executar a recuperação da transação, independentemente de qual gerenciador de filas o ambiente reconecta para seguir uma falha.

Se os MDBs transacionais estiverem sendo usados com gerenciadores de filas independentes, a propriedade **CONNECTIONNAMELIST** deverá conter uma única entrada, para assegurar que a especificação de ativação sempre se reconecte ao mesmo gerenciador de filas em execução no mesmo sistema após uma falha

Propriedade CCDTURL

Ao iniciar, a porta listener tenta se conectar ao gerenciador de filas especificado na propriedade **QMANAGER** usando a primeira entrada na CCDT.

Se a porta do listener não for capaz de se conectar ao gerenciador de filas usando a primeira entrada na tabela, ela será movida para a segunda entrada, e assim por diante, até que uma conexão com o gerenciador de filas tenha sido feita ou que o final da tabela tenha sido atingido.

Se a porta do listener não puder se conectar ao gerenciador de filas especificado usando nenhuma das entradas na CCDT, a porta do listener parará.

A porta listener, então, espera a propriedade customizada do serviço de listener de mensagens **RECOVERY.RETRY.INTERVAL** pelo tempo especificado e tenta se reconectar novamente.

Essa tentativa de reconexão funciona da sua maneira em todas as entradas na CCDT como antes.

Depois que a Porta do listener estiver em execução, ela obterá mensagens do sistema IBM MQ e as entregará a um MDB para processamento.

Se o gerenciador de filas falhar enquanto uma mensagem estiver sendo processada, o ambiente Java EE detectará a falha e tentará reconectar a porta do listener. A porta listener usa as informações na CCDT ao executar as tentativas de reconexão.

Se a porta do listener tentar todas as entradas na CCDT e ainda não puder se conectar ao gerenciador de filas, ela aguardará o período de tempo especificado pela propriedade **RECOVERY.RETRY.INTERVAL** antes de tentar novamente.

A propriedade **MAX.RECOVERY.RETRIES** do serviço de listener de mensagens define o número de tentativas de reconexão consecutivas que são feitas antes que uma porta do listener pare e requeira uma reinicialização manual.

Assim que a porta do listener tiver sido reconectada a um sistema do IBM MQ, o ambiente do Java EE executará qualquer limpeza transacional que for necessária e, em seguida, continuará entregando mensagens para MDBs para processamento.

Para que a limpeza transacional funcione corretamente, o ambiente do Java EE deverá ser capaz de acessar os logs para o gerenciador de filas que falhou.

Se as portas do listener estiverem sendo usadas com MDBs transacionais que participam de transações XA e estiverem se conectando a um gerenciador de filas de várias instâncias, a CCDT deverá conter uma entrada para ambas as instâncias do gerenciador de filas, ativa e em espera.

Isso significa que o ambiente do Java EE poderá acessar os logs do gerenciador de filas se o ambiente precisar executar a recuperação da transação, independentemente de qual gerenciador de filas o ambiente reconecta para seguir uma falha.

Se os MDBs transacionais estiverem sendo usados com gerenciadores de filas independentes, a CCDT deverá conter uma única entrada para assegurar que a porta do listener sempre se reconecte ao mesmo gerenciador de filas em execução no mesmo sistema após uma falha.

Assegure-se de que tenha configurado o valor padrão de *PREFERRED* para a propriedade **AFFINITY** nas CCDTs, usado com portas do listener, para que as conexões sejam feitas no mesmo gerenciador de filas ativo.

Propriedade CLIENTRECONNECTOPTIONS

As portas do listener fornecem sua própria funcionalidade de reconexão. A funcionalidade fornecida permitirá que as portas do listener sejam reconectadas automaticamente a um sistema IBM MQ se o gerenciador de filas ao qual elas estavam conectadas falhar.

Por causa disso, a funcionalidade de reconexão automática do cliente fornecida pelo IBM MQ classes for JMS não é suportada.

Deve-se configurar a propriedade **CLIENTRECONNECTOPTIONS** como *DISABLED* para todas as portas do listener usadas no Java EE.

Enterprise JavaBeans e aplicativos baseados na web
Aplicativos Enterprise JavaBean (EJB) e aplicativos executados dentro do contêiner da web, como Servlets, usam um connection factory do JMS para criar uma conexão com um gerenciador de filas do IBM MQ.

As restrições a seguir se aplicam aos EJBs e aplicativos baseados na web:

- **CONNECTIONNAMELIST** e **CCDTURL** são suportados com restrições
- **CLIENTRECONNECTOPTIONS** não é suportado

Propriedade CONNECTIONNAMELIST

Se o ambiente Java EE fornecer um conjunto de conexões para conexões JMS, consulte "Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões" na página 304 para obter informações sobre como isso afeta o comportamento da propriedade **CONNECTIONNAMELIST**.

Se o ambiente Java EE não fornecer um conjunto de conexões JMS, o aplicativo usará a propriedade **CONNECTIONNAMELIST** da mesma maneira que os aplicativos Java SE.

Se os aplicativos estiverem sendo usados com MDBs transacionais que participam de transações XA e estiverem se conectando a um gerenciador de filas multi-instância, a **CONNECTIONNAMELIST** deverá conter uma entrada para ambas as instâncias do gerenciador de filas, ativa e em espera.

Isso significa que o ambiente do Java EE poderá acessar os logs do gerenciador de filas se o ambiente precisar executar a recuperação da transação, independentemente de qual gerenciador de filas o ambiente reconecta para seguir uma falha.

Se os aplicativos estiverem sendo usados com os gerenciadores de filas independentes, a propriedade **CONNECTIONNAMELIST** deverá conter uma entrada única, para assegurar que o aplicativo sempre se reconecte ao mesmo gerenciador de filas, em execução no mesmo sistema, após uma falha

Propriedade CCDTURL

Se o ambiente Java EE fornecer um conjunto de conexões para conexões JMS, consulte "Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões" na página 304 para obter informações sobre como isso afetará o comportamento da propriedade **CCDTURL**.

Se o ambiente Java EE não fornecer um conjunto de conexões JMS, o aplicativo usará a propriedade **CCDTURL** da mesma maneira que os aplicativos Java SE.

Se os aplicativos estiverem sendo usados com MDBs transacionais que participam de transações XA e estiverem se conectando a um gerenciador de filas de várias instâncias, a CCDT deverá conter uma entrada para ambas as instâncias do gerenciador de filas, ativa e em espera.

Isso significa que o ambiente do Java EE poderá acessar os logs do gerenciador de filas se o ambiente precisar executar a recuperação da transação, independentemente de qual gerenciador de filas o ambiente reconecta para seguir uma falha.

Se os aplicativos estiverem sendo usados com gerenciadores de filas independentes, a CCDT deverá conter uma única entrada para assegurar que a especificação de ativação sempre se reconecte ao mesmo gerenciador de filas em execução no mesmo sistema após uma falha.

Propriedade CLIENTRECONNECTOPTIONS

Deve-se configurar a propriedade **CLIENTRECONNECTOPTIONS** como *DISABLED* para todos os connection factories do JMS usados pelos EJBs ou aplicativos executados no contêiner da web.

Os aplicativos que precisarem se reconectar automaticamente a um novo gerenciador de filas, se o gerenciador de filas que eles estão usando falhar, precisarão implementar sua própria lógica de reconexão. Consulte "Implementando a lógica de reconexão em um aplicativo Java EE" na página 305 para obter mais informações.

Cenários: WebSphere Application Server com o IBM MQ

Cenários: perfil Liberty do WebSphere Application Server com o IBM MQ

Os aplicativos em execução dentro de contêineres do cliente
Alguns ambientes Java EE, como o WebSphere Application Server, fornecem um contêiner do cliente que pode ser usado para executar aplicativos Java SE.

Aplicativos em execução nesses ambientes usam um connection factory JMS para se conectarem a um gerenciador de filas do IBM MQ.

Para aplicativos em execução dentro de contêineres do cliente:

- **CONNECTIONNAMELIST** e **CCDTURL** são totalmente suportados
- **CLIENTRECONNECTOPTIONS** é totalmente suportado

Propriedade CONNECTIONNAMELIST

Se o ambiente Java EE fornecer um conjunto de conexões para conexões JMS, consulte “Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões” na página 304 para obter informações sobre como isso afeta o comportamento da propriedade **CONNECTIONNAMELIST**.

Se o ambiente Java EE não fornecer um conjunto de conexões JMS, o aplicativo usará a propriedade **CONNECTIONNAMELIST** da mesma maneira que os aplicativos Java SE.

Propriedade CCDTURL

Se o ambiente Java EE fornecer um conjunto de conexões para conexões JMS, consulte “Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões” na página 304 para obter informações sobre como isso afetará o comportamento da propriedade **CCDTURL**.

Se o ambiente Java EE não fornecer um conjunto de conexões JMS, o aplicativo usará a propriedade **CCDTURL** da mesma maneira que os aplicativos Java SE.

Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões
Alguns ambientes do Java EE, por exemplo, WebSphere Application Server, fornecem um conjunto de conexões do JMS. contêiner que pode ser usado para executar aplicativos Java SE.

Os aplicativos que criam uma conexão usando um connection factory que foi definido no ambiente Java EE obterão uma conexão livre existente do conjunto de conexões para esse connection factory ou uma nova conexão, se não houver uma adequada no conjunto de conexões.

Isso poderá ter implicações se o connection factory tiver sido configurado com a propriedade **CONNECTIONNAMELIST** ou **CCDTURL** definida.

Na primeira vez que o connection factory é usado para criar uma conexão, o ambiente Java EE usa a **CONNECTIONNAMELIST** ou o **CCDTURL** para criar uma nova conexão com o sistema IBM MQ. Quando essa conexão não for mais necessária, ela será retornada para o conjunto de conexões no qual a conexão se torna disponível para reutilização.

Se algo mais criar uma conexão por meio do connection factory, o ambiente Java EE retornará a conexão por meio do conjunto de conexões, em vez de usar as propriedades **CONNECTIONNAMELIST** ou **CCDTURL** para criar uma nova conexão.

Se uma conexão estiver sendo usada quando uma instância do gerenciador de filas falhar, ela será descartada. No entanto, o conteúdo do conjunto de conexões pode não estar, o que significa que o conjunto pode potencialmente ainda conter conexões com um gerenciador de filas que não está mais em execução.

Nesta situação, na próxima vez que uma solicitação for feita para criar uma conexão por meio do connection factory, uma conexão com o gerenciador de filas com falha será retornada. Todas as tentativas de usar essa conexão falham, já que o gerenciador de filas não está mais em execução, fazendo com que a conexão seja descartada.

Somente quando o conjunto de conexões estiver vazio, o ambiente Java EE usará as propriedades **CONNECTIONNAMELIST** ou **CCDTURL** para criar uma nova conexão com o IBM MQ.

Devido à maneira com que a **CONNECTIONNAMELIST** e os CCDTs são usados para criar conexões JMS, também é possível ter um conjunto de conexões contendo conexões para diferentes sistemas IBM MQ.

Por exemplo, suponha que um connection factory tenha sido configurado com a propriedade **CONNECTIONNAMELIST** configurada com o valor a seguir:

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Suponha que a primeira vez que um aplicativo tentar criar uma conexão com um gerenciador de filas independente por meio desse connection factory, o gerenciador de filas em execução no sistema `hostname1(port1)` não esteja acessível. Isso significa que o aplicativo termina com uma conexão com o gerenciador de filas em execução no `hostname2(port2)`.

Outro aplicativo agora é fornecido e cria uma conexão JMS por meio do mesmo connection factory. O gerenciador de filas no `hostname1(port1)` agora está disponível, portanto, uma nova conexão JMS é criada para esse sistema IBM MQ e é retornada para o aplicativo.

Quando ambos os aplicativos tiverem sido concluídos, eles fecharão suas Conexões JMS, fazendo com que as conexões sejam retornadas para o conjunto de conexões.

O resultado é que o conjunto de conexões para nosso connection factory agora contém duas conexões JMS:

- Uma conexão com o gerenciador de filas em execução no `hostname1(port1)`
- Uma conexão com o gerenciador de filas em execução no `hostname2(port2)`

Isso pode levar a problemas relacionados à recuperação da transação. Se o sistema Java EE precisar recuperar uma transação, ele precisará ser capaz de se conectar a um gerenciador de filas que tenha acesso aos logs de transações.

Implementando a lógica de reconexão em um aplicativo Java EE
O Enterprise JavaBeans e aplicativos baseados na web que desejam se reconectar automaticamente se um gerenciador de filas falhar a necessidade de implementar sua própria lógica de reconexão.

As opções a seguir fornecem mais informações sobre como isso pode ser obtido.

Permitir que o aplicativo falhe

Essa abordagem não requer mudanças no aplicativo, mas requer uma reconfiguração administrativa da definição de connection factory para incluir a propriedade **CONNECTIONNAMELIST**. No entanto, essa abordagem requer que o invocador seja capaz de manipular uma falha apropriadamente. Observe que isso também é necessário para falhas como MQRC_Q_FULL, que não estão relacionadas à falha na conexão.

Código de exemplo para este processo:

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
          try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
```

```
  catch (JMSException je) {
  // process exception
    }
  }
}
```

O código anterior assume que o connection factory que este servlet está usando tem a propriedade **CONNECTIONNAMELIST** definida.

Quando o servlet é processado pela primeira vez, uma nova conexão é criada usando a propriedade **CONNECTIONNAMELIST** , supondo que nenhuma conexão em conjunto esteja disponível a partir de outros aplicativos que se conectam ao mesmo gerenciador de filas

Quando a conexão é liberada após uma chamada `close()`, ela é retornada para o conjunto e reutilizada na próxima vez que o servlet é executado, sem se referir a **CONNECTIONNAMELIST**, até que ocorra uma falha na conexão, em cujo ponto um evento CONNECTION_ERROR_OCCURRED é gerado. Esse evento solicita que o conjunto destrua a conexão com falha.

Quando o aplicativo for executado na próxima vez, nenhuma conexão agrupada estará disponível e a **CONNECTIONNAMELIST** será usada para se conectar ao primeiro gerenciador de filas disponível. Se tiver ocorrido failover do gerenciador de filas (por exemplo, não era uma falha transitória da rede), o servlet se conectará à instância de backup assim que estiver disponível.

Se outros recursos, como bancos de dados, estiverem envolvidos no aplicativo, talvez seja apropriado indicar que o servidor de aplicativos deve recuperar a transação.

Manipular a reconexão dentro do aplicativo

Se o invocador não puder processar uma falha por meio do servlet, a reconexão deverá ser manipulada dentro do aplicativo. Conforme mostrado no exemplo a seguir, manipular uma reconexão dentro do aplicativo requer que o aplicativo solicite uma nova conexão para que possa armazenar em cache o connection factory que ele consultou na JNDI e manipular uma `JMSException` como JMSCMQ0001: `A chamada do WebSphere MQ falhou com o compcode '2' ('MQCC_FAILED') razão '2009' ('MQRC_CONNECTION_BROKEN')`.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
        setupResources();
    }
  }
```

No exemplo a seguir, `setupResources()` cria os objetos do JMS e inclui um loop de suspensão e nova tentativa para manipular a reconexão não instantânea. Na prática, esse método evita muitas tentativas de reconexão. Observe que as condições de saída foram omitidas no exemplo para clareza.

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
```

```
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

Se o aplicativo gerenciar a reconexão, será importante que ele libere quaisquer conexões mantidas para outros recursos, se esses recursos forem outros gerenciadores de filas do IBM MQ ou outros serviços de backend, como bancos de dados. Deve-se restabelecer essas conexões quando a reconexão com uma nova instância do gerenciador de filas do IBM MQ estiver concluída. Se você não restabelecer as conexões, os recursos do servidor de aplicativos serão mantidos desnecessariamente durante a tentativa de reconexão e poderão atingir o tempo limite pelo tempo que foram reutilizados.

Uso do WorkManager

Para aplicativos de longa duração (por exemplo, processamento em lote), em que o tempo de processamento é maior que algumas dezenas de segundos, o WorkManager do WebSphere Application Server pode ser usado. Um exemplo de fragmento de código para WebSphere Application Server segue:

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

onde `web.xml` contém:

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

e o lote agora é implementado por meio da interface de trabalho:

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
```

```
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
               ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

Se o processamento em lote demorar muito tempo para ser executado, por exemplo, devido a mensagens grandes, rede lenta ou acesso extensivo ao banco de dados (especialmente quando acoplado ao failover lento), o servidor começará a produzir saída de avisos de encadeamento interrompidos, semelhante ao exemplo a seguir:

WSVR0605W: O encadeamento "`WorkManager.DefaultWorkManager` : 0" (00000035) está ativo por 694.061 milissegundos e pode ser interrompido Há 1 encadeamento no total no servidor que pode ser interrompido.

Esses avisos podem ser minimizados reduzindo o tamanho do lote ou aumentando o tempo limite de encadeamento interrompido. No entanto, geralmente será preferível implementar esse processamento em um processamento de bean EJB (para envio em lote) ou de bean acionado por mensagens (para consumir ou consumir e responder).

Observe que a reconexão gerenciada pelo aplicativo não fornece uma solução geral para manipular erros de tempo de execução e o aplicativo ainda deve manipular erros que não estão relacionados à falha de conexão.

Por exemplo, tentando colocar uma mensagem em uma fila que está cheia (2053 MQRC_Q_FULL) ou tentando se conectar a um gerenciador de filas usando credenciais de segurança que não são válidas (2035 MQRC_NOT_AUTHORIZED).

O aplicativo também deverá manipular erros 2059 MQRC_Q_MGR_NOT_AVAILABLE quando nenhuma instância estiver imediatamente disponível quando o failover estiver em andamento. Isso pode ser alcançado pelo aplicativo relatando as exceções JMS conforme elas ocorrem, em vez de tentar se reconectar silenciosamente.

Conjunto de Objetos do IBM MQ classes for JMS
Usar um formulário de definição do conjunto de conexões fora do Java EE ajuda a reduzir a carga geral resultante, por exemplo, de alguns aplicativos independentes que usam estruturas ou que são implementados em ambientes de nuvem e também de um número maior de conexões de clientes nos `QueueManagers`, levando a um aumento na consolidação do servidor de aplicativos e gerenciadores de filas

No modelo de programação Java EE, há um ciclo de vida bem definido dos vários objetos em uso. Os beans acionados por mensagens (MDBs) são mais restrito, enquanto Servlets fornecem mais liberdade. Portanto, as opções de definição do conjunto disponíveis nos servidores Java EE se adaptam a vários modelos de programação usados.

Com o Java SE (ou com outra estrutura, como Spring) os modelos de programação são extremamente flexíveis. Portanto, uma estratégia de conjunto único não atende todos. Será necessário considerar, se

estiver indo para uma estrutura no local que possa fazer qualquer forma de conjuntos, por exemplo, Spring.

A estratégia de definição de conjunto a ser usada depende do ambiente no qual seu aplicativo está em execução.

O conjunto de objetos em um ambiente do Java EE
Os servidores de aplicativos do Java EE fornecem a funcionalidade de conjunto de conexão que pode ser usada por aplicativos de bean acionados por mensagens, Enterprise Java Beans e Servlets.

O WebSphere Application Server mantém um conjunto de conexões para um provedor JMS, para melhorar o desempenho. Quando um aplicativo cria uma conexão JMS, o servidor de aplicativos determina se já existe uma conexão no conjunto de conexões livres. Se existir, a conexão será retornada ao aplicativo, caso contrário, uma nova conexão será criada.

A Figura 41 na página 309 mostra como as especificações de ativação e as portas listener estabelecem uma conexão JMS e usam essa conexão para monitorar um destino para mensagens no modo normal.

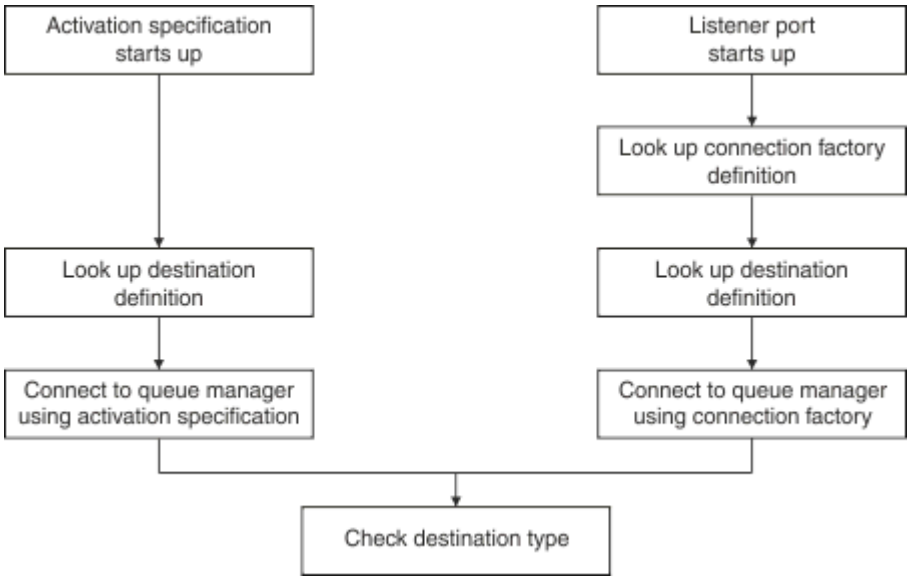


Figura 41. Modo normal

Ao usar o provedor de sistemas de mensagens IBM MQ , os aplicativos que executam o sistema de mensagens de saída (como enterprise Java Beans e servlets) e o componente da porta do listener do bean acionado por mensagens podem usar esses conjuntos de conexões..

As especificações de ativação do provedor de sistemas de mensagens do IBM MQ usam a funcionalidade de definição do conjunto de conexões fornecida pelo adaptador de recursos do IBM MQ. Consulte Configurando propriedades para o adaptador de recursos do WebSphere MQ para obter mais informações.

“Exemplos de uso do conjunto de conexões” na página 313 explica como aplicativos que executam sistemas de mensagens não enviadas e portas do listener usam o conjunto livre ao criar conexões JMS.

“Encadeamentos de manutenção do conjunto de” na página 316 explica o que acontece a essas conexões quando um aplicativo ou porta do listener é concluída com as conexões.

“Exemplos de encadeamento de manutenção do conjunto” na página 317 explica como o conjunto de conexões livres é limpo para evitar que conexões JMS se tornem antigas.

O WebSphere Application Server tem um limite no número de conexões que podem ser criadas a partir de um factory, especificado pela propriedade *maximum connections* do Connection Factory. O valor padrão para essa propriedade é 10, o que significa que pode haver até 10 conexões criadas a partir de um factory, em um determinado momento.

Cada factory tem um conjunto de conexões livres associado. Quando o servidor de aplicativos é inicializado, os conjuntos de conexões livres estão vazios. O número máximo de conexões que pode existir no conjunto livre para um factory também é especificado pela propriedade Máximo de conexões.

Sugestão: Com o JMS 2.0, um connection factory pode ser usado para criar conexões e contextos. Como resultado, é possível ter um conjunto de conexões associado a um connection factory que contém uma combinação de conexões e contextos. É recomendado que um connection factory seja usado somente para criar conexões ou contextos. Isso assegura que o conjunto de conexões para esse connection factory só contenha objetos de um único tipo, o que torna o conjunto mais eficiente.

Para obter informações sobre como a definição do conjunto de conexões funciona no WebSphere Application Server, consulte Configurando a definição do conjunto de conexões para conexões do JMS. Para outros servidores de aplicativos, consulte a documentação apropriada do servidor de aplicativos.

Como o conjunto de conexões é usado

Cada connection factory do JMS tem um conjunto de conexões associado a ele e esse conjunto contém zero ou mais conexões JMS. Cada conexão JMS tem um conjunto de sessões JMS associado e cada conjunto de sessões JMS contém zero ou mais sessões JMS.

Figura 42 na página 310 mostra o relacionamento entre esses objetos.

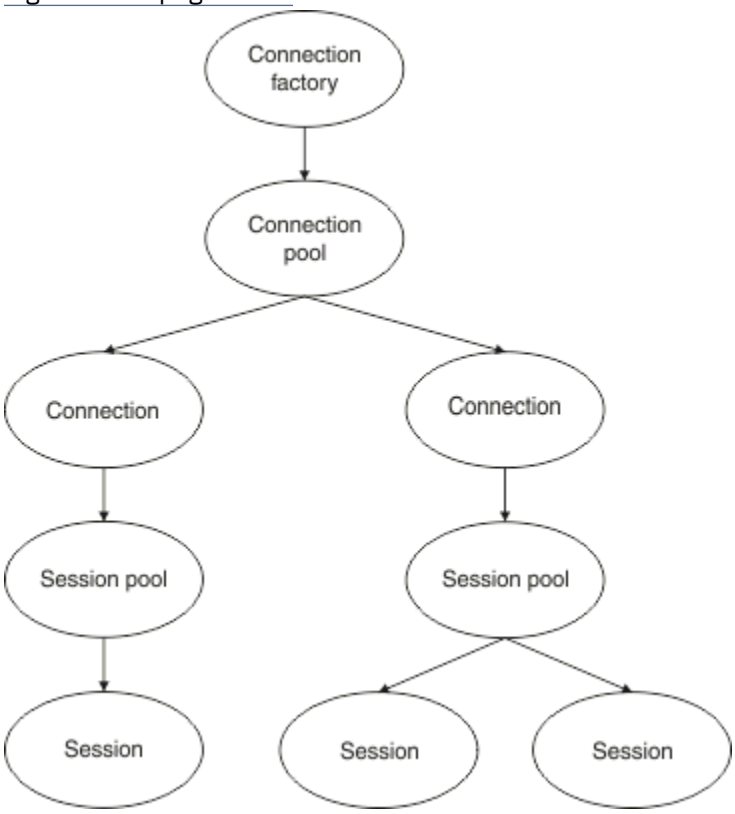


Figura 42. Conjuntos de conexões e conjuntos de sessões

Quando uma porta do listener é inicializada ou um aplicativo que deseja executar o sistema de mensagens não enviadas usa o factory para criar uma conexão, a porta ou o aplicativo chama um dos métodos a seguir:

- **`connectionFactory.createConnection()`**
- **`ConnectionFactory.createConnection(String, String)`**
- **`QueueConnectionFactory.createQueueConnection()`**
- **`QueueConnectionFactory.createQueueConnection(String, String)`**
- **`TopicConnectionFactory.createTopicConnection()`**

- **TopicConnectionFactory.createTopicConnection(String, String)**

O gerenciador de conexões do WebSphere Application Server tenta obter uma conexão do conjunto livre para este factory e retorná-lo para o aplicativo.

Se não houver conexões livres no conjunto e o número de conexões criadas a partir desse factory não tiver atingido o limite especificado na propriedade *maximum connections* desse factory, o Connection Manager criará uma nova conexão para o aplicativo usar.

No entanto, se um aplicativo tentar criar uma conexão, mas o número de conexões criadas a partir desse factory já for igual à propriedade *maximum connections* do factory, o aplicativo aguardará que uma conexão seja disponibilizada (para ser colocada de volta no conjunto livre).

O tempo que o aplicativo aguarda é especificado na propriedade *connection timeout* do conjunto de conexões, que possui um valor padrão de 180 segundos. Se uma conexão for colocada de volta no conjunto livre dentro desse período de 180 segundos, o Connection Manager imediatamente o retirará do conjunto novamente e o passará para o aplicativo. No entanto, se o período de tempo limite passar, uma *ConnectionWaitTimeoutException* será lançada.

Quando um aplicativo tiver concluído a conexão e fechá-la chamando:

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

a conexão será realmente mantida aberta e retornada ao conjunto livre para que possa ser reutilizada por outro aplicativo. Portanto, será possível ter conexões abertas entre o WebSphere Application Server e o fornecedor JMS, mesmo se nenhum aplicativo JMS estiver em execução no servidor de aplicativos.

Propriedades Avançadas do Conjunto de Conexão
Há várias propriedades avançadas que podem ser usadas para controlar o comportamento dos conjuntos de conexões do JMS.

Proteção de segurança

"Como os aplicativos que executam o sistema de mensagens de saída usam o conjunto de conexões" na página 315 descreve o uso do método `sendMessage()`, que incorpora `connectionFactory.createConnection()`.

Considere a situação em que você tem 50 EJBs que criam conexões JMS por meio do mesmo connection factory como parte de seu método `ejbCreate()`.

Se todos esses beans forem criados ao mesmo tempo e não houver conexões no conjunto de conexões livres do factory, o servidor de aplicativos tentará criar 50 conexões JMS para o mesmo provedor JMS simultaneamente. O resultado é um carregamento significativo tanto para o WebSphere Application Server quanto para o provedor JMS.

As propriedades de estabilização podem evitar essa situação limitando o número de conexões JMS que podem ser criadas por meio de um connection factory em um determinado momento e escalonando a criação de conexões adicionais.

A limitação do número de conexões JMS em um determinado momento é obtida usando duas propriedades:

- Limite de Surge
- Intervalo de criação de Surge.

Quando os aplicativos EJB tentam criar uma conexão JMS por meio de um connection factory, o gerenciador de conexões verifica quantas conexões estão sendo criadas. Se esse número for menor ou igual ao valor da propriedade `surge threshold` , o gerenciador de conexão continuará abrindo novas conexões.

No entanto, se o número de conexões que estão sendo criadas exceder a propriedade `surge threshold` , o gerenciador de conexões aguardará o período especificado pela propriedade `surge creation interval` antes de criar e abrir uma nova conexão.

Conexões de Stuck

Uma conexão JMS será considerada `stuck`, se um aplicativo JMS usar essa conexão para enviar uma solicitação para o provedor JMS e o provedor não responder dentro de um determinado período de tempo.

O WebSphere Application Server fornece uma maneira de detectar conexões `stuck` do JMS. Para usar essa função, deve-se configurar três propriedades:

- Cronômetro de Tempo de Stuck
- Hora do Stuck
- Limite de Stuck

"Exemplos de encadeamento de manutenção do conjunto" na página 317 explica como o encadeamento de manutenção do conjunto é executado periodicamente e verifica o conteúdo do conjunto livre de um connection factory, procurando conexões que não foram usadas por um período de tempo ou que existem há muito tempo.

Para detectar conexões presas, o servidor de aplicativos também gerencia um encadeamento de conexões presas que verifica o estado de todas as conexões ativas criadas por meio de um connection factory para ver se alguma delas está aguardando uma resposta do provedor JMS.

A execução do encadeamento de conexões presas é determinada pela propriedade `Stuck time timer`. O valor padrão para essa propriedade é zero, o que significa que a detecção de conexão presa nunca é executada.

Se o encadeamento localizar uma aguardando uma resposta, ele determinará há quanto tempo ela está esperando e comparará esse tempo com o valor da propriedade `Stuck time`.

Se o tempo gasto para o provedor JMS responder exceder o tempo especificado pela propriedade `Stuck time`, o servidor de aplicativos marcará a conexão JMS como presa.

Por exemplo, suponha que a connection factory `jms/CF1` tenha a propriedade `Stuck time timer` configurada como 10 e a propriedade `Stuck time` configurada como 15

O encadeamento de conexões presas fica ativo a cada 10 segundos e verifica se alguma conexão criada por meio de `jms/CF1` está esperando por uma resposta do IBM MQ há mais de 15 segundos.

Suponha que um EJB crie uma conexão JMS com o IBM MQ usando `jms/CF1` e, em seguida, tente criar uma Sessão JMS usando essa conexão chamando `Connection.createSession()`.

No entanto, algo está evitando que o provedor JMS responda à solicitação. Talvez a máquina tenha congelado ou um processo em execução no provedor JMS esteja bloqueado, evitando que qualquer novo trabalho seja processado:

Dez segundos após o EJB chamado `Connection.createSession()`, o cronômetro de conexão presa fica ativo e examina as conexões ativas criadas por meio de `jms/CF1`.

Suponha que haja apenas uma conexão ativa, por exemplo, chamada `c1`. O primeiro EJB está esperando 10 segundos por uma resposta a uma solicitação que enviou para `c1`, que é menor que o valor de `Stuck time`, portanto, o cronômetro de conexão presa ignora essa conexão e fica inativo.

10 segundos mais tarde, o encadeamento de conexões presas fica ativo novamente e examina as conexões ativas para `jms/CF1`. Como antes, suponha que há apenas uma conexão, `c1`.

Agora são 20 segundos desde que o primeiro EJB chamou `createSession()` e o EJB ainda está aguardando uma resposta. 20 segundos é mais longo do que o tempo especificado na propriedade `Stuck time` , portanto, o encadeamento de conexão preso marca `c1` como preso.

Se, cinco segundos depois, o IBM MQ finalmente responder e permitir que o primeiro EJB crie uma Sessão JMS, a conexão estará de volta em uso.

O servidor de aplicativos conta o número de conexões JMS criadas por meio de um connection factory que estão presas. Quando um aplicativo usa esse connection factory para criar uma nova Conexão JMS e não há conexões livres no conjunto livre desse factory, o gerenciador de conexões compara o número de conexões presas com o valor da propriedade `Stuck threshold`.

Se o número de conexões presas for menor do que o valor configurado para a propriedade `Stuck threshold`, o gerenciador de conexões criará uma nova conexão e a fornecerá ao aplicativo.

No entanto, se o número de conexões presas for igual ao valor da propriedade `Stuck threshold` , o aplicativo obterá uma exceção de recurso.

Partições do Conjunto

O WebSphere Application Server fornece duas propriedades que permitem particionar o conjunto de conexões livres para um connection factory:

- `Number of free pool partitions` informa ao servidor de aplicativos em quantas partições você deseja dividir o conjunto de conexões livres.
- `Free pool distribution table size` determina como as partições são indexadas.

Deixe essas propriedades em seus valores padrão de zero, a menos que você seja solicitado a mudá-los pelo Centro de suporte da IBM.

Observe que o WebSphere Application Server possui uma propriedade de conjunto de conexões avançada adicional chamada `Number of shared partitions`. Essa propriedade especifica o número de partições usadas para armazenar conexões compartilhadas. No entanto, como as conexões JMS são sempre não compartilhadas, essa propriedade não se aplica.

Exemplos de uso do conjunto de conexões
O componente de porta do listener do bean acionado por mensagens e os aplicativos que executam o sistema de mensagens de saída usam um conjunto de conexões do JMS.

O Figura 43 na página 313 mostra como o conjunto de conexões funciona para o WebSphere Application Server V7.5 e V8.0.

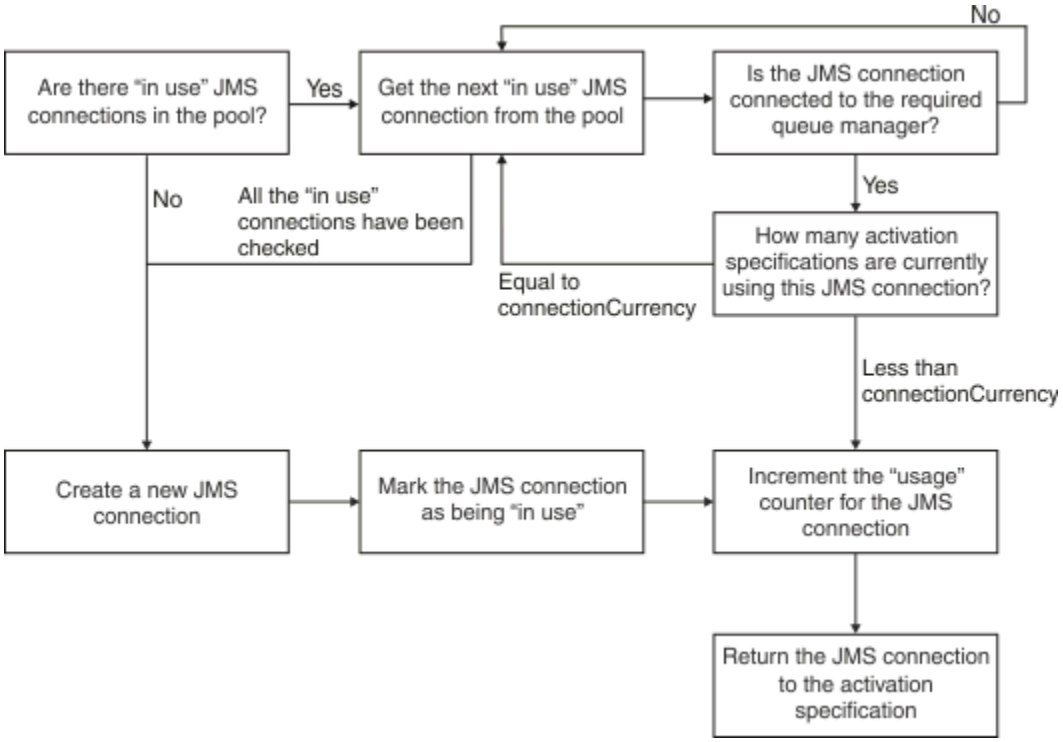


Figura 43. WebSphere Application Server V7.5 e V8.0 - como o conjunto de conexões funciona

O Figura 44 na página 314 mostra como o conjunto de conexões funciona para o WebSphere Application Server V8.5.

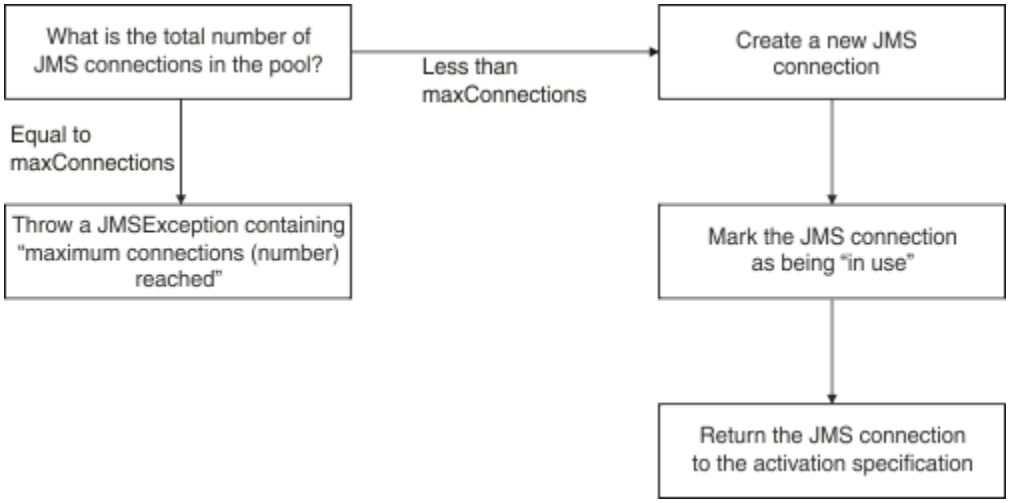


Figura 44. WebSphere Application Server V8.5 - como o conjunto de conexões funciona

Como as portas do listener do MDB usam o conjunto de conexões

Suponha que você tenha um MDB implementado em um sistema WebSphere Application Server Network Deployment que está usando o IBM MQ como o provedor JMS. O MDB é implementado em uma porta do listener que está usando um connection factory chamado, por exemplo, `jms/CF1`, que tem a propriedade *maximum connections* configurada como 2, o que significa que apenas duas conexões podem ser criadas por meio desse factory em um determinado momento.

Quando a porta do listener for inicializada, a porta tentará criar uma conexão com o IBM MQ usando o connection factory `jms/CF1`.

Para fazer isso, a porta solicita uma conexão do gerenciador de conexões. Como esta é a primeira vez que o connection factory `jms/CF1` foi usado, não há conexões no conjunto de conexões livres `jms/CF1`, portanto, o gerenciador de conexões cria uma nova conexão chamada, por exemplo, `c1`. Observe que essa conexão existirá durante toda a vida da porta do listener.

Agora, considere a situação em que você para a porta do listener usando o console administrativo do WebSphere Application Server. Nesse caso, o gerenciador de conexões pega a conexão e a coloca de volta no conjunto livre. No entanto, a conexão com o IBM MQ permanece aberta.

Se você reiniciar a porta do listener, a porta mais uma vez solicitará ao gerenciador de conexões uma conexão com o gerenciador de filas. Como você agora possui uma conexão (`c1`) no conjunto livre, o gerenciador de conexões retira essa conexão do conjunto e a disponibiliza para a porta do listener.

Agora, suponha que você tenha um segundo MDB implementado no servidor de aplicativos e que ele esteja usando uma porta de listener diferente.

Suponha que, em seguida, você tente iniciar uma terceira porta do listener, que também esteja configurada para usar o connection factory `jms/CF1`. A terceira porta do listener solicita uma conexão do gerenciador de conexões, que procura no conjunto livre por `jms/CF1` e descobre que ele está vazio. Ele então verifica quantas conexões já foram criadas por meio do factory `jms/CF1`.

Como a propriedade de conexões máximas para `jms/CF1` está configurada como 2 e você já criou duas conexões por meio desse factory, o gerenciador de conexões esperará 180 segundos (o valor padrão da propriedade de tempo limite de conexão) para uma conexão ser disponibilizada.

No entanto, se você parar a primeira porta do listener, sua conexão `c1` será colocada no conjunto livre para `jms/CF1`. O gerenciador de conexões recupera essa conexão e fornece-a para o terceiro listener.

Se você agora tentar reiniciar o primeiro listener, esse listener terá que esperar que uma das outras portas do listener seja parada antes que o primeiro listener possa reiniciar. Se nenhuma das portas

do listener em execução for interrompida em 180 segundos, o primeiro listener receberá um erro `ConnectionWaitTimeoutException` e parará.

Como os aplicativos que executam o sistema de mensagens de saída usam o conjunto de conexões

Para essa opção, suponha que haja um único EJB chamado, por exemplo, `EJB1`, instalado no servidor de aplicativos. O bean implementa um método chamado `sendMessage()`:

- Criando uma JMS conexão com IBM MQ a partir de um factory `jms/CF1`, usando `connectionFactory.createConnection()`.
- Criando uma sessão do JMS a partir da conexão.
- Criando um produtor de mensagem por meio da sessão.
- Enviando uma mensagem.
- Fechando o produtor.
- Fechando a sessão.
- Fechando a conexão, chamando `connection.close()`.

Suponha que o conjunto livre para o factory `jms/CF1` esteja vazio. Quando o EJB é chamado pela primeira vez, o bean tenta criar uma conexão com o IBM MQ por meio do factory `jms/CF1`. Como o conjunto livre para o factory está vazio, o gerenciador de conexões cria uma nova conexão e fornece-a para o `EJB1`.

Logo antes do método sair, ele chama `connection.close()`. Em vez de fechar `c1`, o gerenciador de conexões pega a conexão e a coloca no conjunto livre para `jms/CF1`.

Na próxima vez que `sendMessage()` for chamado, o método `connectionFactory.createConnection()` retorna `c1` para o aplicativo.

Suponha que você tenha uma segunda instância do EJB em execução ao mesmo tempo que a primeira instância. Quando ambas as instâncias estão chamando `sendMessage()`, duas conexões são criadas por meio do connection factory `jms/CF1`.

Agora, suponha que uma terceira instância do bean seja criada. Quando o terceiro bean chama `sendMessage()`, o método chama `connectionFactory.createConnection()` para criar uma conexão por meio de `jms/CF1`.

No entanto, há atualmente duas conexões criadas por meio de `jms/CF1`, o que iguala o valor de conexões máximas para esse factory. Portanto, o método `createConnection()` aguardará 180 segundos (o valor padrão da propriedade de tempo limite de conexão) para uma conexão ser disponibilizada.

No entanto, se o método `sendMessage()` para o primeiro EJB chamar `connection.close()` e sair, a conexão que ele estava usando, `c1`, será colocada de volta no conjunto de conexões livre. O gerenciador de conexões retira a conexão do conjunto livre e fornece-a para o terceiro EJB. A chamada desse bean para `connectionFactory.createConnection()`, em seguida, é retornada, permitindo que o método `sendMessage()` seja concluído.

Portas do listener do MDB e EJBs usando o mesmo conjunto de conexões

Os dois exemplos anteriores mostram como as portas do listener e os EJBs podem usar o conjunto de conexões no isolamento. No entanto, é possível ter tanto uma porta do listener quanto um EJB em execução dentro do mesmo servidor de aplicativos e criar conexões JMS usando o mesmo connection factory.

É necessário considerar as implicações desta situação

A coisa mais importante a ser lembrada é que o connection factory é compartilhado entre a porta do listener e o EJB.

Por exemplo, suponha que você tenha um listener e um EJB em execução ao mesmo tempo. Ambos estão usando o connection factory `jms/CF1`, o que significa que o limite de conexões especificado pela propriedade de conexões máximas para esse factory foi atingido.

Se você tentar iniciar outra porta do listener ou outra instância de um EJB, terá que esperar que uma conexão seja retornada ao conjunto de conexões livres para `jms/CF1`.

Encadeamentos de manutenção do conjunto de
Associado a cada conjunto de conexões livres está um encadeamento de manutenção do conjunto, que monitora o conjunto livre para assegurar que suas conexões ainda sejam válidas.

Se o encadeamento de manutenção do conjunto decidir que uma conexão no conjunto livre precisa ser descartada, o encadeamento fechará fisicamente a conexão JMS com o IBM MQ.

Como Funciona o Encadeamento de Manutenção

O comportamento do encadeamento de manutenção do conjunto é determinado pelo valor de quatro propriedades do conjunto de conexões:

Tempo limite atingido
O período de tempo que uma conexão permanece aberta.

Conexões mínimas
O número mínimo de conexões que o gerenciador de conexões mantém no conjunto livre de um connection factory.

Tempo de leitura
Com que frequência o encadeamento de manutenção do conjunto é executado.

Tempo limite não utilizado
Quanto tempo uma conexão permanece no conjunto livre antes de ser encerrada.

Por padrão, o encadeamento mantido do conjunto é executado a cada 180 segundos, embora esse valor possa ser mudado ao configurar a propriedade **Reap time** do conjunto de conexões.

O encadeamento de manutenção examina cada conexão no conjunto, verifica quanto tempo ela está no conjunto e quanto tempo decorreu desde que foi criada e usada pela última vez.

Se a conexão não tiver sido usada por um período maior que o valor da propriedade **Unused timeout** para o conjunto de conexões, o encadeamento de manutenção verificará o número de conexões atualmente no conjunto livre. Se esse número for:

- Maior que o valor de **Minimum connections**, o gerenciador de conexões fechará a conexão.
- Igual ao valor de **Minimum connections**, a conexão não será encerrada e permanecerá no conjunto livre.

O valor padrão da propriedade **Minimum connections** é *1*, o que significa que, por motivos de desempenho, o gerenciador de conexões sempre tenta manter pelo menos uma conexão no conjunto livre.

A propriedade **Unused timeout** possui um valor padrão de 1.800 segundos. Por padrão, se uma conexão for colocada de volta no conjunto livre e não for usada novamente por pelo menos 1.800 segundos, essa conexão será encerrada, contanto que fechá-la, deixe pelo menos uma conexão no conjunto livre.

Esse procedimento evita que conexões não usadas se tornem antigas. Para desativar esse recurso, configure a propriedade **Unused timeout** como zero.

Se uma conexão estiver no conjunto livre e o tempo decorrido desde sua criação for maior que o valor da propriedade **Aged timeout** para o conjunto de conexões, ela será encerrada independentemente de quanto tempo ela tem desde a última vez que foi usada.

Por padrão, a propriedade **Aged timeout** é configurada como zero, o que significa que o encadeamento de manutenção nunca executa essa verificação. As conexões que existirem há mais tempo que a propriedade **Aged timeout** serão descartadas, independentemente de quantas conexões

permanecerão no conjunto livre. Observe que a propriedade **Minimum connections** não afeta essa situação.

Desativando o encadeamento de manutenção do conjunto

Na descrição anterior, é possível ver que o encadeamento de manutenção do conjunto faz um grande trabalho quando ativo, especificamente quando há um grande número de conexões no conjunto livre do connection factory.

Por exemplo, suponha que haja três connection factories do JMS, com a propriedade **Maximum connections** configurada como 10 para cada factory. A cada 180 segundos, três encadeamentos de manutenção do conjunto ficam ativos e varrem os conjuntos livres para cada connection factory, respectivamente. Se os conjuntos livres tiverem muitas conexões, os encadeamentos de manutenção terão muito trabalho a fazer, podendo afetar significativamente o desempenho.

É possível desativar o encadeamento de manutenção do conjunto para um conjunto de conexões livres individual configurando sua propriedade **Reap time** como zero.

Desativar o encadeamento de manutenção significa que as conexões nunca serão encerradas, mesmo se o **Unused timeout** tiver decorrido. No entanto, as conexões ainda poderão ser encerradas se o **Aged timeout** tiver passado.

Quando um aplicativo tiver concluído com uma conexão, o gerenciador de conexões verificará por quanto tempo a conexão existe e se esse período for maior que o valor da propriedade **Aged timeout** , o gerenciador de conexões fechará a conexão em vez de retorná-la para o conjunto livre..

Implicações Transacionais do Tempo Limite Aged

Conforme descrito na seção anterior, a propriedade **Aged timeout** especifica quanto tempo uma conexão com o provedor JMS permanece aberta antes de o gerenciador de conexões a fechar.

O valor padrão para a propriedade **Aged timeout** é zero, o que significa que a conexão nunca será encerrada por ser muito velha. Você deve deixar a propriedade **Aged timeout** com esse valor, pois a ativação de **Aged timeout** pode ter implicações transacionais ao usar JMS dentro de EJBs.

No JMS, a unidade de uma transação é uma JMS *sessão*, que é criada a partir de uma JMS *conexão* É a JMS *sessão* que está inscrita em transações e não a JMS *conexão*.

Devido ao design do servidor de aplicativos, as conexões JMS poderão ser encerradas porque o **Aged timeout** decorreu, mesmo se as sessões JMS criadas dessa conexão estiverem envolvidas em uma transação.

Fechar uma conexão JMS faz com que qualquer trabalho transacional pendente em sessões JMS seja recuperado, conforme descrito na especificação JMS. No entanto, o servidor de aplicativos não reconhece que as sessões JMS criadas da conexão não são mais válidas. Quando o servidor tenta usar a sessão para confirmar ou recuperar uma transação, ocorre uma `IllegalStateException`.

Importante: Se desejar usar **Aged timeout** com conexões JMS por meio de EJBs, assegure-se de que qualquer trabalho JMS seja explicitamente confirmado na sessão JMS antes do método EJB que executa as saídas de operações do JMS.

Exemplos de encadeamento de manutenção do conjunto
Usando o exemplo Enterprise JavaBean (EJB) para entender como o encadeamento de manutenção do conjunto funciona. Observe que também é possível usar os Message Driven Beans (MDBs) e as portas do listener, pois tudo o que você precisa é uma maneira de obter conexões no conjunto livre.

Consulte “Como os aplicativos que executam o sistema de mensagens de saída usam o conjunto de conexões” na página 315 para obter detalhes adicionais sobre o método `sendMessage()`.

Você configurou o connection factory com os valores a seguir:

- **Reap time** em seu valor padrão de 180 segundos
- **Aged timeout** em seu valor padrão de zero segundos

- **Unused timeout** configurado para 300 segundos

Depois que o servidor de aplicativos é inicializado, o método `sendMessage()` é chamado.

O método cria uma conexão chamada, por exemplo, `c1`, usando o factory `jms/CF1`, usa esse factory para enviar uma mensagem e, em seguida, chama `connection.close()`, que faz com que `c1` seja colocada no conjunto livre.

Depois de 180 segundos, o encadeamento de manutenção do conjunto é inicializado e examina o conjunto de conexões livres de `jms/CF1`. Há uma conexão livre `c1` no conjunto, portanto, o encadeamento de manutenção examina o horário em que a conexão foi colocada de volta e compara com o horário atual.

180 segundos se passaram desde que a conexão foi colocada no conjunto livre, que é menor que o valor da propriedade **Unused timeout** para `jms/CF1`. Portanto, o encadeamento de manutenção deixa a conexão sozinha.

180 segundos depois, o encadeamento de manutenção do conjunto é executado novamente. O encadeamento de manutenção localiza a conexão `c1` e determina que ela está no conjunto há 360 segundos, que é maior que o valor **Unused timeout** configurado, portanto, o gerenciador de conexões fecha a conexão.

Se agora você executar o método `sendMessage()` novamente, quando o aplicativo chamar `connectionFactory.createConnection()`, o gerenciador de conexões criará uma nova conexão com o IBM MQ porque o conjunto de conexões livres do connection factory está vazio.

O exemplo anterior mostrado como o encadeamento de manutenção usa as propriedades **Reap time** e **Unused timeout** para evitar conexões antigas, quando a propriedade **Aged timeout** é configurada como zero..

Como a propriedade **Aged timeout** funciona?

No exemplo a seguir, suponha que você tenha configurado a:

- Propriedade **Aged timeout** para 300 segundos
- Propriedade **Unused timeout** para zero.

Você chama o método `sendMessage()` e esse método tenta criar uma conexão por meio do connection factory `jms/CF1`.

Como o conjunto livre para esse factory está vazio, o gerenciador de conexões cria uma nova conexão, `c1` e a retorna para o aplicativo. Quando `sendMessage()` chama `connection.close()`, `c1` é colocado de volta no conjunto de conexões livres.

180 segundos depois, o encadeamento de manutenção do conjunto é executado. O encadeamento localiza `c1` no conjunto de conexões livres e verifica há quanto tempo ele foi criado. A conexão existiu por 180 segundos, que é menor que **Aged timeout**, portanto, o encadeamento de manutenção do conjunto a deixa sozinha e volta para a inatividade.

60 segundos depois, `sendMessage()` é chamada novamente. Desta vez, quando o método chama `connectionFactory.createConnection()`, o gerenciador de conexões descobre que há uma conexão, `c1`, disponível no conjunto livre para `jms/CF1`. O gerenciador de conexões retira `c1` do conjunto livre e fornece essa conexão para o aplicativo.

A conexão é retornada para o conjunto livre quando `sendMessage()` sai. 120 segundos depois, o encadeamento de manutenção do conjunto acorda novamente, varre o conteúdo do conjunto livre para `jms/CF1` e descobre `c1`.

Embora a conexão tenha sido usada apenas 120 segundos atrás, o encadeamento de manutenção do conjunto fecha a conexão, porque ela existe há um total de 360 segundos, que é maior que o valor de 300 segundos configurado para a propriedade **Aged timeout**.

Como a propriedade Conexões mínimas afeta o encadeamento de manutenção do conjunto

Usando o exemplo "Como as portas do listener do MDB usam o conjunto de conexões" na página 314 novamente, suponha que você tenha dois MDBs implementados no servidor de aplicativos, cada um usando uma porta de listener diferente.

Cada porta do listener é configurada para usar o connection factory `jms/CF1`, que você configurou com a:

- Propriedade **Unused timeout** configurada como 120 segundos
- Propriedade **Reap time** configurada como 180 segundos
- Propriedade **Minimum connections** configurada como 1

Suponha que o primeiro listener seja interrompido e sua conexão `c1` seja colocada no conjunto livre. 180 segundos mais tarde, o encadeamento de manutenção do conjunto acorda, varre o conteúdo do conjunto livre para `jms/CF1` e descobre que `c1` está no conjunto livre por mais tempo que o valor da propriedade **Unused timeout** do connection factory.

No entanto, antes de fechar `c1`, o encadeamento de manutenção do conjunto procura ver quantas conexões permanecerão no conjunto se essa conexão for descartada. Como `c1` é a única conexão no conjunto de conexões livres, o gerenciador de conexões não a fecha, porque fazer isso tornaria o número de conexões que permanecem no conjunto livre menor que o valor configurado para **Minimum connections**.

Agora, suponha que o segundo listener está interrompido. O conjunto de conexões livres agora contém duas conexões livres - `c1` e `c2`.

180 segundos depois, o encadeamento de manutenção do conjunto é executado novamente. Desta vez, `c1` está no conjunto de conexões livres há 360 segundos e `c2` há 180 segundos.

O encadeamento de manutenção do conjunto verifica `c1` e descobre que ele está no conjunto por mais tempo que o valor da propriedade **Unused timeout**.

O encadeamento, então, verifica para ver quantas conexões estão no conjunto livre e compara com o valor da propriedade **Minimum connections**. Como o conjunto contém duas conexões e **Minimum connections** está configurado como 1, o gerenciador de conexões fecha `c1`.

O encadeamento de manutenção agora examina `c2`. Isso também esteve no conjunto de conexão livre por mais tempo que o valor da propriedade **Unused timeout** . No entanto, como fechar `c2` deixaria o conjunto de conexões livres com menos do que o número configurado de conexões Mínimas nele, o gerenciador de conexões deixa `c2` sozinho.

Conexões do JMS e IBM MQ
Informações sobre o uso do IBM MQ como o provedor JMS.

Usando o Transporte de Ligações

Se um connection factory tiver sido configurado para usar o transporte de ligações, cada conexão JMS estabelecerá uma conversa (também conhecida como **hconn**) com o IBM MQ. A conversa usa a comunicação interprocessual (ou memória compartilhada) para se comunicar com o gerenciador de filas.

Usando o transporte de cliente

Quando um connection factory do provedor de sistemas de mensagens do IBM MQ tiver sido configurado para usar o transporte do cliente, cada conexão criada por meio desse factory estabelecerá uma nova conversa (também conhecida como **hconn**) com o IBM MQ.

Para connection factories que se conectam a um gerenciador de filas usando o modo normal do provedor de sistemas de mensagens do IBM MQ, é possível que diversas conexões JMS criadas por meio do connection factory compartilhem uma conexão TCP/IP com o IBM MQ. Para obter mais informações, consulte "Compartilhando uma conexão TCP/IP em IBM MQ classes for JMS" na página 322.

Para determinar o número máximo de canais de cliente usados pelas conexões JMS em um determinado momento, inclua o valor da propriedade *Maximum connections* para todos os connection factories que apontarem para o mesmo gerenciador de filas.

Por exemplo, suponha que você tenha dois connection factories, `jms/CF1` e `jms/CF2`, que tenham sido configurados para se conectar ao mesmo gerenciador de filas do IBM MQ usando o mesmo canal IBM MQ.

Esses factories estão usando as propriedades padrão do conjunto de conexões, o que significa que *Maximum connections* está configurado como 10. Se todas as conexões estiverem sendo usadas por meio de jms/CF1 e jms/CF2 ao mesmo tempo, haverá 20 conversas entre o servidor de aplicativos e o IBM MQ.

Se o connection factory se conectar ao gerenciador de filas usando o modo normal do provedor de sistemas de mensagens do IBM MQ, o número máximo de conexões TCP/IP que poderão existir entre o servidor de aplicativos e o gerenciador de filas para esses connection factories será:

```
20/the value of SHARECNV for the IBM MQ channel
```

Se o connection factory estiver configurado para se conectar usando o modo de migração do provedor de sistemas de mensagens do IBM MQ, o número máximo de conexões TCP/IP entre o servidor de aplicativos e o IBM MQ para esses connection factories será 20 (um para cada conexão JMS nos conjuntos de conexões para os dois factories).

Conceitos relacionados

“Usando IBM MQ classes for JMS/Jakarta Messaging” na página 83
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

O conjunto de objetos em um ambiente do Java SE
Com o Java SE (ou com outra estrutura, como Spring) os modelos de programação são extremamente flexíveis. Portanto, uma estratégia de conjunto único não atende todos. Será necessário considerar se há uma estrutura adequada que pudesse executar qualquer forma de conjunto, por exemplo, Spring.

Caso contrário, a lógica de aplicativo pode fazer essa ativação. Pergunte a si mesmo o quão complexo é o próprio aplicativo? É melhor entender o aplicativo e o que ele exige a partir da conectividade com o sistema de mensagens. Os aplicativos geralmente são gravados também dentro de seu próprio código de wrapper em torno da API básica do JMS.

Enquanto isso pode ser uma abordagem muito sensata e pode ocultar a complexidade, é importante ter em mente que pode apresentar problemas. Por exemplo, um método genérico `getMessage()`, que é frequentemente chamado, não deve apenas abrir e fechar os consumidores.

Pontos necessários a considerar:

- Quanto tempo o aplicativo precisará acessar o IBM MQ? Todo o tempo ou apenas ocasionalmente.
- Com que frequência as mensagens serão enviadas? Menor frequência, mais uma única conexão com o IBM MQ pode ser compartilhada.
- Uma exceção de conexão interrompida é normalmente um sinal de necessidade de recriar uma conexão agrupada. Sobre:
  - Exceções de segurança ou host não disponível
  - Fila cheia de exceções
- Se uma exceção de conexão interrompida ocorrer, o que deve acontecer às outras conexões livres no conjunto? Elas devem ser fechadas e recriadas?
- Se o TLS estiver sendo usado, por exemplo, quanto tempo deseja que uma única conexão permaneça aberta?
- Como uma conexão agrupada se identifica para que um administrador do gerenciador de filas possa marcar a conexão e rastreá-la de volta.

É necessário considerar todos os objetos do JMS para o conjunto e agrupar esse objeto sempre que for possível fazer isso. Os objetos incluem:

- Conexões do JMS
- Sessão
- Contextos
- Produtores e consumidores de todos os tipos diferentes

Ao usar o transporte do cliente, contextos, sessões e conexões do JMS usará soquetes ao se comunicar com o gerenciador de filas do IBM MQ. O agrupamento desses objetos, a economia está no número de conexões de entrada do IBM MQ (`hConns`) para o gerenciador de filas e uma redução no número de instâncias do canal.

O uso do transporte de ligações com o gerenciador de filas remove a camada de rede inteira. No entanto, muitos aplicativos usam o transporte de cliente para fornecer uma configuração e carga de trabalho balanceada mais altamente disponível.

Produtores e consumidores do JMS abrem destinos no gerenciador de filas. Se menos números de filas ou tópicos forem abertos e várias partes do aplicativo estiverem usando esses objetos, agrupar isso poderá ser útil.

De uma perspectiva do IBM MQ, esse processo salva uma sequência de operações MQOPEN e MQCLOSE.

Conexões, sessões e contextos

Esses objetos todos contêm manipulações de conexões do IBM MQ para o gerenciador de filas e são gerados em um `ConnectionFactory`. É possível incluir lógica em um aplicativo para restringir o número de conexões e outros objetos criados em uma única connection factory para um número específico.

É possível usar uma estrutura de dados simples no aplicativo para conter as conexões criadas. O código do aplicativo que precisa usar uma dessas estruturas de dados pode *fazer check-out* de um objeto a ser usado.

Execute os seguintes fatores em consideração:

- Quando as conexões devem ser removidas do conjunto? Geralmente, crie um listener de exceção na conexão. Quando esse listener for chamado para processar uma exceção, será necessário recriar a conexão e quaisquer sessões criadas por meio dessa conexão.
- Se uma CCDT estiver em uso para o balanceamento de carga de trabalho, as conexões poderão ir para gerenciadores de filas diferentes. Isso pode ser aplicável ao conjunto de requisitos.

Lembre-se de que a especificação do JMS afirma que é um erro de programação para vários encadeamentos estar acessando uma sessão ou um contexto ao mesmo tempo. O código IBM MQ JMS tenta ser rigoroso na sua manipulação de encadeamentos. No entanto, é necessário incluir a lógica no aplicativo, para assegurar-se de que um objeto de sessão ou contexto só é usado por um encadeamento por vez.

Produtores e consumidores

Cada produtor e consumidor criado abre um destino no gerenciador de filas. Se o mesmo destino for ser usado para várias tarefas, faz sentido manter os objetos do consumidor ou do produtor abertos. Apenas feche o objeto quando todo o trabalho for feito.

Apesar de que abrir e fechar um destino sejam operações de curtas, se elas forem realizadas frequentemente o tempo gasto pode ser incluído.

O escopo destes objetos está dentro da sessão ou do contexto ao qual são criados, portanto, eles precisam ser mantidos dentro desse escopo. Geralmente, os aplicativos são gravados de forma que isto seja muito simples a fazer.

Monitoramento

Como os aplicativos irão monitorar seus conjuntos de objetos? A resposta para isso é amplamente determinada pela complexidade da solução do conjunto implementado.

Se você considerar uma implementação do conjunto JavaEE, haverá um grande número de opções, incluindo:

- Tamanho atual dos conjuntos
- Tempo que os objetos gastaram neles
- A limpeza dos conjuntos
- Atualização das conexões

Também é necessário considerar como uma sessão única reutilizada aparece no gerenciador de filas. Há propriedades da connection factory para identificar o aplicativo (como appName) que pode ser útil.

"Usando IBM MQ classes for JMS/Jakarta Messaging" na página 83
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

Compartilhando uma conexão TCP/IP em IBM MQ classes for JMS
Várias instâncias de um canal MQI podem ser feitas para compartilhar uma única conexão TCP/IP.

Os aplicativos que estão em execução dentro do mesmo ambiente de tempo de execução do Java e que usam o adaptador de recursos IBM MQ classes for JMS ou IBM MQ para se conectar a um gerenciador de filas usando o transporte CLIENT podem ser feitos para compartilhar uma instância do canal.

Se um canal for definido com o parâmetro **SHARECNV** configurado para um valor maior que 1, então esse número de conversas poderá compartilhar uma instância de canal. Para ativar um connection factory ou uma especificação de ativação para usar essa função, configure a propriedade **SHARECONVALLOWED** como SIM.

Cada conexão do JMS e sessão do JMS criada por um aplicativo JMS cria sua própria conversa com o gerenciador de filas.

Quando uma especificação de ativação é inicializada, o adaptador de recursos do IBM MQ inicia uma conversa com o gerenciador de filas para a especificação de ativação a ser usada. Cada sessão do servidor no conjunto de sessões do servidor que está associada com a especificação de ativação também inicia uma conversação com o gerenciador de filas.

O atributo **SHARECNV** é uma abordagem de melhor esforço para o compartilhamento de conexão Portanto, quando um valor **SHARECNV** maior que 0 é usado com o IBM MQ classes for JMS, não é garantido que uma nova solicitação de conexão sempre compartilhe uma conexão já estabelecida.

Como as conexões TCP/IP são compartilhadas

Há duas estratégias disponíveis para o compartilhamento de conexões TCP/IP:

A estratégia GLOBAL
Essa estratégia é a estratégia padrão para compartilhar conexões TCP/IP. Qualquer conexão ou sessão JMS pode usar uma conversa em qualquer conexão TCP/IP adequada. A adequação é determinada por fatores como endereço do host, número da porta, ID do usuário e senha e parâmetros TLS/SSL.

Essa abordagem para compartilhar conexões TCP/IP minimiza o número de instâncias de canal que estão em uso, mas à custa de contenção para acesso a um conjunto global de conexões TCP/IP.

A estratégia CONNECTION
Com essa estratégia, as instâncias de canal são compartilhadas apenas entre objetos JMS relacionados. Especificamente, quando uma conexão do JMS é criada, uma instância do canal é criada

para ela e conversas adicionais nessa instância do canal estão disponíveis apenas para sessões do JMS que são criadas por essa conexão do JMS .

Se mais conversas forem criadas do que o atributo SHARECNV especifica, uma nova instância do canal será criada, que pode ser usada apenas por sessões do JMS que são criadas pela conexão JMS original.

Essa abordagem para compartilhar instâncias do canal reduz a contenção para conversas, à custa de potencialmente requerer mais instâncias do canal.

Especificando explicitamente uma estratégia de compartilhamento de instância de canal

V 9.4.0

Por padrão, a estratégia GLOBAL será usada se os aplicativos não forem reconectáveis Aplicativos reconectáveis sempre usam a estratégia CONNECTION.

Para aplicativos que usam IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, a estratégia CONNECTION pode ser ativada para aplicativos não reconectáveis em uma base de todo o aplicativo. É possível ativar a estratégia CONNECTION configurando a propriedade de sistema `com.ibm.mq.jms.channel.sharing` para o valor `CONNECTION`. Esse valor não faz distinção entre maiúsculas e minúsculas e qualquer valor diferente de `CONNECTION` é ignorado.

É possível configurar a propriedade de sistema `com.ibm.mq.jms.channel.sharing` de uma das seguintes maneiras:

- Configure a propriedade como parte da inicialização da JVM usando a opção da linha de comandos "-D":

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- Configure a propriedade antes de qualquer uso do IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging usando `System.setProperty()`

Calculando o número de instâncias de canal para a estratégia de compartilhamento GLOBAL

Use as fórmulas a seguir para determinar o número máximo de instâncias do canal que são criados por um aplicativo:

Especificações de ativação

Número de instâncias do canal = (*maxPoolDepth_value* + 1) / *SHARECNV_value*

Em que *maxPoolDepth_value* é o valor da propriedade **maxPoolDepth** e *SHARECNV_value* é o valor da propriedade **SHARECNV** no canal que é usado pela especificação de ativação.

Outros aplicativos JMS

Número de instâncias do canal = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

Em que *jms_connections* é o número de conexões que um aplicativo criou, *jms_sessions* é o número de JMS sessões que são criadas pelo aplicativo e *SHARECNV_value* é o valor da propriedade **SHARECNV** no canal que é usado pela especificação de ativação.

Calculando o número de instâncias de canal para a estratégia de compartilhamento CONNECTION

O número de instâncias de canal depende da distribuição de JMS sessões entre as conexões JMS no aplicativo.

Permita uma conversa para a conexão JMS e uma conversa para cada sessão JMS sob essa conexão JMS , em seguida, divida pelo valor **SHARECNV** , arredondando para cima. Esse cálculo fornece as instâncias de canal necessárias para essa conexão do JMS .

O mesmo princípio pode ser aplicado às especificações de ativação Considere a especificação de ativação como uma conexão JMS e a propriedade **maxPoolDepth** como o número de sessões JMS .

Exemplos

Os exemplos a seguir mostram como usar as fórmulas para calcular o número de instâncias do canal que são criadas em um gerenciador de filas por aplicativos usando o adaptador de recursos IBM MQ classes for JMS ou IBM MQ.

Exemplo de aplicativoJMS
Uma conexão de aplicativo JMS se conecta a um gerenciador de filas usando o transporte CLIENT e cria uma conexão do JMS e três sessões do JMS. O canal que o aplicativo está usando para se conectar ao gerenciador de filas tem a propriedade **SHARECNV** configurada o valor de 10. Quando o aplicativo está em execução, existem quatro conversas entre o aplicativo e o gerenciador de filas e uma instância do canal. As quatro conversações compartilham entre si a instância do canal.

Exemplo de especificação de ativação
Uma especificação de ativação se conecta a um gerenciador de filas usando o transporte CLIENT. A especificação de ativação é configurada com o conjunto de propriedades **maxPoolDepth** configurado para 10. O canal que a especificação de ativação está configurada para usar tem o conjunto de propriedades **SHARECNV** configurado para 10. Quando a especificação de ativação está em execução e o processando 10 mensagens simultaneamente, o número de conversações entre a especificação de ativação e o gerenciador de filas é 11 (10 conversações para as sessões do servidor e um para a especificação de ativação). O número de instâncias do canal que são usadas pela especificação de ativação é 2.

Exemplo de especificação de ativação
Uma especificação de ativação se conecta a um gerenciador de filas usando o transporte CLIENT. A especificação de ativação é configurada com o conjunto de propriedades **maxPoolDepth** configurado para 5. O canal que a especificação de ativação está configurado para usar tem o conjunto de propriedades **SHARECNV** configurado para 0. Quando a especificação de ativação está em execução, e processando 5 mensagens simultaneamente, o número de conversas entre a especificação de ativação e o gerenciador de filas é de 6 (cinco conversas para as sessões do servidor, e uma para a especificação de ativação). O número de instâncias do canal que são usadas pela especificação de ativação é 6 porque a propriedade **SHARECNV** no canal está configurada como 0; cada conversação usa sua própria instância do canal...

Tarefas relacionadas
“Determinando o número de conexões TCP/IP criadas do WebSphere Application Server para o IBM MQ” na página 512
Usando o recurso de compartilhamento de conversas, várias conversas podem compartilhar instâncias do canal MQI; isso também é conhecido como uma conexão TCP/IP.

A especificação de um intervalo de portas para conexões do cliente no IBM MQ classes for JMS
Use a propriedade LOCALADDRESS para especificar um intervalo de portas ao qual seu aplicativo pode ser ligado.

Quando um aplicativo IBM MQ classes for JMS tentar se conectar a um gerenciador de filas do IBM MQ no modo do cliente, o firewall poderá permitir apenas aquelas conexões originadas de portas especificadas ou de um intervalo de portas. Nesta situação, é possível usar a propriedade LOCALADDRESS de um objeto ConnectionFactory, QueueConnectionFactory ou TopicConnectionFactory para especificar uma porta ou um intervalo de portas, que o aplicativo pode ser ligado.

É possível configurar a propriedade LOCALADDRESS usando a ferramenta de administração do IBM MQ JMS ou chamando o método setLocalAddress() em um aplicativo JMS. A seguir há um exemplo da configuração da propriedade de dentro de um aplicativo:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

Quando o aplicativo se conectar a um gerenciador de filas subsequentemente, o aplicativo será ligado a um endereço IP local e o número da porta no intervalo de 192.0.2.0(2000) para 192.0.2.0(3000).

Em um sistema com mais de uma interface de rede, também é possível usar a propriedade LOCALADDRESS para especificar qual interface de rede deve ser usada para uma conexão.

Para uma conexão em tempo real a um broker, a propriedade LOCALADDRESS será relevante apenas quando multicast for usado. Neste caso, é possível usar a propriedade para especificar qual interface de rede local deve ser usada para uma conexão, mas o valor da propriedade não deve conter um número de porta ou um intervalo de números de porta.

Os erros de conexão poderão ocorrer se você restringir o intervalo de portas. Se ocorrer um erro, uma JMSException será lançada com um MQException integrado que contém o código de razão MQRC_Q_MGR_NOT_AVAILABLE do IBM MQ e a mensagem a seguir:

```
A tentativa de conexão do soquete foi recusada devido às restrições de LOCAL_ADDRESS_PROPERTY
```

Um erro pode ocorrer se todas as portas no intervalo especificado estiverem em uso ou se o endereço IP especificado, um nome do host ou número de porta não for válido (um número de porta negativo, por exemplo).

Como o IBM MQ classes for JMS pode criar conexões diferentes daquelas requeridas por um aplicativo, sempre considere a especificação de um intervalo de portas. Em geral, cada sessão criada por um aplicativo requer uma porta e o IBM MQ classes for JMS pode requerer três ou quatro portas adicionais. Se um erro de conexão ocorrer, aumente o intervalo de portas.

A definição do conjunto de conexões, que é usada por padrão em IBM MQ classes for JMS, pode ter um efeito na velocidade na qual as portas podem ser reutilizadas. Como resultado, um erro de conexão poderá ocorrer enquanto as portas estiverem sendo liberadas.

Compactação de canal no IBM MQ classes for JMS
Um aplicativo IBM MQ classes for JMS pode usar os recursos do IBM MQ para compactar um cabeçalho da mensagem ou dados.

A compactação dos dados que fluem em um canal do IBM MQ pode melhorar o desempenho do canal e reduzir o tráfego na rede. Usando a função fornecida com o IBM MQ, é possível compactar os dados que fluem em canais de mensagens e em canais MQI. Em qualquer tipo de canal, é possível compactar dados de cabeçalho e dados de mensagem de forma independente uns dos outros. Por padrão, nenhum dados é compactado em um canal.

Um aplicativo IBM MQ classes for JMS especifica as técnicas que podem ser usadas para compactar o cabeçalho ou os dados da mensagem em uma conexão criando um objeto java.util.Collection. Cada técnica de compactação é um objeto Integer na coleta e a ordem na qual o aplicativo inclui as técnicas de compactação na coleta é a ordem na qual as técnicas de compactação são negociadas com o gerenciador de filas quando o aplicativo cria a conexão. O aplicativo pode então transmitir a coleta para um objeto ConnectionFactory chamando o método setHdrCompList(), para os dados do cabeçalho ou o método setMsgCompList() para dados da mensagem. Quando o aplicativo estiver pronto, ele poderá criar a conexão.

Os seguintes fragmentos de código ilustram a abordagem descrita. O primeiro fragmento de código mostra como implementar a compactação de dados de cabeçalho:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

O segundo fragmento de código mostra como implementar a compactação dos dados da mensagem:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
```

```
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

No segundo exemplo, as técnicas de compactação são negociadas na ordem RLE, em seguida ZLIBHIGH, quando a conexão é criada. A técnica de compactação selecionada não pode ser mudada durante a existência do objeto Connection. Para usar a compactação em uma conexão, os métodos setHdrCompList() e o setMsgCompList() deverão ser chamados antes de criar o objeto Connection.

Colocando mensagens de forma assíncrona no IBM MQ classes for JMS
Normalmente, quando um aplicativo enviar mensagens para um destino, o aplicativo precisará aguardar para o gerenciador de filas confirmar se ele processou a solicitação. É possível melhorar o desempenho do sistema de mensagens em algumas circunstâncias, escolhendo em vez de colocar mensagens assincronamente. Quando um aplicativo efetuar put de uma mensagem de forma assíncrona, o gerenciador de filas não retornará o sucesso ou falha de cada chamada, mas será possível verificar erros periodicamente.

Se um destino retornar o controle ao aplicativo, sem determinar se o gerenciador de filas recebeu a mensagem com segurança, dependerá das seguintes propriedades:

A propriedade de destino JMS PUTASYNCALLOWED (nome abreviado-PAALD).
PUTASYNCALLOWED controlará se os aplicativos do JMS poderão colocar mensagens de forma assíncrona, se essa opção for permitida pela fila subjacente ou pelo tópico que o destino do JMS representa.

A propriedade de fila ou de tópico IBM MQ DEFPRESP (tipo de resposta de put padrão).
DEFPRESP especifica se os aplicativos que colocam mensagens na fila ou publicam mensagens para o tópico podem usar a funcionalidade de postagem assíncrona.

A tabela a seguir mostra os valores possíveis para as propriedades PUTASYNCALLOWED e DEFPRESP e as combinações de valores usadas para ativar a funcionalidade de postagem assíncrona:

Tabela 46. Como as propriedades PUTASYNCALLOWED e DEFPRESP se combinam para determinar se as mensagens são colocadas em um destino de forma assíncrona.

Propriedade de filas doIBM MQ	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**Funcionalidade de colocação assíncrona ativada**
DEFPRESP=SYNC	Funcionalidade de colocação assíncrona não ativada	Funcionalidade de colocação assíncrona ativada	PUTASYNCALLOWED = AS_DEST or AS_Q_DEF or AS_T_DEF
DEFPRESP=ASYNC	Funcionalidade de colocação assíncrona não ativada	Funcionalidade de colocação assíncrona ativada	PUTASYNCALLOWED = AS_DEST or AS_Q_DEF or AS_T_DEF

É possível mudar o comportamento especificando a propriedade de destino IBM MQ-JMS para dizer "NO" ou "YES", conforme mostrado na tabela, mas ela também pode ser substituída para toda a Java Máquina virtual usando a JVM **SystemProperty** e valor:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

Para as mensagens enviadas em uma sessão transacionada, o aplicativo determinará definitivamente se o gerenciador de filas recebeu as mensagens com segurança quando chama `commit()`.

Se um aplicativo enviar mensagens persistentes em uma sessão transacionada e uma ou mais das mensagens não forem recebidas com segurança, a transação falhará ao confirmar e produz uma exceção. No entanto, se um aplicativo enviar mensagens não persistentes em uma sessão transacionada e uma ou

mais das mensagens não forem recebidas com segurança, a transação será confirmada com sucesso. O aplicativo não recebe nenhum feedback que as mensagens não persistentes não chegou com segurança.

Para as mensagens não persistentes enviadas em uma sessão que não é transacionada, a propriedade SENDCHECKCOUNT do objeto *ConnectionFactory* especifica quantas mensagens deverão ser enviadas, antes de IBM MQ classes for JMS verificar se o gerenciador de filas recebeu as mensagens com segurança.

Se uma verificação descobrir que uma ou mais mensagens não foram recebidas com segurança e o aplicativo registrou um listener de exceção com a conexão, IBM MQ classes for JMS chamará o método `onException()` do listener de exceção para transmitir uma exceção do JMS para o aplicativo.

A exceção JMS tem um código de erro de JMSWMQ0028 e esse código exibe a mensagem a seguir:

```
At least one asynchronous put message failed or gave a warning.
```

A exceção do JMS também tem uma exceção vinculada que fornece mais detalhes. O valor padrão da propriedade SENDCHECKCOUNT é zero, o que significa que nenhum dessas verificações é feita.

Esta otimização é mais benéfica para um aplicativo que se conecta a um gerenciador de filas no modo cliente e precisa enviar uma sequência de mensagens em sucessão rápida, mas não requer feedback imediato do gerenciador de filas para cada mensagem enviada. No entanto, um aplicativo pode ainda usar essa otimização mesmo se ele se conectar a um gerenciador de filas no modo de ligações, mas o benefício de desempenho esperado não será tão grande.

Nota: Se você estiver usando um **`MessageProducer`** não identificado para enviar uma mensagem sob uma transação, por padrão, as mensagens serão colocadas na fila usando o mecanismo de entrada assíncrono.

Isso pode ocorrer porque a API JMS permite que o **`MessageProducer`** seja criado sem especificar um Destino, usando a sintaxe:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

Neste cenário, o Destino JMS é fornecido quando a mensagem é enviada em vez de antes quando o **`MessageProducer`** é construído. Em termos da API do IBM MQ , isso resulta em um MQPUT1 sendo emitido para colocar a mensagem na fila.

Se você fizer isso sob um ponto de sincronização IBM MQ , o que significa (na terminologia JMS) colocar a mensagem em uma transação, usando uma Sessão JMS transacionada ou por meio do uso de um XA`Session` a API IBM MQ classes for JMS alterna para o uso de colocação assíncrona.

Usando a leitura antecipada com o IBM MQ classes for JMS
A funcionalidade de leitura antecipada fornecida pelo IBM MQ permite que mensagens não persistentes recebidas fora de uma transação sejam enviadas ao IBM MQ classes for JMS antes que um aplicativo as solicite. O IBM MQ classes for JMS armazena as mensagens em um buffer interno e as transmite ao aplicativo quando o aplicativo solicitá-las.

Os aplicativos IBM MQ classes for JMS que usam `MessageConsumers` ou `MessageListeners` para receber mensagens de um destino fora de uma transação podem usar a funcionalidade de leitura antecipada. O uso da leitura antecipada permite que os aplicativos que usam esses objetos se beneficiem do desempenho aprimorado quando receberem mensagens.

Se um aplicativo que usa `MessageConsumers` ou `MessageListeners` pode usar a leitura antecipada depende das seguintes propriedades:

A propriedade de destino do JMS READAHEADALLOWED (nome abreviado - RAALD).
READAHEADALLOWED controlará se os aplicativos do JMS poderão usar a leitura antecipada ao obter ou procurar mensagens não persistentes fora de uma transação, se a fila ou o tópico subjacente que o destino do JMS representa, permitir essa opção.

O DEFREADA da propriedade de fila ou tópico do IBM MQ (leitura antecipada padrão).
DEFREADA especifica se os aplicativos que estão recebendo ou procurando mensagens não persistentes fora de uma transação podem usar a leitura antecipada.

A tabela a seguir mostra os valores possíveis para as propriedades READAHEADALLOWED e DEFREADA e as combinações de valores usadas para ativar a funcionalidade de leitura antecipada:

Tabela 47. Como as propriedades READAHEADALLOWED e DEFREADA se combinam para determinar se a leitura antecipada é usada ao receber ou procurar mensagens não persistentes fora de uma transação.

Propriedade de filas doIBM MQ	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST or AS_Q_DEF or AS_T_DEF**
DEFREADA = NO	Funcionalidade de leitura antecipada ativada	Funcionalidade de leitura antecipada não ativada	Funcionalidade de leitura antecipada não ativada
DEFREADA = YES	Funcionalidade de leitura antecipada ativada	Funcionalidade de leitura antecipada não ativada	Funcionalidade de leitura antecipada ativada
DEFREADA = DISABLED	Funcionalidade de leitura antecipada não ativada	Funcionalidade de leitura antecipada não ativada	Funcionalidade de leitura antecipada não ativada

Se a funcionalidade de leitura antecipada estiver ativada, quando um `MessageConsumer` ou `MessageListener` for criado por um aplicativo, o IBM MQ classes for JMS criará um buffer interno para o destino que o `MessageConsumer` ou `MessageListener` estará monitorando. Há um buffer interno de cada `MessageConsumer` ou `MessageListener`. O gerenciador de filas iniciará o envio de mensagens não persistentes para o IBM MQ classes for JMS quando o aplicativo chamar um dos seguintes métodos:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

O IBM MQ classes for JMS retorna automaticamente a primeira mensagem de volta ao aplicativo através da chamada de método que o aplicativo fez. As outras mensagens não persistentes são armazenadas pelo IBM MQ classes for JMS no buffer interno criado para o destino. Quando o aplicativo solicitar a próxima mensagem para processar, o IBM MQ classes for JMS retornará a próxima mensagem no buffer interno.

O IBM MQ classes for JMS solicitará mais mensagens não persistentes a partir do gerenciador de filas quando o buffer interno estiver vazio.

O buffer interno usado pelo IBM MQ classes for JMS será excluído quando um aplicativo fechar um `MessageConsumer` ou o JMS Session que um `MessageListener` está associado.

Para `MessageConsumers`, quaisquer mensagens não processadas no buffer interno serão perdidas.

Ao usar o `MessageListeners`, o que acontece com as mensagens no buffer interno depende do READAHEADCLOSEPOLICY de propriedade de destino do JMS (nome abreviado - RACP). O valor padrão da propriedade é DELIVER_ALL, o que significa que a sessão do JMS usada para criar o `MessageListener` não será fechada até que todas as mensagens no buffer interno sejam entregues para o aplicativo. Se a propriedade for configurada como DELIVER_CURRENT, a sessão do JMS será fechada após a mensagem atual ter sido processada pelo aplicativo e todas as mensagens restantes no buffer interno serão descartadas.

Publicações retidas no IBM MQ classes for JMS
Um cliente IBM MQ classes for JMS pode ser configurado para usar publicações retidas.

Um publicador pode especificar que uma cópia de uma publicação deve ser retida para que ela possa ser enviada aos assinantes futuros que registrarem um interesse no tópico. Isso é feito em IBM MQ classes for JMS, ao configurar a propriedade inteira JMS_IBM_RETENÇÃO para o valor 1. Constantes foram definidas para esses valores na interface do tipo decom.ibm.msg.client.jms.JmsConstants. Por exemplo, se você tiver criado uma mensagem *msg*, para configurar como uma publicação retida, use o código a seguir:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

Agora é possível enviar a mensagem como normal. JMS_IBM_RETAIN também pode ser consultada em uma mensagem recebida. Portanto, é possível consultar se uma mensagem recebida é uma publicação retida.

Suporte XA no IBM MQ classes for JMS

JMS suporta transações compatíveis com XA em ligações e modos de cliente com um gerenciador de transações suportado dentro de um contêiner do JEE.

Se você precisar da funcionalidade XA em um ambiente de servidor de aplicativos, deverá configurar seu aplicativo de forma apropriada. Consulte a documentação própria de seu servidor de aplicativos para obter informações sobre como configurar os aplicativos para usar as transações distribuídas.

Um gerenciador de filas do IBM MQ não pode agir como um gerenciador de transações para JMS.

Atraso de entrega para mensagens JMS

Para JMS 2.0 ou posterior, é possível especificar um atraso de entrega ao enviar uma mensagem. O gerenciador de filas não entrega a mensagem até que o atraso de entrega especificado tenha decorrido posteriormente.

Um aplicativo pode especificar um atraso de entrega em milissegundos, quando ele envia uma mensagem, usando `MessageProducer.setDeliveryDelay(long deliveryDelay)` ou `JMSProducer.setDeliveryDelay(long deliveryDelay)`. Esse valor é incluído ao tempo que a mensagem é enviada e fornece a hora mais antiga em que qualquer outro aplicativo possa obter essa mensagem.

O atraso de entrega é implementado usando uma única fila de preparação interna. Mensagens que têm um atraso de entrega diferente de zero são colocadas nessa fila com um cabeçalho que indica o atraso da entrega e informações sobre a fila de destino. Um componente do gerenciador de filas chamado de o processador de atraso de entrega monitora as mensagens na fila temporária. Quando o atraso de entrega da mensagem estiver concluído, a mensagem será tirada da fila temporária e colocada na fila de destino.

Clientes de mensagens

A implementação do IBM MQ de atraso de entrega está disponível apenas para uso quando você estiver usando o cliente JMS. As restrições a seguir se aplicarão se você estiver usando o atraso de entrega com o IBM MQ. Essas restrições se aplicam igualmente a `MessageProducers` e `JMSProducers`, mas `JMSRuntimeExceptions` são lançadas no caso de `JMSProducers`..

- Qualquer tentativa de chamar `MessageProducer.setDeliveryDelay` com um valor diferente de zero quando conectado a um gerenciador de filas anterior ao IBM MQ 8.0 resultará em uma `JMSException` com uma mensagem MQRC_FUNCTION_NOT_SUPPORTED.
- O atraso de entrega não é suportado para destinos em cluster que têm um valor de **DEFBIND** diferente de MQBND_BIND_NOT_FIXED. Se um `MessageProducer` possui um conjunto de atraso de entrega diferente de zero e é feita uma tentativa de enviar para um destino que não atende esse requisito, a chamada resulta em uma `JMSException` com uma mensagem MQRC_OPTIONS_ERROR.
- Qualquer tentativa de configurar um valor de tempo de vida menor que um atraso de entrega diferente de zero especificado anteriormente, ou vice-versa, resulta em um `JMSException` com uma mensagem MQRC_EXPIRY_ERROR. Esta verificação é feita na chamada `setTimeToLive` ou `setDeliveryDelay` ou métodos `send`, dependendo do conjunto exato de operações escolhidas.

- O uso de publicações retidas e atraso de entrega não é suportado. A tentativa de publicar uma mensagem com um atraso de entrega, se essa mensagem foi marcada como retida usando `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)`, resulta em uma `JMSException` com uma mensagem MQRC_OPTIONS_ERROR.
- O atraso de entrega e o agrupamento de mensagens não são suportados e qualquer tentativa de usar essa combinação resulta em um `JMSException` com uma mensagem MQRC_OPTIONS_ERROR

Qualquer falha ao enviar uma mensagem com atraso de entrega resulta no cliente emitir uma `JMSException` com uma mensagem de erro adequada, por exemplo, fila cheia. Em algumas situações, a mensagem de erro pode se aplicar ao destino ou a fila temporária, ou ambos.

Nota: IBM MQ permite que os aplicativos que colocarem uma mensagem em uma unidade de trabalho obtenham a mesma mensagem novamente, mesmo que a unidade de trabalho não seja confirmada. Esta técnica não funciona com o atraso de entrega, a mensagem não será colocada na fila temporária até que a unidade de trabalho seja confirmada e como resultado não terá sido enviada para o destino.

Autorização

IBM MQ executará as verificações de autorização no destino original quando o aplicativo enviar uma mensagem com um atraso de entrega diferente de zero. Se o aplicativo não estiver autorizado, o envio falhará. Quando o gerenciador de filas detectar que o atraso de entrega de mensagem foi concluído, ele abrirá a fila de destino. Nenhuma verificação de autorização é realizada neste ponto.

SYSTEM.DDELAY.LOCAL.QUEUE

Uma fila do sistema, SYSTEM.DDELAY.LOCAL.QUEUE, é usada para implementar o atraso na entrega.

- Multi Em Multiplataformas, SYSTEM.DDELAY.LOCAL.QUEUE existe por padrão. A fila do sistema deve ser alterada para que seus atributos MAXMSGL e MAXDEPTH sejam suficientes para a carga esperada.
- z/OS Em IBM MQ for z/OS, SYSTEM.DDELAY.LOCAL.QUEUE é usado como uma fila de migração para mensagens que são enviadas com atraso de entrega para filas locais e compartilhadas. No z/OS, a fila deve ser criada e definida para que seus atributos MAXMSGL e MAXDEPTH sejam suficientes para a carga esperada.

Quando esta fila for criada, ela deverá ser protegida de modo que poucos usuários possível tenha acesso à ela. O acesso à fila deve ser para fins de manutenção e monitorando apenas.

Quando uma mensagem for enviada por um aplicativo JMS com um atraso de entrega diferente de zero, ela será colocada nessa fila com um novo ID de mensagem. O ID de mensagem original é colocado no ID de correlação da mensagem. Esse ID de correlação permite que um aplicativo recupere uma mensagem da fila temporária quando necessário, por exemplo, se um atraso de entrega grande foi usado por engano.

Considerações para o z/OS

z/OS

Se seu sistema estiver em execução no z/OS, haverá contraprestações adicionais para levar em conta se você desejar usar o atraso de entrega.

Se o atraso de entrega for para ser usado, o SYSTEM.DDELAY.LOCAL.QUEUE da fila do sistema deverá ser definido. Ele deve ser definido com uma classe de armazenamento que seja suficiente para a respectiva carga esperada e com INDXTYPE(NONE) e MSGDLVSQ(FIFO) especificados. Uma definição de amostra da fila do sistema é fornecida e comentada, na JCL CSQ4INSG.

Filas compartilhadas

O atraso de entrega é suportado para enviar mensagens para filas compartilhadas. No entanto, existe apenas uma única fila temporária privada usada independentemente se a fila de destino é compartilhada

ou não. O gerenciador de filas que possui essa fila privada deve estar em execução para enviar a mensagem atrasada para sua fila compartilhada de destino quando o atraso for concluído.

Nota: Se uma mensagem não persistente for colocada com um atraso de entrega em uma fila compartilhada e o gerenciador de filas que possui a fila temporária for encerrado, a mensagem original será perdida. Como resultado, as mensagens não persistentes enviadas com atraso de entrega para uma fila compartilhada são mais prováveis que sejam perdidas que as mensagens não persistentes enviadas sem atraso de entrega para uma fila compartilhada.

Resolução de destino

Se a mensagem for enviada para uma fila, a resolução será acionada duas vezes; uma vez pelo aplicativo JMS e uma vez pelo gerenciador de filas quando ela tirar a mensagem da fila temporária e enviá-la para a fila de destino.

As assinaturas de destino para as publicações serão correspondidas quando o aplicativo JMS chamar o método send.

Se uma mensagem for enviada com persistência ou prioridade de acordo com a definição de fila, então o valor será configurado na primeira resolução e não na segunda.

Intervalo de expiração

O atraso de entrega preserva o comportamento da propriedade de expiração, **MQMD.Expiry** Por exemplo, se uma mensagem foi colocada a partir de um aplicativo JMS com um intervalo de expiração de 20.000 ms e um atraso de entrega de 5.000 ms, e chegou após um tempo decorrido de 10.000 ms, então o valor do campo MQMD.expiry poderá ser aproximadamente 50 décimos de segundo. Este valor indica que 15 segundos decorreu desde o momento em que a mensagem foi colocada, até o momento em que ela foi obtido.

Se uma mensagem expirar enquanto estiver na fila temporária e uma das opções MQRO_EXPIRATION_* estiver configurada, então o relatório gerado será para a mensagem original como enviada pelo aplicativo, o cabeçalho usado para conter as informações de atraso de entrega será removido.

Parando e iniciando o processador de atraso de entrega

z/OS No z/OS, o processador de atraso de entrega é integrado no espaço de endereço MSTR do gerenciador de filas. Quando o gerenciador de filas for iniciado, o processador de atraso de entrega também será iniciado. Se a fila temporária estiver disponível, ela abrirá a fila e aguardará a chegada de mensagens nela para ser processada. Se a fila temporária não foi definida ou estiver desativada para gets, ou outro erro ocorrer, o processador de atraso de entrega será encerrado. Se a fila temporária for posteriormente definida ou alterada para ser ativada, o processador de atraso de entrega será reiniciado. Se o processador de atraso de entrega for encerrado por qualquer outro motivo, ele poderá ser reiniciado alterando o atributo **PUT** da fila de migração de dados de ENABLED para DISABLED e novamente para ENABLED . Se você precisar parar o processador de atraso de entrega por qualquer razão, configure o atributo PUT da fila temporária para DISABLED.

Multi Em Multiplataformas, o processador de atraso é iniciado com o gerenciador de filas e é reiniciado automaticamente no caso de uma falha recuperável.

Falha ao colocar na fila de destino

Se uma mensagem atrasada não puder ser colocada na fila de destino quando seu atraso for concluído, a mensagem será tratada conforme indicado em suas opções de relatório: ela será descartada ou enviada à fila de mensagens não entregues. Se esta ação falhar, então uma tentativa será feita para colocar a mensagem posteriormente. Se a ação for bem-sucedida, um relatório de exceções será gerado e enviado à fila especificada, se o relatório for solicitado. Se a mensagem de relatório não pôde ser enviada, a mensagem de relatório será enviada à fila de mensagens não entregues. Se o envio do relatório para a fila de mensagens não entregues falhar e a mensagem for persistente, todas as mudanças serão descartadas

e a mensagem original retrocedida e entregue novamente mais tarde. Se a mensagem não for persistente a mensagem de relatório será descartada, mas outras mudanças serão confirmadas. Se uma publicação atrasada não puder ser entregue porque um assinante cancelou a assinatura ou, no caso de um assinante não durável, porque ela foi desconectada, a mensagem será descartada silenciosamente. As mensagens de relatório são ainda geradas conforme descrito anteriormente.

Se uma publicação atrasada não puder ser entregue a um assinante e em vez disso for colocada na fila de mensagens não entregues e falhar, a mensagem será descartada.

Para reduzir a probabilidade de falha na colocação na fila de destino após o atraso de entrega ter sido concluído, o gerenciador de filas executará algumas verificações básicas quando o cliente JMS enviar uma mensagem com um atraso de entrega diferente de zero. Essas verificações incluem se a fila é colocada desativada, se a mensagem é maior que o comprimento máximo de mensagem permitido e se a fila está cheia.

Publicação/assinatura

A correspondência de uma publicação para assinaturas disponíveis ocorrerá quando o aplicativo JMS enviar uma mensagem com um atraso de entrega diferente de zero. Uma mensagem para cada assinante correspondente é colocada na fila SYSTEM.DDELAY.LOCAL.QUEUE, em que é mantida até que o atraso de entrega ser concluído. Se um desses assinantes for uma assinatura de proxy para outro gerenciador de filas, então o fan-out nesse gerenciador de filas ocorrerá após o atraso de entrega ser concluído. Isso pode resultar em assinantes no outro gerenciador de filas recebendo publicações que foram originalmente publicadas antes que eles tenham assinados. Este é um desvio da especificação JMS 2.0 ou mais recente

O atraso de entrega com publicação/assinatura será suportado apenas se o tópico de destino estiver configurado com (N)PMSGDLV = ALLAVAIL. Uma tentativa de usar qualquer outros valores resulta em um erro MQRC_PUBLICATION_FAILURE. Se o processador de atraso de entrega falhar enquanto ele estiver colocando a mensagem na fila de destino, o resultado será como descrito na seção "Falha ao colocar na fila de destino".

Mensagens de relatório

Todas as opções de relatório são suportadas e acionadas pelo processador de entrega, além das seguintes opções ignoradas, mas transmitidas através da mensagem quando ela for enviada para a fila de destino:

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

Assinaturas clonadas e compartilhadas

Existem dois métodos para fornecer a vários consumidores acesso à mesma assinatura. Esses dois métodos são através do uso de assinaturas clonadas ou compartilhadas.

Assinaturas clonadas

Assinatura clonada é uma extensão do IBM MQ. As assinaturas clonadas ativam vários consumidores em acesso simultâneo diferente do Java Virtual Machines (JVMs) para a assinatura. Esse comportamento pode ser usado, configurando a propriedade **CLONESUPP** como `Ativado` em um objeto connectionFactory. Por padrão **CLONESUPP** é `Desativado`. As assinaturas clonadas podem ser ativadas apenas nas assinaturas duráveis. Se **CLONESUPP** for ativado, cada conexão subsequente feita usando esta connectionFactory será clonada.

Uma assinatura durável poderá ser considerada clonada, se um ou mais consumidores forem criados para receber mensagens dessa assinatura, ou seja, eles foram criados especificando o mesmo nome de assinatura. Isso poderá ser feito somente se a conexão com o qual os consumidores foram criados tem

CLONESUPP configurado como `Ativado` na MQConnectionFactory. Quando uma mensagem for publicada no tópico da assinatura, uma cópia dessa mensagem será enviada para a assinatura. A mensagem está disponível para qualquer um dos consumidores, mas apenas um o recebe.

Nota: A ativação de assinaturas clonadas estende a especificação de JMS.

Assinaturas compartilhadas

As assinaturas compartilhadas, que foram introduzidas pela especificação JMS 2.0 , permitem que as mensagens de uma assinatura de tópico sejam compartilhadas entre vários consumidores Cada mensagem da assinatura é entregue somente para um dos consumidores nessa assinatura. As assinaturas compartilhadas são ativadas pela chamada relevante para a API JMS 2.0 ou posterior.

As APIs podem ser chamadas em uma das seguintes maneiras:

- De um aplicativo Java SE (ou contêiner do cliente Java EE).
- A partir de um servlet ou da implementação de um MDB.

A especificação JMS 2.0 ou mais recente não define nenhuma maneira padrão de conduzir um MDB a partir de uma assinatura compartilhada, portanto, o IBM MQ fornece a propriedade de especificação de ativação **sharedSubscription** para esse propósito Para obter mais informações sobre essa propriedade, consulte "Configurando o adaptador de recursos para comunicação de entrada" na página 461 e "Exemplos de como definir a propriedade sharedSubscription" na página 478.

Se uma assinatura compartilhada for ativada, ela não poderá ser não compartilhada.

As assinaturas compartilhadas podem ser criadas como assinaturas duráveis ou não duráveis. Não há nenhum requisito para criar separadamente quaisquer objetos no lado do gerenciador de filas além da configuração normal do JMS Quaisquer objetos necessários são criados dinamicamente.

Decidindo entre assinaturas compartilhadas ou clonadas

Quando estiver decidindo se deseja usar assinaturas compartilhadas ou clonadas, considere os benefícios de ambos. Onde possível, use assinaturas compartilhadas, pois é um comportamento definido por especificação, em vez de uma extensão específica IBM MQ .

A tabela a seguir contém algumas das questões a serem consideradas quando estiver decidindo entre assinaturas compartilhadas e clonadas:

Tabela 48. Comparação de considerações para assinaturas compartilhadas e assinaturas clonadas

Assinaturas compartilhadas	**Assinaturas clonadas**
Assinaturas compartilhadas são uma parte padrão da especificação JMS 2.0 ou posterior.	As assinaturas clonadas são uma extensão específica do IBM MQ.
As assinaturas são criadas usando chamadas de método explícito da API.	As assinaturas clonadas são controladas administrativamente no nível de ConnectionFactory.
Assinaturas compartilhadas podem ser duráveis ou não.	Assinaturas clonadas somente pode ser duráveis.
As assinaturas são explicitamente criadas em uma base de assinatura individual.	As assinaturas clonadas são usadas para qualquer assinatura durável em uma conexão na qual a função está ativada.
Se uma assinatura for criada como compartilhada, ela não poderá ser mudada posteriormente para não compartilhada ou vice-versa.	Uma assinatura poderá ser mudada de clonada para não clonada toda vez que for reaberta, se a propriedade **CLONESUPP** da conexão proprietária tiver mudado.

Conceitos relacionados

Assinantes e assinaturas

Durabilidade da assinatura

Tarefas relacionadas

Usando assinaturas compartilhadas do JMS 2.0

Referências relacionadas

“Exemplos de como definir a propriedade sharedSubscription” na página 478

Você pode definir a propriedade sharedSubscription de uma especificação de ativação dentro de um arquivo WebSphere Liberty `server.xml`. Como alternativa, é possível definir a propriedade dentro de um bean acionado por mensagens (MDB) usando anotações.

CLONESUPP

Propriedade SupportMQExtensions

A especificação JMS 2.0 introduziu mudanças na maneira como determinados comportamentos funcionam.. IBM MQ 8.0 e posterior inclui a propriedade **`com.ibm.mq.jms.SupportMQExtensions`**, que pode ser configurada como TRUE para reverter esses comportamentos alterados de volta para as implementações anteriores

JM 3.0 A propriedade **`com.ibm.mq.jakarta.jms.SupportMQExtensions`** (Jakarta Messaging 3.0) é suportada pelo IBM MQ classes for Jakarta Messaging, que está disponível em `com.ibm.mq.jakarta.client.jar`.

JMS 2.0 A propriedade **`com.ibm.mq.jms.SupportMQExtensions`** (JMS 2.0) é suportada pelo IBM MQ classes for JMS, que está disponível em `com.ibm.mq.allclient.jar` ou `com.ibm.mqjms.jar`.

Três áreas de funcionalidade são revertidas configurando **`SupportMQExtensions`** para `True`:

Prioridade da mensagem
: As mensagens podem receber uma prioridade de 0 a 9. Antes de JMS 2.0, as mensagens também poderiam usar o valor -1, indicando que uma prioridade padrão da fila é usada. JMS 2.0 e mais recente não permitem que uma prioridade de mensagem de -1 seja configurada para. Ativar **`SupportMQExtensions`** permite que o valor de -1 seja usado.

Identificador de cliente
: A especificação JMS 2.0 ou posterior requer que IDs de cliente não nulos sejam verificados para exclusividade quando eles fizerem uma conexão. Ativar **`SupportMQExtensions`** significa que esse requisito é desconsiderado e que um ID de cliente pode ser reutilizado.

NoLocal
: A especificação JMS 2.0 ou posterior requer que quando essa constante for ativada, um consumidor não possa receber mensagens publicadas pelo mesmo ID do cliente. Antes do JMS 2.0, esse atributo era configurado em um assinante para evitar que recebesse mensagens publicadas por sua própria conexão. Ativar **`SupportMQExtensions`** reverterá esse comportamento para sua implementação anterior.

Essa propriedade pode ser configurada como a seguir:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

Essa propriedade pode ser configurada como uma propriedade padrão do Sistema da JVM no comando **`java`** ou contida no arquivo de configuração do IBM MQ classes for JMS.

Conceitos relacionados

“O arquivo de configuração IBM MQ classes for JMS/Jakarta Messaging” na página 100

Os arquivos de configuração IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging especificam as propriedades que são usadas para configurar IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging.

Referências relacionadas

“Propriedades usadas para configurar o comportamento do cliente JMS” na página 107
Use essas propriedades para configurar o comportamento do cliente JMS.

Usando assinaturas compartilhadas em aplicativos JMS

Com assinaturas compartilhadas, uma única assinatura é compartilhada entre vários consumidores, com apenas um dos consumidores recebendo uma publicação em qualquer momento.

As assinaturas compartilhadas estão disponíveis a partir do JMS 2.0 . Quando você está desenvolvendo um aplicativo JMS para IBM MQ 8.0 ou mais recente, pode ser necessário, portanto, considerar o impacto dessa funcionalidade em seu gerenciador de filas

A ideia por trás das assinaturas compartilhadas é compartilhar a carga entre vários consumidores.. Uma assinatura durável também pode ser compartilhada entre vários consumidores.

Por exemplo, suponha que haja:

- Uma assinatura SUB, assinando um tópico FIFA2014/UPDATES para receber atualizações de correspondência de futebol, sendo compartilhada por três consumidores C1, C2 e C3
- Um produtor P1 publicando no tópico FIFA2014/UPDATES

Quando for feita uma publicação no FIFA2014/UPDATES, ela será recebida por apenas um dos três consumidores (C1, C2 ou C3) mas não todos.

A amostra a seguir demonstra o uso de assinaturas compartilhadas e também demonstra o uso da API adicional em JMS 2.0, `Message.receiveBody()`, para recuperar somente o corpo da mensagem

A amostra cria três encadeamentos de assinante, que criam uma assinatura compartilhada para o tópico FIFA2014/UPDATES e um encadeamento do publicador.

JM 3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
```

```
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){
```

```
                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
```

```
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

Conceitos relacionados

Interfaces de linguagem do IBM MQ Java

V 9.4.0 Configurando seu aplicativo modular para usar IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

V 9.4.0 É possível usar IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging de maneira modular, requerendo o módulo apropriado em seu aplicativo e incluindo o diretório apropriado no caminho do módulo.

A embalagem modular

Os arquivos JAR unificados para IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging fornecem nomes de módulos automáticos, que substituem os nomes padrão derivados dos nomes de arquivos JAR.

- IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) são fornecidos com um nome de módulo `com.ibm.mq.javax`.
- IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) são fornecidos com um nome de módulo `com.ibm.mq.jakarta`.

O diretório padrão `MQ_HOME/java/lib` é inadequado para uso modular porque os módulos não podem conter o mesmo pacote e o diretório padrão contém os mesmos pacotes em vários JARs Portanto, novos diretórios estão disponíveis para que contenham apenas os arquivos JAR necessários, sem duplicação de pacotes entre os JARs. Esses diretórios são adequados para inclusão em um `module-path`

Nota: Se você tiver aplicativos que usam os arquivos JAR disponíveis em um contexto modular, dependendo dos nomes de módulo padrão, deverá atualizar seus aplicativos para requerer os novos nomes de módulo. Os nomes de módulos padrão são derivados dos nomes de arquivos JAR

Configurando seu aplicativo modular para usar o IBM MQ classes for JMS

É possível configurar seu aplicativo modular para usar IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) concluindo as etapas a seguir:

- Configure o aplicativo para requerer o módulo `com.ibm.mq.javax` .
- Configure o aplicativo para incluir o diretório `MQ_HOME/java/lib/modules/javax` no caminho do módulo..

Configurando seu aplicativo modular para usar o IBM MQ classes for Jakarta Messaging

É possível configurar seu aplicativo modular para usar IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) concluindo as etapas a seguir:

- Configure o aplicativo para requerer o módulo `com.ibm.mq.jakarta` .
- Configure o aplicativo para incluir o diretório `MQ_HOME/java/lib/modules/jakarta` no caminho do módulo..

Configurando seu aplicativo modular para usar o IBM MQ classes for Java

Para usar IBM MQ classes for Java de um aplicativo modular, é possível usar a configuração para IBM MQ classes for JMS ou a configuração para IBM MQ classes for Jakarta Messaging, pois ambos os arquivos JAR do cliente suportam IBM MQ classes for Java. No entanto, seu aplicativo deve usar apenas uma dessas configurações, não ambas.

IBM MQ classes for JMS Application Server Facilities

Este tópico descreve como o IBM MQ classes for JMS implementa a classe ConnectionConsumer e a funcionalidade avançada na classe Session. Ele também resume a função de um conjunto de sessões do servidor.

Importante: Estas informações são apenas para referência. Um aplicativo não deve ser gravado para usar esta interface: é usado dentro do adaptador de recursos do IBM MQ para se conectar a servidores Java EE. Para obter informações de conexão na prática, consulte "Usando o adaptador de recursos do IBM MQ" na página 444.

O IBM MQ classes for JMS suporta o Application Server Facilities (ASF) especificado na *Java Message Service Especificação* (consulte Oracle Technology Network for Java Developers). Esta especificação identifica três funções nesse modelo de programação:

- **O provedor JMS** fornece as funcionalidades ConnectionConsumer e Sessão avançada.
- **O servidor de aplicativos** fornece a funcionalidade ServerSessionPool e ServerSession.
- **O aplicativo cliente** usa a funcionalidade que o servidor de aplicativos e o provedor JMS fornecem.

As informações neste tópico não se aplicarão se um aplicativo usar uma conexão em tempo real a um broker.

O ConnectionConsumer do JMS

A interface ConnectionConsumer fornece um método de alto desempenho para entregar mensagens simultaneamente para um conjunto de encadeamentos.

A especificação do JMS permite que um servidor de aplicativos integre diretamente com uma implementação do JMS usando a interface `ConnectionConsumer`. Este recurso fornece processamento simultâneo de mensagens. Geralmente, um servidor de aplicativos cria um conjunto de encadeamentos e a implementação do JMS faz com que as mensagens estejam disponíveis para esses encadeamentos. Um servidor de aplicativos reconhecido pelo JMS (como WebSphere Application Server) pode usar esse recurso para fornecer funcionalidade do sistema de mensagens de alto nível, como beans acionados por mensagens.

Os aplicativos não usam o ConnectionConsumer, mas os clientes JMS especialistas podem usá-lo. Para esses clientes, a ConnectionConsumer fornece um método de alto desempenho para entregar mensagens simultaneamente para um conjunto de encadeamentos. Quando uma mensagem chegar em uma fila ou um tópico, JMS selecionará um encadeamento do conjunto e entregará um lote de mensagens para ela. Para fazer isso, o JMS executa um método `onMessage()` do MessageListener associado.

É possível obter o mesmo efeito, construindo vários objetos Session e MessageConsumer, cada um com um MessageListener registrado. No entanto, o ConnectionConsumer fornece melhor desempenho, menos uso de recursos e maior flexibilidade. Em particular, menos objetos Session são necessários.

Planejando um aplicativo com ASF

Esta seção informa como planejar um aplicativo, incluindo:

- "Princípios gerais para o sistema de mensagens ponto a ponto usando o ASF" na página 340
- "Princípios gerais para o sistema de mensagens de publicação/assinatura usando o ASF" na página 341
- "Removendo mensagens da fila em ASF" na página 342
- Manipulando mensagens suspeitas no ASF. Consulte o "Manipulando mensagens suspeitas no IBM MQ classes for JMS" na página 238.

Princípios gerais para o sistema de mensagens ponto a ponto usando o ASF

Use este tópico para obter informações gerais sobre o sistema de mensagens ponto a ponto usando o ASF.

Quando um aplicativo criar um ConnectionConsumer a partir de um objeto QueueConnection, ele especificará uma sequência do seletor e um objeto da fila do JMS. O ConnectionConsumer, em seguida, começa a fornecer mensagens para sessões no ServerSessionPool associado. As mensagens chegam na fila e se corresponderem ao seletor, elas serão entregues para sessões no ServerSessionPool associado.

Em termos do IBM MQ, o objeto da fila se refere a um QLOCAL ou um QALIAS no gerenciador de filas local. Se for um QALIAS, esse QALIAS deverá se referir a um QLOCAL. O QLOCAL IBM MQ totalmente resolvido é conhecido como o *QLOCAL subjacente*. Um ConnectionConsumer será considerado como *ativo* se ele não estiver fechado e seu pai QueueConnection for iniciado.

É possível para vários ConnectionConsumers, cada um com diferentes seletores, ser executado no mesmo QLOCAL subjacente. Para manter o desempenho, as mensagens indesejadas não devem se acumular na fila. As mensagens indesejadas são aquelas as quais nenhum ConnectionConsumer ativo possui um seletor correspondente. É possível configurar o QueueConnectionFactory para que essas mensagens indesejadas sejam removidas da fila (para obter detalhes, consulte "Removendo mensagens da fila em ASF" na página 342). É possível configurar esse comportamento em uma de duas maneiras:

- Use a ferramenta de administração do JMS para configurar o QueueConnectionFactory para MRET(NO).
- Em seu programa, use:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Se você não mudar esta configuração, o padrão será reter essas mensagens indesejadas na fila.

Ao configurar o gerenciador de filas IBM MQ, considere os seguintes pontos:

- O QLOCAL subjacente deve estar ativado para a entrada compartilhada. Para fazer isso, use o seguinte comando MQSC:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- Seu gerenciador de filas deve ter uma fila de mensagens não entregues ativada. Se um ConnectionConsumer tiver um problema quando ele colocar uma mensagem na fila de mensagens não entregues, a entrega da mensagem a partir do QLOCAL subjacente será interrompida. Para definir uma fila de mensagens não entregues, use:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- O usuário que executa o ConnectionConsumer deve ter autoridade para executar MQOPEN com MQOO_SAVE_ALL_CONTEXT e MQOO_PASS_ALL_CONTEXT. Para obter detalhes, consulte a documentação do IBM MQ para sua plataforma específica.
- Se mensagens indesejadas forem deixadas na fila, elas irão degradar o desempenho do sistema. Portanto, planeje seus seletores de mensagens para que entre eles, o ConnectionConsumers removerá todas as mensagens da fila.

Para obter detalhes sobre comandos MQSC, consulte Comandos MQSC.

Princípios gerais para o sistema de mensagens de publicação/assinatura usando o ASF
ConnectionConsumers recebem mensagens para um Tópico especificado. Um ConnectionConsumer pode ser durável ou não durável. Deve-se especificar qual fila ou quais filas o ConnectionConsumer usa.

Quando um aplicativo criar um ConnectionConsumer a partir de um objeto TopicConnection, ele especificará um objeto Tópico e uma sequência do seletor. O ConnectionConsumer, em seguida, começa a receber mensagens que correspondem ao seletor nesse tópico, incluindo todas as publicações retidas para o tópico assinado.

Alternativamente, um aplicativo pode criar um ConnectionConsumer durável associado a um nome específico. Este ConnectionConsumer recebe mensagens que foram publicadas no Tópico desde o ConnectionConsumer durável estar ativo pela última vez. Ele recebe todas as mensagens que correspondem ao seletor no Tópico. No entanto, se o ConnectionConsumer estiver usando a leitura antecipada, ele poderá perder as mensagens não persistentes que estão no buffer do cliente quando ele for fechado.

Se o IBM MQ classes for JMS estiver em modo de migração do provedor com o sistema de mensagens do IBM MQ, uma fila separada será usada para as assinaturas do ConnectionConsumer não duráveis.A opção configurável CCSUB no TopicConnectionFactory especifica a fila a ser usada. Normalmente, o CCSUB especifica uma fila única para uso por todos os ConnectionConsumers que usam o mesmo TopicConnectionFactory. Entretanto, é possível fazer cada ConnectionConsumer gerar uma fila temporária especificando um prefixo de nome da fila seguidos de um asterisco (*).

Se o IBM MQ classes for JMS estiver no modo de migração do provedor com o sistema de mensagens do IBM MQ, a propriedade CCDSUB do Tópico especificará a fila a ser usada para assinaturas duráveis. Novamente, esta pode ser uma fila que já existe ou um prefixo de nome de fila seguido por um asterisco (*). Se você especificar uma fila que já existe, todos os ConnectionConsumers duráveis que assinam o Tópico utilizam esta fila. Se você especificar um prefixo de nome da fila seguido de um asterisco (*), uma fila será gerada pela primeira vez que um ConnectionConsumer durável for criado com um nome específico. Esta fila será reutilizada posteriormente quando um ConnectionConsumer durável for criado com o mesmo nome.

Ao configurar o gerenciador de filas IBM MQ, considere os seguintes pontos:

- Seu gerenciador de filas deve ter uma fila de mensagens não entregues ativada. Se um ConnectionConsumer tiver um problema quando ele colocar uma mensagem na fila de mensagens não entregues, a entrega da mensagem a partir do QLOCAL subjacente será interrompida. Para definir uma fila de mensagens não entregues, use:

  ```
  ALTER QMGR DEADQ( your.dead.letter.queue.name )
  ```

- O usuário que executa o ConnectionConsumer deve ter autoridade para executar MQOPEN com MQOO_SAVE_ALL_CONTEXT e MQOO_PASS_ALL_CONTEXT. Para obter detalhes, consulte a documentação do IBM MQ para sua plataforma.
- É possível otimizar o desempenho para um ConnectionConsumer individual criando uma fila separada e dedicada para isso. Este é o custo de uso de recursos extras.

Removendo mensagens da fila em ASF
Quando um aplicativo usa ConnectionConsumers, o JMS pode precisar remover as mensagens da fila em inúmeras situações.

Estas situações são as seguintes:

Mensagem formatada incorretamente
Pode chegar uma mensagem que o JMS não pode analisar.

Mensagem suspeita
Uma mensagem pode atingir o limite de restauração, mas o ConnectionConsumer não deve ser recolocado na fila de restauração.

Nenhum ConnectionConsumer interessado
Para o sistema de mensagens ponto a ponto, quando o QueueConnectionFactory for configurado para que ele não retenha mensagens indesejadas, uma mensagem chega que não é desejada por quaisquer dos ConnectionConsumers.

Nessas situações, o ConnectionConsumer tenta remover a mensagem da fila. As opções de disposição no campo de relatório de MQMD da mensagem configura o comportamento exato. Estas opções são:

MQRO_DEAD_LETTER_Q
A mensagem é enfileirada para a fila de mensagens não entregues do gerenciador de filas. Esse é o padrão.

MQRO_DISCARD_MSG
A mensagem será descartada.

O ConnectionConsumer também gera uma mensagem de relatório e isso também depende do campo de relatório MQMD da mensagem. Esta mensagem é enviada para o ReplyToQ da mensagem no ReplyToQmgr. Se houver um erro enquanto a mensagem de relatório estiver sendo enviada, a mensagem será enviada para a fila de mensagens não entregues. As opções de relatório de exceção no campo de relatório do MQMD da mensagem configura os detalhes da mensagem de relatório. Estas opções são:

MQRO_EXCEPTION
Uma mensagem de relatório que contém o MQMD da mensagem original é gerada. Ela não contém nenhum dado do corpo da mensagem.

MQRO_EXCEPTION_WITH_DATA
Uma mensagem de relatório que contém o MQMD, quaisquer cabeçalhos MQ e 100 bytes de dados do corpo é gerada.

MQRO_EXCEPTION_WITH_FULL_DATA
Uma mensagem de relatório que contém todos os dados da mensagem original é gerada.

padrão
Nenhuma mensagem de relatório é gerada.

Quando mensagens de relatório são geradas, as seguintes opções são favorecidas:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Se uma mensagem suspeita não puder ser enfileirada novamente, talvez porque a fila de mensagens não entregues esteja cheia ou a autorização está especificada incorretamente, o que acontece depende da persistência da mensagem. Se a mensagem for não persistente, a mensagem será descartada e nenhuma mensagem de relatório será gerada. Se a mensagem for persistente, a entrega de mensagens para todos os consumidores de conexão atendendo nesse destino parará. Estes consumidores de conexão devem ser fechados e o problema resolvido antes que possam ser recriados e a entrega de mensagens reiniciada.

É importante definir uma fila de mensagens não entregues e verificá-la regularmente para assegurar que nenhum problema ocorra. Particularmente, certifique-se de que a fila de mensagens não entregues não tenha atingido sua profundidade máxima e que seu tamanho máximo de mensagem é grande o suficiente para todas as mensagens.

Quando uma mensagem é recolocada na fila de mensagens não entregues, é precedida por um cabeçalho de mensagens não entregues do IBM MQ (MQDLH). Consulte MQDLH – Cabeçalho de mensagens não entregues para obter detalhes sobre o formato do MQDLH. É possível identificar mensagens que um ConnectionConsumer colocou na fila de mensagens não entregues ou mensagens de relatório que um ConnectionConsumer tenha gerado pelos seguintes campos:

- PutApplType é `MQAT_JAVA (0x1C)`
- PutApplO nome é "`MQ JMS ConnectionConsumer`"

Estes campos estão no MQDLH de mensagens na fila de mensagens não entregues e no MQMD de mensagens de relatório. O campo de feedback do MQMD e o campo Reason do MQDLH contêm um código que descreve o erro. Para obter detalhes sobre esses códigos, consulte “Códigos de razão e de feedback em ASF” na página 344. Outros campos estarão conforme descrito em MQDLH – Cabeçalho de mensagens não entregues.

Manipulando mensagens suspeitas no ASF
No Application Server Facilities, a manipulação de mensagens suspeitas é tratada de modo ligeiramente diferente do que em outras partes no IBM MQ classes for JMS.

Para obter informações sobre manipulação de mensagens suspeitas no IBM MQ classes for JMS, consulte “Manipulando mensagens suspeitas no IBM MQ classes for JMS” na página 238.

Ao usar o Application Server Facilities (ASF), o ConnectionConsumer, em vez de o MessageConsumer, processa mensagens suspeitas. O ConnectionConsumer recoloca mensagens na fila de acordo com as propriedades BackoutThreshold e BackoutRequeueQName da fila.

Quando um aplicativo usa ConnectionConsumers, as circunstâncias nas quais uma mensagem é restaurada dependem da sessão que o servidor de aplicativos fornece:

- Quando a sessão é não transacionada, com AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, uma mensagem é restaurada somente após um erro do sistema ou se o aplicativo for finalizado inesperadamente.

- Quando a sessão é não transacionada com CLIENT_ACKNOWLEDGE, mensagens não reconhecidas podem ser restauradas pelo servidor de aplicativos chamando Session.recover().

  Geralmente, a implementação do cliente do MessageListener ou do servidor de aplicativos chama Message.acknowledge(). Message.acknowledge() reconhece todas as mensagens entregues na sessão até o momento.
- Quando a sessão é transacionada, as mensagens não reconhecidas podem ser restauradas pelo servidor de aplicativos chamando Session.rollback().
- Se o servidor de aplicativos fornecer uma XASession, as mensagens serão confirmadas ou restauradas, dependendo de uma transação distribuída. O servidor de aplicativos assume a responsabilidade por concluir a transação.

Conceitos relacionados

"Manipulando mensagens suspeitas no IBM MQ classes for JMS" na página 238
Uma mensagem suspeita é uma que não pode ser processada por um aplicativo de recebimento. Se uma mensagem suspeita for entregue a um aplicativo e retrocedida um número especificado de vezes, o IBM MQ classes for JMS poderá movê-la para uma fila de restauração.

Manipulação de erros

Esta seção abrange diversos aspectos de manipulação de erros, incluindo "Recuperação de condições de erro no ASF" na página 344 e "Códigos de razão e de feedback em ASF" na página 344.

Recuperação de condições de erro no ASF

Se um ConnectionConsumer receber um erro grave, a entrega da mensagem para todos os ConnectionConsumers com um interesse no mesmo QLOCAL será interrompida. Quando isso ocorrer, qualquer ExceptionListener registrado com o Connection afetado será notificado. Há duas maneiras nas quais um aplicativo pode se recuperar destas condições de erro.

Geralmente, um grave erro desta natureza ocorrerá se o ConnectionConsumer não puder reenfileirar uma mensagem na fila de mensagens não entregues ou se ocorrer um erro ao ler mensagens a partir de QLOCAL.

Como qualquer ExceptionListener registrado com o Connection afetado é notificado, é possível usá-las para identificar a causa do problema. Em alguns casos, o administrador do sistema deve intervir para resolver o problema.

Use uma das técnicas a seguir para se recuperar destas condições de erro:

- Chame `close()` em todos os ConnectionConsumers afetados. O aplicativo poderá criar novos ConnectionConsumers somente após todos os ConnectionConsumers afetados serem fechados e quaisquer problemas do sistema resolvidos.
- Chame `stop()` em todas as Connections afetadas. Quando todas as Connections forem interrompidas e quaisquer problemas no sistema resolvidos, o aplicativo poderá efetuar `start()` de suas Connections com sucesso.

Códigos de razão e de feedback em ASF

Use códigos de razão e de feedback para determinar a causa de um erro. Códigos de razão comuns gerados pelo ConnectionConsumer são fornecidos aqui.

Para determinar a causa de um erro, use as seguintes informações:

- O código de feedback em todas as mensagens de relatório
- O código de razão no MQDLH de todas as mensagens na fila de mensagens não entregues

ConnectionConsumers gera os códigos de razão a seguir.

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Causa

A mensagem alcançou o limite de restauração definido no QLOCAL, mas nenhuma fila de restauração está definida.

Em plataformas nas quais não é possível definir a fila de restauração, a mensagem alcançou o limite de restauração de 20 definido no JMS.

Ação

Se isso não for desejado, defina a fila de restauração para o QLOCAL relevante. Além disso, procure pela causa das múltiplas restaurações.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Causa

No o sistema de mensagens ponto a ponto, há uma mensagem que não corresponde a nenhum dos seletores para os ConnectionConsumers que monitoram a fila. Para manter o desempenho, a mensagem é enfileirada novamente na fila de mensagens não entregues.

Ação

Para evitar essa situação, assegure-se de que ConnectionConsumers usando a fila forneçam um conjunto de seletores que lidam com todas as mensagens ou configure QueueConnectionFactory para reter as mensagens.

Como alternativa, investigue a origem da mensagem.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Causa

O JMS não pode interpretar a mensagem na fila.

Ação

Investigue a origem da mensagem. O JMS normalmente entrega mensagens de um formato inesperado como uma BytesMessage ou TextMessage. Ocasionalmente, isso falhará se a mensagem for muito mal formatada.

Outros códigos que aparecem nesses campos são causados por uma tentativa fracassada de enfileirar novamente a mensagem em uma fila de restauração. Nesta situação, o código descreve o motivo pelo qual o reenfileiramento falhou. Para diagnosticar a causa desses erros, consulte API> conclusão da API e códigos de razão.

Se a mensagem de relatório não puder ser colocada na ReplyToQ, ela será colocada na fila de mensagens não entregues. Nesta situação, o campo de feedback do MQMD estará concluído conforme descrito neste tópico. O campo razão no MQDLH explica o motivo pelo qual a mensagem de relatório não pôde ser colocada na ReplyToQ.

A função de um conjunto de sessões do servidor em AFS

Este tópico resume a função de um conjunto de sessões do servidor.

Figura 45 na página 346 resume os princípios da funcionalidade ServerSessionPool e ServerSession.

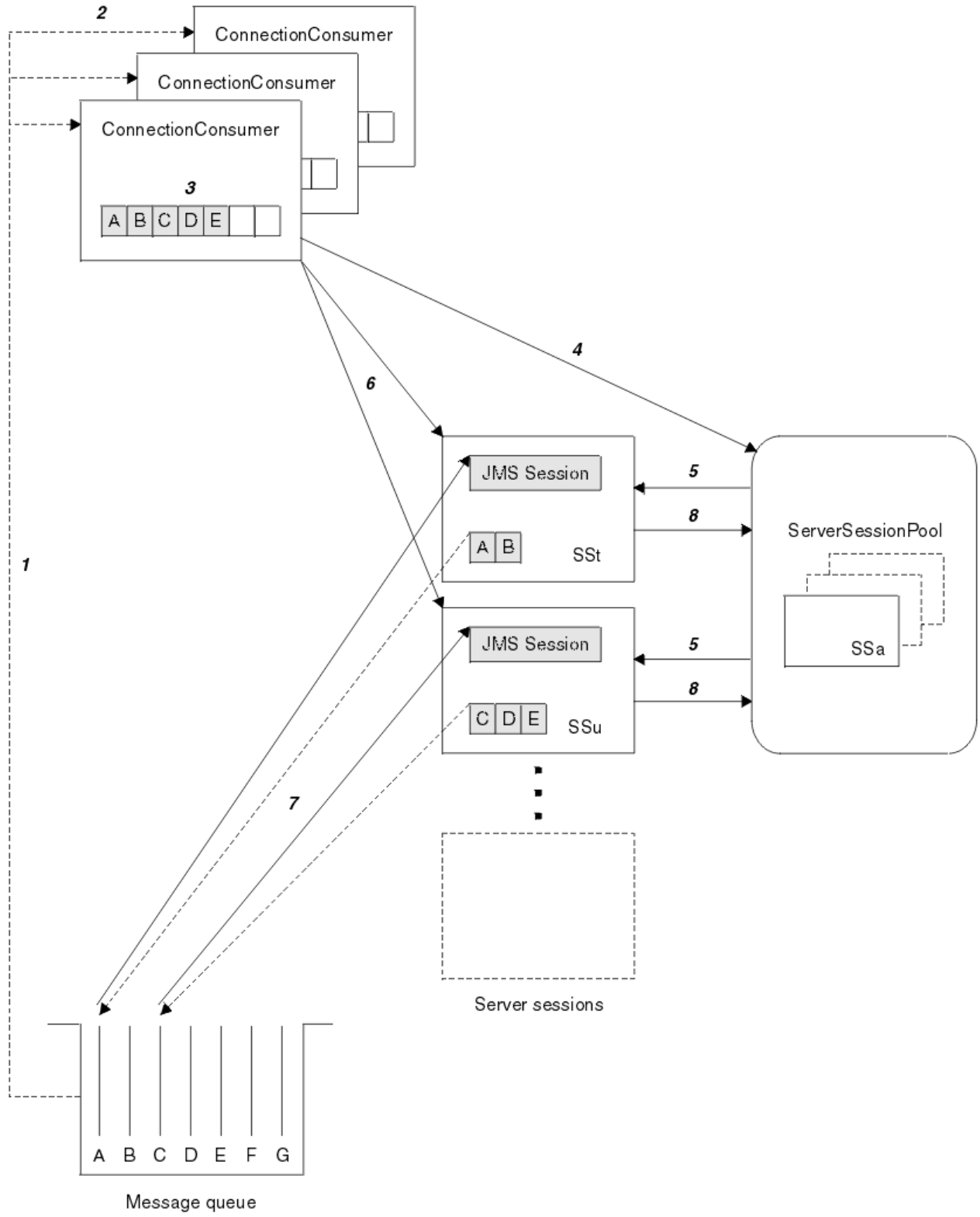


Figura 45. Funcionalidade ServerSessionPool e ServerSession

1. O ConnectionConsumers obtém referências de mensagem a partir da fila.
2. Cada ConnectionConsumer seleciona referências de mensagens específicas.
3. O buffer ConnectionConsumer contém as referências de mensagens selecionadas.
4. As solicitações ConnectionConsumer um ou mais ServerSessions a partir do ServerSessionPool.

5. ServerSessions são alocadas a partir do ServerSessionPool.
6. O ConnectionConsumer designa referências de mensagens ao ServerSessions e inicia os encadeamentos em execução ServerSession.
7. Cada ServerSession recupera suas mensagens referenciadas a partir da fila. Ele transmite-as ao método onMessage do MessageListener associado ao JMS Session.
8. Após concluir seu processamento, o ServerSession será retornado ao conjunto.

Um servidor de aplicativos geralmente fornece a funcionalidade ServerSessionPool e ServerSession.

z/OS Using IBM MQ classes for JMS in a CICS Liberty JVM server

Java programs running in a CICS Liberty JVM server can use the IBM MQ classes for JMS to access IBM MQ.

You must be using an IBM MQ 9.1.0 or later version of the IBM MQ resource adapter. You can obtain the resource adapter from Fix Central (see “Instalando o adaptador de recursos no Liberty” on page 453).

There are two flavors of Liberty Profile JVMs available in CICS 5.3 and later, the types of connection possible to IBM MQ are restricted as follows:

CICS Liberty Standard

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in CLIENT mode
- The IBM MQ resource adapter can connect to any in-service version of IBM MQ for z/OS in BINDINGS mode when there is no CICS connection (active CICS MQCONN resource definition) to the same queue manager from the same CICS region.

CICS Liberty Integrated

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in CLIENT mode.
- BINDINGS mode connection is not supported.

For details on setting up and configuring your system, see Using IBM MQ classes for JMS in a Liberty JVM server in the CICS documentation.

Usando IBM MQ classes for JMS/ Jakarta Messaging em IMS

O suporte ao sistema de mensagens baseado em padrões em um ambiente IMS é fornecido por meio do uso de IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging.

Verifique os requisitos do sistema para o sistema IMS usado por sua empresa. Consulte IMS 15.2 para obter mais informações.

Este conjunto de tópicos descreve como configurar o IBM MQ classes for JMS em um ambiente do IMS e as restrições de API que se aplicam ao usar as interfaces clássica (JMS 1.1) e simplificada (JMS 2.0). Consulte “Restrições de API do JMS” na página 352 para obter uma lista das informações específicas de API.

Nota: Restrições semelhantes se aplicam a interfaces específicas de domínio anterior (JMS 1.0.2), mas não são descritas especificamente aqui.

JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando IBM MQ classes para JMS/Jakarta Messaging.

Regiões dependentes do IMS suportadas

Os tipos de regiões dependentes a seguir são suportados:

- MPR
- BMP

- IFP
- Java virtual machines (JVMs) JMP de 31 e 64 bits
- JVMs JBP de 31 e 64 bits

A menos que especificamente mencionado nos tópicos a seguir, IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging comportam-se da mesma forma em todos os tipos de região

Java Virtual Machines suportadas

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging requerem IBM Runtime Environment, Java Technology Edition 8. IBM Semeru Runtime Certified Edition para z/OS a Versão 11 não é suportada.

Outras restrições

As restrições a seguir se aplicam ao usar o IBM MQ classes for JMS em um ambiente do IMS:

- Conexões de modo cliente não são suportadas.
- As conexões são suportadas apenas para os gerenciadores de filas do IBM MQ 8.0 usando o modo de IBM MQprovedor de mensagens `Normal`.

  O atributo **PROVIDERVERSION** no connection factory deve ser não especificado, ou um valor maior ou igual a sete.
- O uso de qualquer um das dos XA connection factories, por exemplo `com.ibm.mq.jms.MQXAConnectionFactory`, não é suportado.

Tarefas relacionadas

Definindo IBM MQ para IMS

Configurando o adaptador IMS para uso com IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging fazem uso do mesmo adaptador IBM MQ-IMS usado por outras linguagens de programação. Esse adaptador usa o IMS External Subsystem Attach Facility (ESAF).

Antes de começar

Antes de concluir o procedimento a seguir, deve-se configurar o adaptador do IMS para os gerenciadores de filas relevantes e as regiões de controle e dependentes do IMS, conforme descrito em Configurando o adaptador do IMS.

Atenção: Não é necessário executar a etapa que descreve a construção de um stub dinâmico, a menos que você precise do stub dinâmico para outros propósitos.

Depois de ter configurado o adaptador IMS , execute o procedimento a seguir:

Procedimento

1. Atualize a variável LIBPATH no membro de seu IMS PROCLIB que é referenciado pelo parâmetro ENVIRON na JCL da região dependente (por exemplo, DFSJVMEV) para que inclua as bibliotecas nativas do IBM MQ classes for JMS.

   Ou seja, o diretório zFS que contém `libmqjims.so`. Por exemplo, DFSJVMEV pode ser semelhante ao seguinte, em que a última linha é o diretório que contém as bibliotecas nativas IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging :

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
```

```
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. Inclua o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging no caminho de classe da JVM, usado por sua região dependente do IMS , atualizando a opção `java.class.path` .

   Faça isso seguindo as instruções em Membro DFSJVMMS do conjunto de dados IMS PROCLIB.

   Por exemplo, é possível usar o seguinte, em que a linha em negrito indica a atualização:

JM 3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   Nota: Embora haja muitos arquivos jar diferentes disponíveis no diretório que contém o IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, você precisa apenas `com.ibm.mq.allclient.jar` (JMS 2.0) ou `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0).

3. Pare e reinicie quaisquer regiões dependentes do IMS que farão uso do IBM MQ classes for JMS ou do IBM MQ classes for Jakarta Messaging

Como proceder a seguir

Crie e configure connection factories e destinos.

Existem três abordagens possíveis para instanciar as implementações do IBM MQ de connection factories e destinos. Consulte “Criando e configurando connection factories e destinos” na página 208 para obter detalhes.

Observe que essas três abordagens são todas válidas em um ambiente do IMS.

Conceitos relacionados

Configurando o adaptador do IMS

Definindo o IBM MQ para IMS

Comportamento transacional

As mensagens enviadas e recebidas pelo IBM MQ classes for JMS em um ambiente do IMS são sempre associadas à IMS unidade de trabalho (UOW) que está ativa na tarefa atual.

Essa UOW pode ser concluída apenas ao chamar os métodos commit ou rollback em uma instância do objeto `com.ibm.ims.dli.tm.Transaction` ou pela tarefa IMS que termina normalmente, caso em que a UOW está implicitamente confirmada. Se a tarefa IMS terminar de modo anormal, a UOW será retrocedida.

Como resultado disso, os valores dos argumentos **`transacted`** e **`acknowledgeMode`** serão ignorados ao chamar qualquer um dos métodos `Connection.createSession` ou `ConnectionFactory.createContext`. Além disso, os seguintes métodos não são suportados. Ao chamar qualquer um dos seguintes métodos resultará em um `IllegalStateException`, no caso da sessão:

- `javax.jms.Session.commit()`
- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

e um `IllegalStateRuntimeSession` no caso de contexto do JMS:

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`

- `javax.jms.JMSContext.rollback()`

Há uma exceção para esse comportamento. Se uma sessão ou contexto do JMS for criado usando um dos mecanismos a seguir:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

então, o comportamento dessa sessão ou contexto do JMS será o seguinte:

- Todas as mensagens enviadas, serão enviadas fora da UOW IMS. Ou seja, elas estarão disponíveis no destino imediatamente ou quando o intervalo de atraso de entrega fornecido estiver concluído.
- Quaisquer mensagens não persistentes serão recebidas fora da UOW do IMS, a menos que a propriedade SYNCPOINTALLGETS tenha sido especificada na connection factory que criou a sessão ou contexto do JMS.
- As mensagens persistentes serão sempre recebidas dentro da UOW do IMS.

Isso pode ser útil se, por exemplo, você deseja gravar uma mensagem de auditoria para uma fila mesmo se a UOW retroceder.

Implicações de pontos de sincronização do IMS

O IBM MQ classes for JMS agrega ao suporte existente do adaptador do IBM MQ que usa ESAF. Isso significa que o comportamento documentado se aplica, inclusive todos os identificadores abertos serem fechados pelo adaptador do IMS quando um ponto de sincronização ocorre.

z/OS Consulte “Syncpoints in IMS applications” na página 72 para obter mais informações.

Para ilustrar esse ponto, considere o código a seguir em execução em um ambiente do JMP. A segunda chamada a `mp.send()` resulta em uma `JMSException`, pois o código `messageQueue.getUnique(inputMessage)` resulta em todas as conexões do IBM MQ e identificadores de objetos abertos serem fechados.

Comportamento semelhante é observado se a chamada `getUnique()` tiver sido substituída por `Transaction.commit()`, mas não se `Transaction.rollback()` tiver sido usado.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

O código correto a ser usado nesse cenário é o seguinte. Nesse caso, a conexão com o IBM MQ é fechada antes de chamar `getUnique()`. A conexão e a sessão são, então, recriadas a fim de enviar outra mensagem.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
```

```
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

Considerações ao usar o adaptador IMS

É necessário estar ciente das restrições a seguir. É possível ter somente uma manipulação de conexões para cada gerenciador de filas. Há implicações na interação com o IBM MQ ao usar o JMS e código nativo. Há limitações para autenticação e autorização da conexão.

Uma manipulação de conexões para cada gerenciador de filas

Somente uma manipulação de conexões por vez em um gerenciador de filas específico é permitida nas regiões dependentes do IMS. Todas as tentativas subsequentes para se conectar ao mesmo gerenciador de filas reutilizam o identificador existente.

Embora esse comportamento não deva causar problemas em um aplicativo que use somente o IBM MQ classes for JMS, esse comportamento pode causar problemas em aplicativos que interagem com o IBM MQ ao usar o IBM MQ classes for JMS e a MQI no código nativo escrito em linguagens como COBOL ou C.

Implicações de interagir com o IBM MQ ao usar o JMS e o código nativo

Podem ocorrer problemas ao intercalar o código Java e o código nativo que usam a funcionalidade IBM MQ e quando a conexão com o IBM MQ não é fechada antes de deixar o código nativo ou Java

Por exemplo, no pseudo código a seguir, uma manipulação de conexões para um gerenciador de filas é originalmente estabelecida no código Java usando o IBM MQ classes for JMS. A manipulação de conexões é reutilizada no código COBOL e invalidada por uma chamada para MQDISC.

Na próxima vez que o IBM MQ classes for JMS usar a manipulação de conexões, isso resultará em uma `JMSException` com um código de razão MQRC_HCONN_ERROR.

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
```

```
Get session from static variable
Create a message consumer - fails as connection handle invalidated
```

Há outros padrões de uso semelhantes que podem resultar em MQRC_HCONN_ERROR.

Embora seja possível compartilhar identificadores de conexão do IBM MQ entre o código nativo e o código Java (por exemplo, o exemplo anterior funcionaria se não houvesse uma chamada MQDISC) em geral, a melhor prática é fechar quaisquer identificadores de conexão antes de mudar do código Java para o código nativo ou o contrário.

Autenticação e autorização de conexão

A especificação JMS permite que um nome de usuário e uma senha sejam especificados para autenticação e autorização ao criar uma conexão ou um objeto de contexto do JMS.

Isso não é suportado em um ambiente do IMS. Tentar criar uma conexão ao especificar um nome de usuário e senha resulta em um `JMS Exception` sendo lançado. Tentar criar um contexto do JMS, ao especificar um nome de usuário e uma senha, resulta em uma `JMSRuntimeException` ser lançada.

Em vez disso, os mecanismos existentes para autenticação e autorização ao se conectar ao IBM MQ a partir de um ambiente do IMS devem ser usados.

Para obter mais informações, consulte Configurando a segurança no z/OS. Em particular, consulte IDs do usuário para verificação de segurança, que descreve os IDs de usuário que podem ser usados.

Tarefas relacionadas

Configurando a Segurança em z/OS

Restrições de API do JMS

A partir de uma perspectiva de especificação JMS , o IBM MQ classes for JMS trata IMS como um servidor de aplicativos compatível com Java EE ou Jakarta EE , que sempre possui uma transação JTA em andamento.

Por exemplo, nunca é possível chamar `javax.jms.Session.commit()` no IMS, porque os estados de especificação JMS que você não pode chamar em um EJB JEE ou contêiner da Web, enquanto uma transação JTA estiver em andamento.

Isso resulta nas restrições a seguir para a API JMS, além das descritas em "Comportamento transacional" na página 349.

Restrições de API clássica

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` sempre lança um `JMSException`..
- O `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` sempre lança um `JMSException`
- As três variantes de `javax.jms.Connection.createSession` sempre lançam uma `JMSException` se a conexão já tem uma sessão ativa existente.
- O `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` sempre lança um `JMSException`
- O `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` sempre lança um `JMSException`
- O `javax.jms.Connection.setClientID()` sempre lança um `JMSException`
- O `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` sempre lança um `JMSException`
- O `javax.jms.Connection.stop()` sempre lança um `JMSException`
- O `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` sempre lança um `JMSException`

- O `javax.jms.MessageConsumer.getMessageListener()` sempre lança um `JMSException`
- O `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` sempre lança um `JMSException`
- O `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` sempre lança um `JMSException`
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` sempre lança um `JMSException`..
- O `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener )` sempre lança um `JMSException`
- O `javax.jms.Session.run()` sempre lança um `JMSRuntimeException`
- O `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` sempre lança um `JMSException`
- O `javax.jms.Session.getMessageListener()` sempre lança um `JMSException`

Restrições de API simplificadas

- O `javax.jms.JMSContext.createContext(int)` sempre lança um `JMSRuntimeException`
- O `javax.jms.JMSContext.setClientID(String)` sempre lança um `JMSRuntimeException`
- O `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` sempre lança um `JMSRuntimeException`
- O `javax.jms.JMSContext.stop()` sempre lança um `JMSRuntimeException`
- O `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` sempre lança um `JMSRuntimeException`

Usando o IBM MQ classes for Java

Use o IBM MQ em um ambiente Java. O IBM MQ classes for Java permite que um aplicativo Java se conecte ao IBM MQ como um cliente IBM MQ ou se conecte diretamente a um gerenciador de filas do IBM MQ.

Nota:

Stabilized A IBM não fará aprimoramentos adicionais no IBM MQ classes for Java e ele ficará funcionalmente estabilizado no nível enviado no IBM MQ 8.0. Os aplicativos existentes que usam o IBM MQ classes for Java continuam sendo totalmente suportados, mas novos recursos não serão incluídos e as solicitações de aprimoramentos serão rejeitadas. "Totalmente suportado" significa que os defeitos serão corrigidos, além de quaisquer mudanças necessárias, por meio de mudanças nos requisitos do sistema IBM MQ.

O IBM MQ classes for Java não é suportado no IMS.

O IBM MQ classes for Java não é suportado no WebSphere Liberty. Eles não devem ser usados com o recurso do sistema de mensagens do IBM MQ Liberty ou com o suporte genérico do JCA. Para obter mais informações, consulte Usando as interfaces Java do WebSphere MQ em ambientes J2EE/JEE.

IBM MQ classes for Java é uma das três APIs alternativas que os aplicativos Java podem usar para acessar recursos do IBM MQ . As outras APIs são:

- JM 3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

Para obter informações adicionais, consulte “Acessando IBM MQ de Java -Opção de API” na página 87.

No IBM MQ 9.3, o IBM MQ classes for Java é construído com o Java 8. O ambiente de tempo de execução do Java 8 suporta a execução das versões anteriores do arquivo de classe.

O IBM MQ classes for Java contém a Message Queue Interface (MQI), a API nativa do IBM MQ, e usa um modelo de objeto semelhante às interfaces C++ e .NET para o IBM MQ.

Opções programáveis permitem que o IBM MQ classes for Java se conecte ao IBM MQ de uma das maneiras a seguir:

- No modo de cliente como um IBM MQ MQI client, usando o Protocolo de Controle de Transmissões/ Protocolo da Internet (TCP/IP)
- No modo de ligações, conectando-se diretamente ao IBM MQ usando a Java Native Interface (JNI)

Nota: A reconexão de cliente automática não é suportada pelo IBM MQ classes for Java.

Conexão do modo cliente

Um aplicativo IBM MQ classes for Java pode se conectar a qualquer gerenciador de filas suportado usando o modo de cliente.

Para se conectar a um gerenciador de filas no modo cliente, um aplicativo IBM MQ classes for Java pode ser executado no mesmo sistema no qual o gerenciador de filas está em execução ou em um sistema diferente. Em cada caso, o IBM MQ classes for Java se conecta ao gerenciador de filas por meio de TCP/IP.

Para obter mais informações sobre como gravar aplicativos para usar conexões no modo cliente, consulte “Modos de conexão do IBM MQ classes for Java” na página 379.

Conexão do modo de ligações

Quando usado no modo de ligações, o IBM MQ classes for Java usa a Java Native Interface (JNI) para chamar diretamente para a API do gerenciador de filas existente, em vez de se realizar a comunicação por uma rede. Na maioria dos ambientes, a conexão no modo de ligações fornece melhor desempenho para aplicativos IBM MQ classes for Java do que a conexão no modo cliente, evitando o custo de comunicação TCP/IP.

Aplicativos que usam a IBM MQ classes for Java para se conectarem no modo de ligações devem ser executados no mesmo sistema que o gerenciador de filas ao qual estão se conectando.

O Java Runtime Environment, que está sendo usado para executar o aplicativo IBM MQ classes for Java, deve ser configurado para carregar as bibliotecas do IBM MQ classes for Java; consulte “IBM MQ classes for Java bibliotecas” na página 364 para obter informações adicionais.

Para obter mais informações sobre como gravar aplicativos para usar conexões do modo de ligações, consulte “Modos de conexão do IBM MQ classes for Java” na página 379.

Conceitos relacionados

Interfaces de linguagem do IBM MQ Java

“Usando IBM MQ classes for JMS/Jakarta Messaging” na página 83

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

Tarefas relacionadas

Rastreando aplicativos do IBM MQ classes for Java

Resolução de problemas Java e JMS

Por que devo usar o IBM MQ classes for Java?

Um aplicativo Java pode usar o IBM MQ classes for Java ou o IBM MQ classes for JMS para acessar os recursos do IBM MQ.

Nota: Embora os aplicativos existentes que usam o IBM MQ classes for Java continuem sendo totalmente suportados, novos aplicativos devem usar o IBM MQ classes for Jakarta Messaging. Os recursos que

foram recentemente incluídos no IBM MQ, como consumo assíncrono e reconexão automática, não estão disponíveis no IBM MQ classes for Java, mas estão disponíveis no IBM MQ classes for JMS e no IBM MQ classes for Jakarta Messaging Para obter mais informações, consulte “Por que devo usar o IBM MQ classes for JMS?” na página 86 e “Por que devo usar o IBM MQ classes for Jakarta Messaging?” na página 84.

Nota:

Stabilized A IBM não fará aprimoramentos adicionais no IBM MQ classes for Java e ele ficará funcionalmente estabilizado no nível enviado no IBM MQ 8.0. Os aplicativos existentes que usam o IBM MQ classes for Java continuam sendo totalmente suportados, mas novos recursos não serão incluídos e as solicitações de aprimoramentos serão rejeitadas. "Totalmente suportado" significa que os defeitos serão corrigidos, além de quaisquer mudanças necessárias, por meio de mudanças nos requisitos do sistema IBM MQ.

O IBM MQ classes for Java não é suportado no IMS.

O IBM MQ classes for Java não é suportado no WebSphere Liberty. Eles não devem ser usados com o recurso do sistema de mensagens do IBM MQ Liberty ou com o suporte genérico do JCA. Para obter mais informações, consulte Usando as interfaces Java do WebSphere MQ em ambientes J2EE/JEE.

Conceitos relacionados

“Acessando IBM MQ de Java -Opção de API” na página 87
IBM MQ fornece três interfaces de linguagem Java .

Pré-requisitos para o IBM MQ classes for Java

Para usar o IBM MQ classes for Java, você precisa de outros produtos de software determinados.

Para obter informações sobre os pré-requisitos para o IBM MQ classes for Java, veja a página da web do Requisitos do sistema para IBM MQ.

Para desenvolver aplicativos IBM MQ classes for Java, você precisa de um Java Development Kit (JDK). Os detalhes dos JDKs suportados com o sistema operacional podem ser localizados nas informações do Requisitos do sistema para IBM MQ.

Para executar aplicativos do IBM MQ classes for Java, é necessário ter os componentes de software a seguir:

- Um gerenciador de filas do IBM MQ para aplicativos que se conectam a um gerenciador de filas
- Um Java Runtime Environment (JRE), para cada sistema no qual você executa aplicativos. Um JRE adequado é fornecido com o IBM MQ.
- IBM i Para o IBM i, QShell, que é a opção 30 do sistema operacional
- z/OS Para o z/OS, z/OS UNIX System Services (z/OS UNIX)

Se conexões TLS forem requeridas para usar módulos criptográficos que foram certificados pelo FIPS 140-2, será necessário o provedor do IBM Java JSSE FIPS (IBMJSSEFIPS). Cada IBM JDK e JRE na versão 1.4.2 ou mais recente contém IBMJSSEFIPS.

É possível usar Internet Protocol versão 6 (IPv6) endereços em seus IBM MQ classes for Java aplicativos se IPv6 for suportado por sua Java máquina virtual (JVM) e a implementação TCP/IP em seu sistema operacional.

Executando aplicativos IBM MQ classes for Java no Java EE

Há determinadas restrições e considerações de design que devem ser levadas em conta antes de usar o IBM MQ classes for Java no Java EE.

O IBM MQ classes for Java tem restrições quando usado em um ambiente do Java Platform, Enterprise Edition (Java EE). Também há considerações adicionais que devem ser levadas em conta ao projetar, implementar e gerenciar um aplicativo IBM MQ classes for Java que é executado dentro de um ambiente do Java EE. Essas restrições e contraprestações são descritas nas seções a seguir.

Restrições de transações JTA

O único gerenciador de transações suportado para aplicativos usando o IBM MQ classes for Java é o próprio IBM MQ. Embora um aplicativo no controle de JTA possa fazer uso do IBM MQ classes for Java, qualquer trabalho executado através dessas classes não será controlado pelas unidades de trabalho da JTA. Ao invés disso, elas formam as unidades de trabalho locais a partir dessas gerenciadas pelo servidor de aplicativos através das interfaces de JTA. Em especial, qualquer retrocesso da transação de JTA não resulta em um retrocesso de quaisquer mensagens enviadas ou recebidas. Esta restrição se aplica a transações de aplicativo ou gerenciadas por bean e a transações gerenciadas por contêiner e todos os contêineres do Java EE. Para executar o trabalho do sistema de mensagens diretamente com o IBM MQ dentro das transações coordenadas pelo servidor do aplicativo, o IBM MQ classes for JMS deve ser usado no lugar.

Criação de encadeamento

IBM MQ classes for Java cria encadeamentos internamente para diversas operações. Por exemplo, quando em execução no modo BINDINGS para chamar diretamente em um gerenciador de filas local, as chamadas serão feitas em um encadeamento 'trabalhador' criado internamente pelo IBM MQ classes for Java. Outros encadeamentos podem ser criados internamente, por exemplo, para limpar conexões não usadas a partir de um conjunto de conexões ou para remover assinaturas para aplicativos de publicação/assinatura finalizados.

Alguns aplicativos Java EE (por exemplo, aqueles em execução em contêineres EJB e de web) não devem criar novos encadeamentos. Em vez disso, todo o trabalho deve ser executado nos encadeamentos do aplicativo principal gerenciado pelo servidor de aplicativos. Quando os aplicativos usar o IBM MQ classes for Java, o servidor de aplicativos poderá não ser capaz de distinguir entre o código de aplicativo e o código do IBM MQ classes for Java, portanto, os encadeamentos descritos anteriormente fazem com que o aplicativo não seja compatível com a especificação do contêiner. IBM MQ classes for JMS não interrompe essas especificações do Java EE e, portanto, pode ser usado no lugar.

Restrições de segurança

As políticas de segurança implementadas por um servidor de aplicativos podem evitar que determinadas operações sejam realizadas pela API do IBM MQ classes for Java, como criar e operar novos encadeamentos de controle (conforme descrito nas seções anteriores).

Por exemplo, servidores de aplicativos geralmente são executados com o Java Security desativado por padrão e permite que seja ativado através de alguma configuração específica do servidor de aplicativo (alguns servidores de aplicativos também permitem a configuração mais detalhada das políticas usadas dentro do Java Security). Quando o Java Security estiver ativado, o IBM MQ classes for Java poderá interromper as regras de encadeamento da política do Java Security definidas para o servidor de aplicativos e a API não pode ser capaz de criar todos os encadeamentos necessários para funcionar. Para evitar problemas com o gerenciamento de encadeamentos, o uso de IBM MQ classes for Java não é suportado em ambientes em que o Java Security está ativado.

Contraprestações de isolamento do aplicativo

Um benefício desejado da execução de aplicativos em um ambiente do Java EE é o isolamento de aplicativos. O design e a implementação do IBM MQ classes for Java anterior ao ambiente Java EE . IBM MQ classes for Java pode ser usado de forma que não suporte o conceito de isolamento de aplicativos. Os exemplos de contraprestações específicas nesta área incluem:

- O uso de configurações estáticas (todo o processo da JVM) dentro da classe MQEnvironment, como:
  - o ID do usuário e a senha a serem usados para autenticação e identificação de conexão
  - o nome do host, porta e canal usados para conexões do cliente
  - Configuração TLS para conexões do cliente asseguradas

  A modificação de qualquer uma das propriedades MQEnvironment para o benefício de um aplicativo também afeta outros aplicativos usando as mesmas propriedades. Ao executar em um ambiente com

vários aplicativos, como Java EE, cada aplicativo deve usar sua própria configuração distinta por meio da criação de objetos MQQueueManager com um conjunto específico de propriedades, em vez de padronizar para as propriedades configuradas na classe MQEnvironment em todo o processo.

- A classe MQEnvironment apresenta uma série de métodos estáticos que agem globalmente em todos os aplicativos que usam o IBM MQ classes for Java dentro do mesmo processo da JVM e não há uma maneira de substituir esse comportamento por aplicativos específicos. Exemplos incluem:
  - configurando propriedades TLS, como o local do keystore
  - configurando as saídas de canais do cliente
  - ativando ou desativando o rastreio de diagnóstico
  - gerenciando o conjunto de conexões padrão usado para otimizar o uso de conexões para os gerenciadores de filas

  A chamada de tais métodos afeta todos os aplicativos em execução no mesmo ambiente do Java EE.
- A definição do conjunto de conexões é ativada para otimizar o processo de fazer várias conexões no mesmo gerenciador de filas. O gerenciador de conjunto de conexões padrão é todo o processo e compartilhado por vários aplicativos. As mudanças na configuração do conjunto de conexões, como substituir o gerenciador de conexão padrão por um aplicativo usando o método MQEnvironment.setDefaultConnectionManager(), portanto, afeta outros aplicativos em execução no mesmo servidor de aplicativos Java EE.
- O TLS é configurado para aplicativos que usam o IBM MQ classes for Java usando a classe MQEnvironment e as propriedades de objeto MQQueueManager. Ele não está integrado com a configuração de segurança gerenciada do próprio servidor de aplicativos. Deve-se assegurar-se de configurar o IBM MQ classes for Java corretamente para fornecer o nível de segurança necessário e não usar a configuração do servidor de aplicativos.

Restrições do modo de ligações

IBM MQ e WebSphere Application Server podem ser instalados na mesma máquina, de modo que as versões principais do gerenciador de filas e do adaptador de recursos do IBM MQ (RA) fornecidas no WebSphere Application Server sejam diferentes.

Se as versões principais do adaptador de recursos e do gerenciador de filas forem diferentes, as conexões de ligações não poderão ser usadas. Quaisquer conexões a partir do WebSphere Application Server para o gerenciador de filas usando o adaptador de recursos devem usar conexões do tipo de cliente. As conexões de ligações poderão ser usadas se as versões forem as mesmas.

Conversões de sequências de caracteres no IBM MQ classes for Java

Os IBM MQ classes for Java usam CharsetEncoders e CharsetDecoders diretamente para conversão de sequência de caracteres. O comportamento padrão para a conversão de sequência de caracteres pode ser configurado com duas propriedades do sistema. A manipulação de mensagens que contêm caracteres não mapeáveis pode ser configurada por meio de `com.ibm.mq.MQMD`.

Antes do IBM MQ 8.0, as conversões de sequências no IBM MQ classes for Java foram feitas chamando os métodos `java.nio.charset.Charset.decode(ByteBuffer)` e `Charset.encode(CharBuffer)`.

Ao usar um desses métodos resultará em uma substituição padrão (REPLACE) de dados malformados ou não convertidos. Esse comportamento pode ocultar erros em aplicativos e conduzir a caracteres inesperados, por exemplo ?, em dados traduzidos.

A partir de IBM MQ 8.0, para detectar tais problemas mais cedo e de forma mais eficaz, os IBM MQ classes for Java usam CharsetEncoders e CharsetDecoders diretamente e configuram o manuseio de dados malformados e intraduzíveis explicitamente. O comportamento padrão é para REPORT tais problemas ao lançar um `MQException` adequado.

Configurando

Converter de UTF-16 (a representação de caracteres usada no Java) para um conjunto de caracteres nativos, como UTF-8, é denominado *encoding*, enquanto converter na direção oposta é denominado *decoding*.

Decodificação assume o comportamento padrão para `CharsetDecoders`, relatando erros ao lançar uma exceção.

Uma configuração é usada para especificar um `java.nio.charset.CodingErrorAction` para controlar a manipulação de erros na codificação e decodificação. Uma outra configuração é usada para controlar o byte de substituição, ou bytes, ao codificar. A sequência de substituição padrão do Java será usada em operações de decodificação.

Configuração de manipulação de dados intraduzíveis em IBM MQ classes for Java

A partir de IBM MQ 8.0, `com.ibm.mq.MQMD` inclui os dois campos a seguir:

byte[] unMappableReplacement
A sequência de bytes que serão gravados em uma sequência codificada se um caractere de entrada não puder ser convertido e você tiver especificado `REPLACE`.

Padrão: "?". getBytes ()
A sequência de substituição padrão do Java é usada em operações de decodificação.

java.nio.charset.CodingErrorAction unmappableAction
Especifica a ação a ser executada para os dados não convertidos na codificação e decodificação:

Padrão: CodingErrorAction.REPORT;

Propriedades do sistema para configuração de padrões do sistema

No IBM MQ 8.0, as duas propriedades do sistema Java a seguir estão disponíveis para configurar o comportamento padrão sobre conversão de sequência de caracteres.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Especifica a ação a ser executada para os dados não convertidos na codificação e decodificação. O valor pode ser `REPORT`, `REPLACE` ou `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Configura ou obtém os bytes de substituição para aplicar quando um caractere não puder ser mapeado em uma operação de codificação. A sequência de substituição padrão do Java é usada em operações de decodificação.

Para evitar confusão entre as representações de byte nativo e caractere do Java, é necessário especificar `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` como um número decimal representando o byte de substituição no conjunto de caracteres nativo.

Por exemplo, o valor decimal de ?, como um byte nativo, será 63 se o conjunto de caracteres nativo for baseado em ASCII, como ISO-8859-1, enquanto que será 111, se o conjunto de caracteres nativo for EBCDIC.

Nota: Observe que se um objeto MQMD ou MQMessage tiver os campos **`unmappableAction`** ou **`unMappableReplacement`** configurados, os valores desses campos terão precedência sobre as propriedades do sistema Java. Isso permite que os valores especificados pelas propriedades do sistema Java sejam substituídos para cada mensagem, se necessário.

Conceitos relacionados
“Conversões de sequências de caracteres no IBM MQ classes for JMS” na página 140
Os IBM MQ classes for JMS usam CharsetEncoders e CharsetDecoders diretamente para conversão de sequência de caracteres. O comportamento padrão para a conversão de sequência de caracteres pode ser configurado com duas propriedades do sistema. A manipulação de mensagens que contêm caracteres não mapeáveis pode ser configurada por meio de propriedades da mensagem para configurar o UnmappableCharacterAction e os bytes de substituição.

Instalando e configurando o IBM MQ classes for Java

Esta seção descreve os diretórios e arquivos criados ao instalar o IBM MQ classes for Java e informará como configurar o IBM MQ classes for Java após a instalação.

O que é instalado para IBM MQ classes for Java

A versão mais recente do IBM MQ classes for Java é instalada com o IBM MQ. Talvez seja necessário substituir opções de instalação padrão para assegurar que isso seja feito.

Para obter informações adicionais sobre como instalar o IBM MQ, consulte:

- 

Instalando o IBM MQ
- 

Instalando o IBM MQ for z/OS produto

IBM MQ classes for Java estão contidos nos arquivos Java archive (JAR), `com.ibm.mq.jar` e `com.ibm.mq.jmqi.jar`.

O suporte para cabeçalhos de mensagem padrão, como o formato de comando programável (PCF), está contido no arquivo JAR `com.ibm.mq.headers.jar`.

O suporte para o formato de comando programável (PCF) está contido no arquivo JAR `com.ibm.mq.pcf.jar`.

Nota: Não é recomendado usar o IBM MQ classes for Java em um servidor de aplicativos. Para obter informações sobre as restrições que se aplicam quando em execução nesse ambiente, veja “Executando aplicativos IBM MQ classes for Java no Java EE” na página 355. Para obter mais informações, consulte Usando Interfaces do WebSphere MQ Java em J2EE/JEE .

Importante: Além dos arquivos JAR realocáveis descritos em “IBM MQ classes for Java arquivos JAR realocáveis” na página 359, não é suportada a cópia dos arquivos JAR ou das bibliotecas nativas do IBM MQ classes for Java em outras máquinas ou em um local diferente de uma máquina na qual o IBM MQ classes for Java foi instalado.

IBM MQ classes for Java arquivos JAR realocáveis

Os arquivos JAR realocáveis podem ser movidos para sistemas que precisam executar o IBM MQ classes for Java.

Importante:

- Além dos arquivos JAR realocáveis descritos em Arquivos JAR realocáveis, não é suportada a cópia dos arquivos JAR ou das bibliotecas nativas do IBM MQ classes for Java em outras máquinas ou em um local diferente de uma máquina na qual o IBM MQ classes for Java foi instalado.
- Para evitar conflitos do carregador de classes, não é recomendado empacotar os arquivos JAR realocáveis dentro de vários aplicativos dentro do mesmo tempo de execução de Java. Neste cenário, considere tornar os arquivos JAR realocáveis do IBM MQ disponíveis no caminho de classe do tempo de execução do Java.
- Não inclua os arquivos JAR realocáveis dentro de aplicativos implementados em servidores de aplicativos Java EE, como o WebSphere Application Server. Nesses ambientes, o adaptador de recursos do IBM MQ deve ser implementado e usado como alternativa, pois ele contém o IBM MQ classes for Java. Observe que o WebSphere Application Server integra o adaptador de recursos do IBM MQ, portanto, não há necessidade de implementá-lo manualmente nesse ambiente. Além disso, os IBM MQ classes for Java não são suportados no WebSphere Liberty. Para obter mais informações, consulte “Liberty e o adaptador de recursos do IBM MQ” na página 450.
- Se você estiver empacotando os arquivos JAR realocáveis dentro de seus aplicativos, assegure-se de incluir todos os arquivos JAR de pré-requisito, conforme descrito em Arquivos JAR realocáveis. Também é necessário garantir os procedimentos apropriados para atualizar os arquivos JAR empacotados como parte da manutenção do aplicativo, a fim de assegurar que os IBM MQ classes for Java permaneçam atuais e que os problemas conhecidos sejam remediados.

Arquivos JAR relocalizáveis

Dentro de uma empresa, os seguintes arquivos podem ser movidos para os sistemas que precisam executar aplicativos IBM MQ classes for Java:

- JMS 2.0 `com.ibm.mq.allclient.jar` "1" na página 360
- JM 3.0 `com.ibm.mq.jakarta.client.jar` "2" na página 360
- V 9.4.0 `bcpkix-jdk18on.jar` "3" na página 360
- `bcpkix-jdk15to18.jar` "4" na página 360
- V 9.4.0 `bcprov-jdk18on.jar` "3" na página 360
- `bcprov-jdk15to18.jar` "4" na página 360
- V 9.4.0 `bcutil-jdk18on.jar` "3" na página 360
- `bcutil-jdk15to18.jar` "4" na página 360
- `org.json.jar`

Notas:

1. JMS 2.0 e JMS 1.1
2. Jakarta Messaging 3.0
3. De IBM MQ 9.4.0
4. AntesIBM MQ 9.4.0

O provedor de segurança Bouncy Castle e o CMS suportam arquivos JAR

O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS são necessários. Para mais informações, consulte o Suporte para JREs não-IBM com AMS.

V 9.4.0 São necessários os seguintes arquivos JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

O arquivo `org.json.jar` é necessário se o seu aplicativo IBM MQ classes for Java usa um CCDT em formato JSON.

com.ibm.mq.allclient.jar e com.ibm.mq.jakarta.client.jar

Os arquivos `com.ibm.mq.allclient.jar` e `com.ibm.mq.jakarta.client.jar` contêm o IBM MQ classes for JMS, o IBM MQ classes for Javae as classes PCF e de cabeçalhos. Se você mover esses arquivos para um novo local, certifique-se de executar as etapas para manter esse novo local mantido com novos Fix Packs do IBM MQ . Além disso, certifique-se de que o uso dos arquivos seja conhecido para o Suporte IBM se você estiver obtendo uma correção temporária

Para determinar a versão do arquivo `com.ibm.mq.allclient.jar` ou do arquivo `com.ibm.mq.jakarta.client.jar` , use o comando a seguir:

JM 3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

O exemplo a seguir mostra uma saída de amostra desse comando:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Diretórios de instalação para o IBM MQ classes for Java
Arquivos e amostras doIBM MQ classes for Java estão instalados em locais diferentes de acordo com a plataforma. O local do Java Runtime Environment (JRE) que é instalado com o IBM MQ também varia de acordo com a plataforma.

Diretórios de instalação para arquivos do IBM MQ classes for Java

O Tabela 49 na página 361 mostra onde os arquivos do IBM MQ classes for Java são instalados.

Tabela 49. Diretórios de instalação do IBM MQ classes for Java

Plataforma	**Diretório**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib
Linux Linux	*MQ_INSTALLATION_PATH*/java/lib
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

os diretórios de instalação para amostras

Alguns aplicativos de amostra, como o Installation Verification Programs (IVPs), são fornecidos com o IBM MQ. O Tabela 50 na página 362 mostra onde os aplicativos de amostra são instalados. As amostras IBM MQ classes for Java estão em um subdiretório chamado `wmqjava`. As amostras PCF estão em um subdiretório chamado `pcf`.

Tabela 50. Diretórios de amostras	
Plataforma	**Diretório**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i	/QIBM/ProdData/mqm/java/samples
Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

diretórios de instalação para o JRE

IBM MQ classes for JMS requer um Java 7 (ou superior) Java Runtime Environment (JRE). Um JRE adequado é instalado com o IBM MQ. O Tabela 51 na página 362 mostra onde este JRE é instalado. Para executar programas Java, como as amostras fornecidas, usando esse JRE, chame *JRE_LOCATION*`/bin/java` explicitamente ou inclua *JRE_LOCATION*`/bin` no ambiente PATH (ou equivalente) para a sua plataforma, em que *JRE_LOCATION* é o diretório fornecido em Tabela 51 na página 362.

Tabela 51. diretórios do JRE	
Plataforma	**Diretório**
AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i	/QIBM/ProdData/mqm/java/jre
Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Variáveis de ambiente relevantes para o IBM MQ classes for Java
Para executar aplicativos IBM MQ classes for Java, o caminho de classe deles deve incluir os diretórios do IBM MQ classes for Java e de amostra.

Para que os aplicativos IBM MQ classes for Java sejam executados, o caminho de classe deles deve incluir o diretório apropriado do IBM MQ classes for Java. Para executar os aplicativos de amostra, o caminho de classe também deve incluir os diretórios de amostras apropriados. Essas informações podem ser fornecidas no comando de chamada Java ou na variável de ambientes **CLASSPATH** .

Importante: A configuração da opção Java `-Xbootclasspath` para incluir o IBM MQ classes for Java não é suportada.

Tabela 52 na página 363 mostra a configuração **CLASSPATH** apropriada para usar em cada plataforma para executar aplicativos IBM MQ classes for Java , incluindo os aplicativos de amostra.

*Tabela 52. configuração **CLASSPATH** para executar aplicativos IBM MQ classes for Java , incluindo os aplicativos de amostra IBM MQ classes for Java*	
Plataforma	**CLASSPATH como definir**
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: . *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: . *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/samples/pcf

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Se compilar usando a opção `-Xlint`, você poderá ver uma mensagem avisando que o `com.ibm.mq.ese.jar` não está presente. É possível ignorar o aviso. Esse arquivo estará presente somente se você tiver instalado o Advanced Message Security.

Os scripts fornecidos com o IBM MQ classes for JMS usam as variáveis de ambiente a seguir:

MQ_JAVA_DATA_PATH
Essa variável de ambiente especifica o diretório para log e saída de rastreio.

MQ_JAVA_INSTALL_PATH
Essa variável de ambiente especifica o diretório em que os IBM MQ classes for Java estão instalados, conforme mostrado em Diretórios de instalação do IBM MQ classes for Java.

MQ_JAVA_LIB_PATH
Essa variável de ambiente especifica o diretório no qual as bibliotecas do IBM MQ classes for Java estão armazenadas, conforme mostrado em O local das bibliotecas do IBM MQ classes for Java para cada plataforma. Alguns scripts fornecidos com o IBM MQ classes for Java, como IVTRun, usam esta variável de ambiente.

Windows No Windows, todas as variáveis de ambiente são configuradas automaticamente durante a instalação.

Linux AIX No AIX and Linux, é possível usar o script `setjmsenv` (se você estiver usando uma JVM de 32 bits) ou `setjmsenv64` (se estiver usando uma JVM de 64 bits) para configurar as variáveis de ambiente. Estes scripts estão no diretório *MQ_INSTALLATION_PATH*/java/bin.

IBM i No IBM i, a variável de ambiente **QIBM_MULTI_THREADED** deve ser configurada como Y. Em seguida, é possível executar aplicativos multiencadeados da mesma maneira que você executa aplicativos de encadeamento único. Para obter mais informações, consulte Configurando IBM MQ com Java e JMS..

O IBM MQ classes for Java requer um Java 7 Java Runtime Environment (JRE). Para obter informações sobre o local de um JRE adequado instalado com o IBM MQ, consulte “Diretórios de instalação para o IBM MQ classes for Java” na página 361.

IBM MQ classes for Java bibliotecas
O local das bibliotecas do IBM MQ classes for Java varia de acordo com a plataforma. Especifique este local ao iniciar um aplicativo.

Para especificar o local das bibliotecas da Java Native Interface (JNI), inicie seu aplicativo usando um comando **java** com o formato a seguir:

```
java -Djava.library.path= library_path application_name
```

em que *library_path* é o caminho para o IBM MQ classes for Java, que inclui as bibliotecas da JNI. Tabela 53 na página 364 mostra o local das bibliotecas do IBM MQ classes for Java para cada plataforma. Nesta tabela, *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Tabela 53. O local das bibliotecas do IBM MQ classes for Java para cada plataforma

Plataforma	**Diretório que contém as bibliotecas do IBM MQ classes for Java**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Linux Linux (plataforma x86)	*MQ_INSTALLATION_PATH*/java/lib
Linux Linux (plataformas POWER, x86-64 e zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\java\lib64 (bibliotecas de 64 bits)
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (bibliotecas de 32 bits e de 64 bits)

Nota:

1. Linux AIX No AIX ou no Linux (plataforma Power), use as bibliotecas de 32 ou 64 bits. Use as bibliotecas de 64 bits somente se você estiver executando seu aplicativo em uma Java virtual machine (JVM) de 64 bits em uma plataforma de 64 bits. Caso contrário, use as bibliotecas de 32 bits.
2. Windows No Windows, é possível usar a variável de ambiente PATH para especificar o local das bibliotecas do IBM MQ classes for Java em vez de especificar a localização das mesmas no comando **java**.
3. IBM i Para usar o IBM MQ classes for Java no modo de ligações no IBM i, assegure que a biblioteca QMQMJAVA esteja em sua lista de bibliotecas.
4. z/OS No z/OS, é possível usar uma Java virtual machine (JVM) de 32 bits ou de 64 bits. Você não precisa especificar quais bibliotecas usar; o IBM MQ classes for Java pode determinar para ele mesmo quais bibliotecas da JNI carregar.

Conceitos relacionados
Usando o IBM MQ classes for Java

Após instalar o IBM MQ classes for Java, será possível configurar sua instalação para que seus próprios aplicativos sejam executados.

Suporte para OSGi com o IBM MQ classes for Java
OSGi fornece uma estrutura que suporta a implementação de aplicativos como pacotes configuráveis. Três pacotes configuráveis do OSGi são fornecidos como parte do IBM MQ classes for Java.

O OSGi fornece uma estrutura Java de propósito geral, segura e gerenciada, que suporta a implementação de aplicativos fornecidos na forma de pacotes configuráveis. Os dispositivos compatíveis com o OSGi poderão fazer download e instalar pacotes configuráveis, e removê-los quando não forem mais necessários. A estrutura gerencia a instalação e a atualização de pacotes configuráveis de um modo dinâmico e escalável.

O IBM MQ classes for Java inclui os seguintes pacotes configuráveis do OSGi.

com.ibm.mq.osgi.java_*version_number*.jar
Os arquivos JAR para permitir que os aplicativos usem o IBM MQ classes for Java.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 Para Jakarta Messaging 3.0, esse arquivo JAR permite que os aplicativos usem o IBM MQ classes for JMS e o IBM MQ classes for Javae também inclui o código para manipular mensagens PCF.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Para JMS 2.0, esse arquivo JAR permite que aplicativos usem o IBM MQ classes for JMS e o IBM MQ classes for Javae também inclui o código para manipular mensagens PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 Para Jakarta Messaging 3.0, esse arquivo JAR fornece os pré-requisitos para `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Para JMS 2.0, esse arquivo JAR fornece os pré-requisitos para `com.ibm.mq.osgi.allclient_version_number.jar`.

em que *version_number* é o número da versão de IBM MQ que está instalado

Os bundles são instalados no subdiretório `java/lib/OSGi` da sua instalação IBM MQ, ou a pasta `java\lib\OSGi` no Windows.

A partir de IBM MQ 8.0, use os bundles `com.ibm.mq.osgi.allclient_8.0.0.0.jar`, e `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` para quaisquer novos aplicativos. O uso desses pacotes configuráveis remove a restrição de não ser capaz de executar ambos IBM MQ classes for JMS e o IBM MQ classes for Java dentro da mesma estrutura do OSGi. No entanto, todas as outras restrições ainda se aplicam. Para versões do IBM MQ antes do IBM MQ 8.0, a restrição do uso de um IBM MQ classes for JMS ou IBM MQ classes for Java, se aplicará.

Outros nove pacotes configuráveis também são instalados no sub-diretório `java/lib/OSGi` da instalação do IBM MQ na pasta `java\lib\OSGi` no Windows. Esses pacotes configuráveis são parte do IBM MQ classes for JMS e não devem ser carregados em um ambiente de tempo de execução do OSGi que tenha o pacote configurável IBM MQ classes for Java carregado. Se o pacote configurável do OSGi IBM MQ classes for Java for carregado em um ambiente de tempo de execução do OSGi que também tem os pacotes configuráveis IBM MQ classes for JMS carregados, os erros, conforme mostrado no exemplo a seguir, ocorrerão quando os aplicativos que usam o pacote configurável IBM MQ classes for Java ou os pacotes configuráveis IBM MQ classes for JMS forem executados:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

O pacote configurável do OSGi para o IBM MQ classes for Java foi gravado na especificação da liberação 4 do OSGi; ele não funciona em um ambiente de liberação 3 do OSGi.

Deve-se configurar seu caminho de sistema ou caminho da biblioteca corretamente para que o ambiente de tempo de execução do OSGi possa localizar quaisquer arquivos da DLL ou bibliotecas compartilhadas requeridas.

Se você usar o pacote configurável do OSGi para o IBM MQ classes for Java, as classes de saída do canal gravadas em Java não serão suportadas devido a um problema inerente em classes de carregamento em um ambiente múltiplo de carregador de classes como OSGi. Um pacote configurável do usuário pode estar ciente do pacote configurável IBM MQ classes for Java, mas o pacote configurável IBM MQ classes for Java não está ciente de qualquer pacote configurável do usuário. Como resultado, o carregador de classes usado em um pacote configurável do IBM MQ classes for Java não pode carregar uma classe de saída do canal que está em um pacote configurável do usuário.

Para obter mais informações sobre o OSGi, consulte o website do OSGi Alliance.

z/OS *Installation of IBM MQ classes for Java on z/OS*

On z/OS, the STEPLIB used at runtime must contain the IBM MQ SCSQAUTH and SCSQANLE libraries.

From z/OS UNIX System Services, you can add these libraries by using a line in your `.profile` as shown in the following example, replacing `thlqual` with the high level data set qualifier that you chose when installing IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In other environments, you typically need to edit the startup JCL to include SCSQAUTH on the STEPLIB concatenation:

```
 STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

O arquivo de configuração do IBM MQ classes for Java

Um arquivo de configuração do IBM MQ classes for Java especifica propriedades que são usadas para configurar o IBM MQ classes for Java.

O formato de um arquivo de configuração do IBM MQ classes for Java é o de um arquivo de propriedades padrão do Java.

Um arquivo de configuração de amostra, `mqjava.config`, é fornecido no subdiretório `bin` do diretório de instalação IBM MQ classes for Java. Este arquivo documenta todas as propriedades compatíveis e seus valores padrão.

Nota: O arquivo de configuração de amostra é sobrescrito quando a instalação do IBM MQ é submetida a upgrade para um futuro Fix Pack. Portanto, é recomendável que você faça uma cópia do arquivo de configuração de amostra para uso com seus aplicativos.

É possível escolher o nome e o local de um arquivo de configuração do IBM MQ classes for Java. Ao iniciar o seu aplicativo, use um comando **java** com o seguinte formato:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

No comando, *config_file_url* é um Localizador Uniforme de Recursos (URL) que especifica o nome e o local do arquivo de configuração do IBM MQ classes for Java. As URLs dos tipos a seguir são suportadas: `http`, `file`, `ftp` e `jar`.

O exemplo a seguir mostra um comando **java**:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

Este comando identifica o arquivo de configuração IBM MQ classes for Java como o arquivo `D:\mydir\mqjava.config` no sistema Windows local.

Quando um aplicativo for iniciado, o IBM MQ classes for Java lê o conteúdo do arquivo de configuração e armazena as propriedades especificadas em um armazenamento de propriedade interna. Se o comando **java** não identificar um arquivo de configuração ou se o arquivo de configuração não puder ser localizado, o IBM MQ classes for Java usará os valores padrão para todas as propriedades. Se necessário, será possível substituir qualquer propriedade no arquivo de configuração especificando-a como uma propriedade de sistema no comando **java**.

Um arquivo de configuração do IBM MQ classes for Java pode ser usado com qualquer um dos transportes suportados entre um aplicativo e um gerenciador de filas ou broker.

Substituindo propriedades especificadas em um arquivo de configuração do IBM MQ MQI client

Um arquivo de configuração do IBM MQ MQI client também pode especificar propriedades que são usadas para configurar o IBM MQ classes for Java. No entanto, as propriedades que são especificadas em um arquivo de configuração do IBM MQ MQI client se aplicam somente quando um aplicativo se conecta a um gerenciador de filas no modo cliente.

Se necessário, será possível substituir qualquer atributo em um arquivo de configuração do IBM MQ MQI client especificando-o como uma propriedade em um arquivo de configuração do IBM MQ classes for Java. Para substituir um atributo em um arquivo de configuração do IBM MQ MQI client, use uma entrada com o formato a seguir no arquivo de configuração do IBM MQ classes for Java:

```
com.ibm.mq.cfg.stanza.propName=propValue
```

As variáveis na entrada possuem os seguintes significados:

stanza
O nome da sub-rotina no arquivo de configuração do IBM MQ MQI client que contém o atributo.

propName
O nome do atributo, conforme especificado no arquivo de configuração do IBM MQ MQI client.

propValue
O valor da propriedade que substitui o valor do atributo que é especificado no arquivo de configuração do IBM MQ MQI client.

Como alternativa, é possível substituir um atributo em um arquivo de configuração do IBM MQ MQI client, especificando a propriedade como uma propriedade de sistema no comando **java**. Use o formato anterior para especificar a propriedade como uma propriedade de sistema.

Somente os atributos a seguir em um arquivo de configuração do IBM MQ MQI client são relevantes para IBM MQ classes for Java. Se você especificar ou substituir outros atributos, isso não terá efeito. Especificamente, observe que o `ChannelDefinitionFile` e o `ChannelDefinitionDirectory` na Sub-rotina CHANNELS do arquivo de configuração do cliente não são usados. Consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for Java” na página 383 para obter detalhes de como usar o CCDT com o IBM MQ classes for Java.

Tabela 54. Qual sub-rotina do arquivo de configuração do cliente contém qual atributo

Sub-rotina	**Atributo**
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente	Put1DefaultAlwaysSync
Sub-rotina CHANNELS do Arquivo de Configuração do Cliente	PasswordProtection
Sub-rotina ClientExitPath do arquivo de configuração do cliente	Caminho Padrão das Saídas
Sub-rotina ClientExitPath do arquivo de configuração do cliente	ExitsDefaultPath64

Tabela 54. Qual sub-rotina do arquivo de configuração do cliente contém qual atributo (continuação)	
Sub-rotina	**Atributo**
Sub-rotina ClientExitPath do arquivo de configuração do cliente	JavaExitsClasspath
Sub-rotina JMQI do arquivo de configuração do cliente	useMQCSPauthentication
Sub-rotina MessageBuffer do arquivo de configuração do cliente	MaximumSize
Sub-rotina MessageBuffer do arquivo de configuração do cliente	PurgeTime
Sub-rotina MessageBuffer do arquivo de configuração do cliente	UpdatePercentage
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntRcvBuffSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntSndBuffSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	Connect_Timeout
Sub-rotina TCP do Arquivo de Configuração do Cliente	KeepAlive

Para obter mais informações sobre a configuração IBM MQ MQI client , consulte o arquivo de configuração IBM MQ MQI client , `mqclient.ini`.

Tarefas relacionadas

Rastreando classes do IBM MQ para aplicativos Java

Usando o rastreio do Java Standard Environment Trace para configurar Java

Use a sub-rotina de Configurações de rastreio do ambiente Java Standard para configurar o recurso de rastreio do IBM MQ classes for Java.

`com.ibm.msg.client.commonservices.trace.outputName` = *`traceOutputName`*

`traceOutputName` é o nome do diretório e do arquivo ao qual a saída de rastreio é enviada.

Por padrão, as informações de rastreio são gravadas em um arquivo de rastreio no diretório atualmente em funcionamento do aplicativo. O nome do arquivo de rastreio depende do ambiente no qual o aplicativo está em execução:

- Se o aplicativo tiver carregado o IBM MQ classes for Java do arquivo JAR relocalizável `com.ibm.mq.allclient.jar`, o rastreio será gravado em um arquivo chamado `mqjavaclient_`*`%PID%`*`.cl`*`%u`*`.trc`.
- Se o aplicativo tiver carregado o IBM MQ classes for Java do arquivo JAR `com.ibm.mq.jar`, o rastreio será gravado em um arquivo chamado `mqjava_`*`%PID%`*`.cl`*`%u`*`.trc`.

em que *%PID%* é o identificador de processo do aplicativo que está sendo rastreado e *%u* é um número exclusivo para diferenciar arquivos entre os encadeamentos que executam rastreio sob diferentes carregadores de classes Java.

Se um ID de processo estiver indisponível, um número aleatório será gerado e prefixado com a letra `f`. Para incluir o ID do processo em um nome de arquivo você especificar, use a sequência `%PID%`.

Se você especificar um diretório alternativo, ele deve existir e você deve ter permissão de gravação para esse diretório. Se você não tiver permissão de gravação, a saída de rastreio será gravada para `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList é uma lista de pacotes e classes que são rastreados ou os valores especiais ALL ou NONE.

Nomes de classe ou pacote separados com um ponto e vírgula, ;. *includeList* é padronizado para ALL e rastreia todos os pacotes e classes em IBM MQ classes for Java.

Nota: É possível incluir um pacote, mas depois excluir os subpacotes desse pacote. Por exemplo, se você incluir o pacote a.b e excluir o pacote a.b.x, o rastreio inclui tudo no a.b.y e a.b.z, mas não a.b.x ou a.b.x.1.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList é uma lista de pacotes e classes que não são rastreados ou os valores especiais ALL ou NONE.

Nomes de classe ou pacote separados com um ponto e vírgula, ;. *excludeList* é padronizado para NONE e, portanto, não exclui pacotes e classes em IBM MQ classes for JMS de serem rastreados.

Nota: É possível excluir um pacote, mas depois incluir os subpacotes desse pacote. Por exemplo, se você excluir o pacote a.b e incluir o pacote a.b.x, o rastreio incluirá tudo em a.b.x e a.b.x.1, mas não a.b.y ou a.b.z.

Qualquer pacote ou classe que estiver especificado, no mesmo nível, como ambos incluídos e excluídos, será incluído.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes é o número máximo de bytes que são rastreados a partir de qualquer matriz de byte.

Se *maxArrayBytes* for configurado como um número inteiro positivo, ele limitará o número de bytes em uma matriz de bytes que é gravada no arquivo de rastreio. Ela trunca a matriz de bytes depois da gravação de *maxArrayBytes*. Configurar *maxArrayBytes* reduz o tamanho do arquivo de rastreio resultante e reduz o efeito de rastreio no desempenho do aplicativo.

Um valor 0 para esta propriedade significa que nenhum conteúdo de qualquer matriz de bytes é enviado para o arquivo de rastreio.

O valor padrão é -1, o que remove qualquer limite no número de bytes em uma matriz de bytes que são enviados para o arquivo de rastreio.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes é o número máximo de bytes que são gravados em um arquivo de saída de rastreio.

maxTraceBytes trabalha com *traceCycles*. Se o número de bytes de rastreio gravado estiver perto do limite, o arquivo será fechado e um novo arquivo de saída de rastreio será iniciado.

Um valor 0 significa que um arquivo de saída de rastreio tem o comprimento zero. O valor padrão é -1, o que significa que a quantidade de dados a serem gravados em um arquivo de saída de rastreio é ilimitada.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles é o número de arquivos de saída de rastreio para percorrer.

Se o arquivo de saída de rastreio atual atingir o limite especificado por *maxTraceBytes*, o arquivo será fechado. A saída de rastreio adicional é gravada para o próximo arquivo de saída de rastreio na sequência. Cada arquivo de saída de rastreio é distinto por um sufixo numérico anexado ao nome do arquivo. O arquivo de saída de rastreio atual ou mais recente é mqjms.trc.0, o próximo arquivo de saída de rastreio mais recente é mqjms.trc.1. Os arquivos de rastreio seguem o mesmo padrão de numeração até o limite.

O valor padrão de *traceCycles* é 1. Se *traceCycles* for 1, quando o arquivo de saída de rastreio atual atingir seu tamanho máximo, o arquivo será fechado e excluído. Um novo arquivo de saída de rastreio com o mesmo nome é iniciado. Portanto, existe só um arquivo de saída de rastreio por vez.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters controla se os parâmetros de método e valores de retorno são incluídos no rastreio.

traceParameters é padronizado para TRUE. Se *traceParameters* for configurado como FALSE, somente as assinaturas de método serão rastreadas.

com.ibm.msg.client.commonservices.trace.startup = *startup*
Há uma fase de inicialização do IBM MQ classes for Java durante a qual os recursos são alocados. O recurso de rastreio principal é inicializado durante a fase de alocação de recurso.

Se *startup* estiver configurado como TRUE, o rastreio de inicialização será usado. As informações de rastreio são produzidas imediatamente e incluem a configuração de todos os componentes, incluindo o próprio recurso de rastreio. As informações de rastreio de inicialização podem ser usadas para diagnosticar os problemas de configuração. As informações de rastreio de inicialização são sempre gravadas em `System.err`.

startup é padronizado para FALSE.

startup é verificado antes da inicialização ser concluída. Por essa razão, especifique apenas a propriedade na linha de comandos como uma propriedade do sistema Java. Não a especifique no arquivo de configuração do IBM MQ classes for Java.

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Configure *compressedTrace* para TRUE para compactar a saída de rastreio.

O valor padrão de *compressedTrace* é FALSE.

Se *compressedTrace* for configurado como TRUE, a saída de rastreio será compactada. O nome do arquivo de saída de rastreio padrão tem a extensão `.trz`. Se a compactação estiver configurada como FALSE, o valor padrão, o arquivo terá a extensão `.trc` para indicar que está descompactado. No entanto, se o nome do arquivo para a saída de rastreio tiver sido especificado em *traceOutputName*, esse nome será usado no lugar; nenhum sufixo será aplicado ao arquivo.

A saída de rastreio compactada é menor do que a descompactada. Como há menos E/S, isso pode ser gravado mais rápido do que o rastreio descompactado. O rastreio compactado tem menos efeito no desempenho do IBM MQ classes for Java que o rastreio descompactado.

Se *maxTraceBytes* e *traceCycles* estiverem configurados, vários arquivos de rastreio compactados serão criados no lugar de vários arquivos simples.

Se o IBM MQ classes for Java for finalizado de maneira não controlada, um arquivo de rastreio compactado pode não ser válido. Por essa razão, a compactação de rastreio deve ser usada somente quando o IBM MQ classes for Java for fechado de maneira controlada. Use apenas a compactação de rastreio se os problemas que estão sendo investigados não fizerem com que a própria JVM pare inesperadamente. Não use a compactação de rastreio ao diagnosticar problemas que possam resultar em encerramentos `System.Halt()` ou términos de JVM não controlados e anormais.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel especifica um nível de filtragem para o rastreio. Os níveis de rastreio definidos são os seguintes:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Cada nível de rastreio inclui todos os níveis inferiores. Por exemplo, se o nível de rastreio for configurado em `TRACE_INFO`, qualquer ponto de rastreio com um nível definido de

TRACE_EXCEPTION, TRACE_WARNINGou TRACE_INFO será gravado no rastreio. Todos os outros pontos de rastreio são excluídos.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace controla se o serviço de rastreio do cliente IBM MQ classes for Java é usado em um ambiente do WebSphere Application Server.

Se *standaloneTrace* for configurado como TRUE, as propriedades de rastreio do cliente IBM MQ classes for Java são usadas para determinar a configuração de rastreio.

Se *standaloneTrace* for configurado como FALSE e o cliente IBM MQ classes for Java estiver em execução em um contêiner do WebSphere Application Server, o serviço de rastreio do WebSphere Application Server será usado. As informações de rastreio que são geradas dependem das configurações de rastreio do servidor de aplicativos.

O valor padrão de *standaloneTrace* é FALSE.

IBM MQ classes for Java e ferramentas de gerenciamento de software
Ferramentas de gerenciamento de software como Apache Maven podem ser usadas com o IBM MQ classes for Java.

Muitas das grandes organizações de desenvolvimento usam essas ferramentas para gerenciar centralmente os repositórios de bibliotecas de terceiros.

Os IBM MQ classes for Java são compostos por um número de arquivos JAR. Quando você estiver desenvolvendo aplicativos de linguagem do Java usando essa API, uma instalação de um IBM MQ Server, IBM MQ Client ou IBM MQ Client SupportPac será requerida na máquina em que o aplicativo está sendo desenvolvido.

Se desejar usar uma ferramenta de gerenciamento de software e incluir os arquivos JAR que formam o IBM MQ classes for Java para um repositório gerenciado centralmente, os seguintes pontos deverão ser observados:

- Um repositório ou contêiner deve ser disponibilizado somente para os desenvolvedores em sua organização. Qualquer distribuição fora da organização não é permitida.
- O repositório precisa conter um conjunto completo e consistente de arquivos JAR a partir de uma liberação única ou fix pack do IBM MQ.
- Você é responsável por atualizar o repositório com qualquer manutenção fornecida pelo suporte IBM.

A partir de IBM MQ 8.0, o arquivo JAR `com.ibm.mq.allclient.jar` precisa ser instalado no repositório.

No IBM MQ 9.0, o provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS são necessários. Para obter mais informações, consulte “IBM MQ classes for Java arquivos JAR realocáveis” na página 359 e Suporte para JREs não IBM.

Configuração de pós-instalação para os aplicativos IBM MQ classes for Java

Após instalar o IBM MQ classes for Java, será possível configurar sua instalação para que seus próprios aplicativos sejam executados.

Lembre-se de verificar o arquivo leia-me do produto IBM MQ para obter as informações mais recentes ou para obter informações mais específicas sobre seu ambiente. A versão mais recente do arquivo leia-me do produto está disponível na página da web do IBM MQ, WebSphere MQe MQSeries leituras do produto.

Antes de tentar executar um aplicativo IBM MQ classes for Java no modo de ligações, certifique-se de ter configurado o IBM MQ conforme descrito em Configurando.

Configurando seu gerenciador de filas para aceitar conexões do cliente do IBM MQ classes for Java
Para configurar seu gerenciador de filas para aceitar pedidos de conexão que chegam de clientes, definir e permitir a utilização de um canal de conexão do servidor e iniciar um programa listener.

Consulte “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081 para obter detalhes.

Executando aplicativos IBM MQ classes for Java no Java security manager
O IBM MQ classes for Java pode executar com o Java security manager ativado. Para executar com sucesso aplicativos com o Java security manager ativado, deve-se configurar o Java Virtual Machine (JVM) com um arquivo de definição de política adequado.

A maneira mais simples de criar um arquivo de definição de política adequado é mudar o arquivo de políticas fornecido com o Java runtime environment (JRE). Na maioria dos sistemas, esse arquivo é armazenado no `path lib/security/java.policy` em relação ao diretório JRE. É possível editar os arquivos de políticas usando seu editor preferencial ou usando o programa policytool fornecido com seu JRE.

Você deve dar autoridade ao arquivo `com.ibm.mq.jmqi.jar` para que ele possa:

- Criar soquetes (no modo cliente)
- Carregara biblioteca nativa (no modo de ligações)
- Ler várias propriedades a partir do ambiente

A propriedade de sistema **`os.name`** deve estar disponível para o IBM MQ classes for Java ao executar no Java security manager.

Se o aplicativo Java usar o Java security manager, você deverá incluir a permissão a seguir no arquivo `java.security.policy` usado pelo aplicativo, caso contrário, serão lançadas exceções para o aplicativo:

```
permission java.lang.RuntimePermission "modifyThread";
```

Essa RuntimePermission é requerida pelo cliente como parte do gerenciamento de designação e encerramento de conversas multiplexadas sobre conexões TCP/IP com gerenciadores de filas.

Exemplo de entrada de arquivo de políticas

Aqui está um exemplo de uma entrada de arquivo de políticas que permite que o IBM MQ classes for Java seja executado com sucesso no gerenciador de segurança padrão. Substitua a sequência *MQ_INSTALLATION_PATH* neste exemplo com o local no qual o IBM MQ classes for Java está instalado em seu sistema.

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
```

```
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

Este exemplo de um arquivo de políticas permite que o IBM MQ classes for Java funcione corretamente no gerenciador de segurança, mas você ainda poderá precisar ativar seu próprio código para executar corretamente antes de seus aplicativos funcionarem.

O código de amostra fornecido com o IBM MQ classes for Java não foi ativado especificamente para uso com o gerenciador de segurança; entretanto, os testes IVT são executados com este arquivo de políticas e o gerenciador de segurança padrão no lugar.

Importante:

O utilitário de rastreio do IBM MQ classes for Java requer permissões adicionais, já que ele executa consulta adicional das propriedades do sistema e também operações adicionais do sistema de arquivos.

Um arquivo de política de segurança de template adequado para execução sob um gerenciador de segurança com rastreio ativado é fornecido no diretório `samples/wmqjava` da instalação IBM MQ como `example.security.policy`.

Para uma instalação padrão, o arquivo `example.security.policy` está localizado, em:

Windows
Em `C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`

Linux
Em `/opt/mqm/samp/wmqjava/samples/example.security.policy`

Solaris
Em `/opt/mqm/samp/wmqjava/samples/example.security.policy`

AIX

Em `/usr/mqm/samp/wmqjava/samples/example.security.policy`

Running IBM MQ classes for Java applications under CICS Transaction Server

An IBM MQ classes for Java application can be run as a transaction under CICS Transaction Server.

To run an IBM MQ classes for Java application as a transaction under CICS Transaction Server for z/OS, perform the following steps:

1. Define the application and transaction to CICS by using the supplied CEDA transaction.
2. Ensure that the IBM MQ CICS adapter is installed in your CICS system. z/OS (See Using IBM MQ with CICS for details.)
3. Ensure that the JVM environment specified in CICS includes the appropriate CLASSPATH and LIBPATH entries.
4. Initiate the transaction by using any of your normal processes.

For more information on running CICS Java transactions, refer to your CICS system documentation.

Verificando a Instalação do IBM MQ classes for Java

Um programa de verificação de instalação, MQIVP, é fornecido com o IBM MQ classes for Java. É possível usar esse programa para testar todos os modos de conexão do IBM MQ classes for Java.

O programa solicita diversas opções e outros dados para determinar qual modo de conexão você deseja verificar. Use o procedimento a seguir para verificar sua instalação:

1. Se for executar o programa no modo cliente, configure seu gerenciador de filas conforme descrito em “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081. A fila a ser usada é SYSTEM.DEFAULT.LOCAL.QUEUE
2. Se for executar o programa no modo cliente, consulte também “Usando o IBM MQ classes for Java” na página 353.

   Execute as etapas restantes deste procedimento no sistema no qual irá executar o programa.
3. Certifique-se de que você tenha atualizado sua variável de ambiente CLASSPATH de acordo com as instruções em “Variáveis de ambiente relevantes para o IBM MQ classes for Java” na página 362.
4. Mude o diretório para *MQ_INSTALLATION_PATH*`/mqm/samp/wmqjava/samples`, em que *MQ_INSTALLATION_PATH* é o caminho para a instalação do IBM MQ. Em seguida, no prompt de comandos, insira:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   em que *library_path* é o caminho para as bibliotecas do IBM MQ classes for Java (consulte “IBM MQ classes for Java bibliotecas” na página 364).

   No prompt marcado (1):

   - Para usar uma conexão TCP/IP, insira um nome de host do servidor IBM MQ.
   - Para usar conexão nativa (modo de ligações), deixe o campo em branco (não insira um nome).

   O programa tenta:

   - 1. Conectar-se ao gerenciador de filas
   - 2. Abra a fila SYSTEM.DEFAULT.LOCAL.QUEUE, coloque uma mensagem na fila, obtenha uma mensagem da fila e, em seguida, feche a fila
   - 3. Desconecte do gerenciador de filas
   - 4. Retorne uma mensagem se as operações forem bem-sucedidas

Aqui está um exemplo de prompts e respostas que você pode ver. Os prompts reais e suas respostas dependem da sua rede do IBM MQ.

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

Nota:

1. z/OS No z/OS, deixe o campo em branco no prompt marcado como (1).
2. Se você escolher conexão do servidor, não verá os prompts marcados como (2).
3. IBM i No IBM i, será possível emitir somente o comando `java MQIVP` a partir do QShell. Como alternativa, é possível executar o aplicativo usando o comando CL `RUNJVA CLASS(MQIVP)`.

Usando os aplicativos de amostra IBM MQ classes for Java

Os aplicativos de amostra do IBM MQ classes for Java fornecem uma visão geral dos recursos comuns da API do IBM MQ classes for Java. É possível usá-los para verificar a sua instalação e o servidor de sistema de mensagens configurado e para ajudar a construir os seus próprios aplicativos.

Sobre esta tarefa

Se você precisar de ajuda para criar seus próprios aplicativos, será possível usar os aplicativos de amostra como um ponto de início. Tanto a origem quanto uma versão compilada são fornecidas para cada aplicativo. Revise o código-fonte de amostra e identifique as principais etapas para criar cada objeto necessário para o seu aplicativo (MQQueueManager, MQConstants, MQMessage, MQPutMessageOptions e MQDestination) e configurar qualquer propriedade específica necessária para determinar como você deseja que o aplicativo funcione. Para obter informações adicionais, consulte “Gravando aplicativos do IBM MQ classes for Java” na página 378. As amostras podem estar sujeitas a mudanças em futuras liberações do IBM MQ Java.

A Tabela 55 na página 375 mostra o local de instalação dos aplicativos de amostra do IBM MQ classes for Java em cada plataforma:

Tabela 55. Diretórios de instalação para os aplicativos de amostra do IBM MQ classes for Java

Plataforma	Diretório
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\wmqjava\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/wmqjava/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/wmqjava`

A Tabela 56 na página 376 mostra os conjuntos de aplicativos de amostra fornecidos com IBM MQ classes for Java.

Tabela 56. Aplicativos de Amostra do IBM MQ classes for Java	
Nome da amostra	**Descrição**
IMSBridgeSample.java	Programa simples para demonstrar o uso da Ponte do IMS com o IBM MQ classes for Java.
MQIVP.java	Programa de verificação de instalação do IBM MQ Java.
MQMessagePropertiesSample.java	Demonstra o uso da API de Propriedades da Mensagem
MQPubSubApiSample.java	Demonstra o uso da API de publicação / assinatura
MQSample.java	Programa simples para demonstrar a colocação e o recebimento de uma mensagem de uma fila.
MQSampleMessageManager.java	A classe do utilitário para manipulação de mensagens nas amostras IBM MQ base Java
mqjcivp.properties	Este pacote de recursos contém as mensagens usadas pelo programa de verificação de instalação IBM MQ classes for Java (`MQIVP.java`).

O IBM MQ classes for Java fornece um script chamado `runjms` que pode ser usado para executar os aplicativos de amostra. Esse script configura o ambiente do IBM MQ para permitir que você execute os aplicativos de amostra do IBM MQ classes for Java.

A Tabela 57 na página 376 mostra o local do script em cada plataforma:

Tabela 57. Localização do script `runjms`	
Plataforma	**Diretório**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	ou
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`java/bin/runjms`

Para usar o script `runjms` para chamar um aplicativo de amostra, conclua as seguintes etapas:

Procedimento

1. Ative um prompt de comandos e navegue até o diretório que contém o aplicativo de amostra que você deseja executar.
2. Insira o seguinte comando:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   O aplicativo de amostra exibe uma lista dos parâmetros necessários.
3. Insira o comando a seguir para executar a amostra com estes parâmetros:

```
Path to the runjms script/runjms sample_application_name parameters
```

Exemplo

Linux Por exemplo, para executar a amostra MQIVP no Linux, insira os comandos a seguir:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

Conceitos relacionados

"O que é instalado para IBM MQ classes for JMS" na página 90
Vários arquivos e diretórios são criados ao instalar o IBM MQ classes for JMS. No Windows, algumas configurações são executadas durante a instalação, configurando automaticamente variáveis de ambiente. Em outras plataformas e em determinados ambientes de Windows, deve-se configurar as variáveis de ambiente antes de poder executar aplicativos IBM MQ classes for JMS.

Resolvendo problemas do IBM MQ classes for Java

Inicialmente, execute o programa de verificação da instalação. Também pode ser necessário usar o recurso de rastreio.

Se um aplicativo não for concluído com sucesso, execute o programa de verificação de instalação e siga as recomendações fornecidas nas mensagens de diagnóstico. O programa de verificação de instalação é descrito em "Verificando a Instalação do IBM MQ classes for Java" na página 374.

Se os problemas continuarem e for necessário contatar a equipe de serviço IBM, poderá ser solicitado que você ative o recurso de rastreio. Faça isso conforme mostrado no exemplo a seguir.

Para rastrear o programa `MQIVP`:

- Crie um arquivo de propriedades `com.ibm.mq.commonservices` (consulte Usando o com.ibm.mq.commonservices.
- Insira o seguinte comando:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

em que:

- *commonservices_properties_file* é o caminho (incluindo o nome do arquivo) para o arquivo de propriedades `com.ibm.mq.commonservices` .
- *library_path* é o caminho para as bibliotecas do IBM MQ classes for Java (consulte "IBM MQ classes for Java bibliotecas" na página 364).

Para obter mais informações sobre como usar o rastreio, consulte Rastreando aplicativos IBM MQ classes for Java.

z/OS MQ Adv. VUE Java client connectivity to batch applications running on z/OS

Under certain conditions, an IBM MQ classes for Java application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

By using a client connection, an IBM MQ classes for Java application can connect to a remote z/OS queue manager if the following conditions apply:

- The application is running in a batch environment.
- O gerenciador de filas que está sendo conectado está em execução com a autorização IBM MQ Advanced for z/OS Value Unit Edition e, portanto, possui o parâmetro **ADVCAP** configurado como ENABLED.

Para obter mais informações sobre o IBM MQ Advanced for z/OS Value Unit Edition , consulte IBM MQ identificadores do produto e informações de exportação

Consulte DISPLAY QMGR para obter mais informações sobre **ADVCAP** e START QMGR para obter mais informações sobre **QMGRPROD**.

An IBM MQ classes for Java application on z/OS cannot use a client mode connection to connect to a queue manager that is not running on z/OS

If an IBM MQ classes for Java application on z/OS attempts to connect using client mode, and is not allowed to do so, MQRC_ENVIRONMENT_ERROR is returned.

Advanced Message Security (AMS) support

IBM MQ classes for Java client applications can use AMS when connecting to remote z/OS queue managers, subject to the conditions previously described in this topic.

Para usar AMS dessa maneira, os aplicativos clientes devem usar um tipo de armazenamento de chave `jceracfks` em `keystore.conf`, em que:

- O prefixo do nome da propriedade é `jceracfks` e esse prefixo de nome não faz distinção entre maiúsculas e minúsculas.
- O armazenamento de chaves é um conjunto de chaves RACF.
- As senhas não são necessárias e serão ignoradas. Isso ocorre porque conjuntos de chaves RACF não usam senhas.
- Se você especificar o provedor, ele deverá ser `IBMJCE`.

Quando você usa `jceracfks` com o AMS, o armazenamento de chaves deve estar no formato: `safkeyring://`*`user`*`/`*`keyring`*, em que:

- `safkeyring` é um literal e esse nome não faz distinção entre maiúsculas e minúsculas
- *`user`* é o ID do usuário do RACF que possui o conjunto de chaves
- *`keyring`* é o nome do conjunto de chaves RACF e o nome do conjunto de chaves faz distinção entre maiúsculas e minúsculas

O exemplo a seguir usa o conjunto de chaves padrão AMS para o usuário `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Related concepts

“JMS/Jakarta Messaging client connectivity to batch applications running on z/OS” on page 127
Under certain conditions, an IBM MQ classes for JMS/Jakarta Messaging application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

Gravando aplicativos do IBM MQ classes for Java

Esta coleta de tópicos fornece informações para ajudar na gravação de aplicativos do Java para interagir com os sistemas IBM MQ.

Para usar o IBM MQ classes for Java para acessar filas do IBM MQ, grave aplicativos Java que contêm chamadas que colocam e recebem mensagens de filas do IBM MQ. Para obter detalhes de classes individuais, consulte IBM MQ classes for Java.

Nota: A reconexão de cliente automática não é suportada pelo IBM MQ classes for Java.

A interface do IBM MQ classes for Java

A interface de IBM MQ programação do aplicativo processual do usa verbos que agem sobre objetos. A interface de programação do Java usa objetos os quais o cliente aciona chamando métodos.

A interface de programação do aplicativo processual do IBM MQ é construída a cerca de verbos como estes:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Todos esses verbos aceitam, como um parâmetro, um manipulador para o objeto do IBM MQ no qual devem operar. Seu programa consiste em um conjunto de objetos do IBM MQ, que é acionado ao chamar métodos nestes objetos.

Ao usar a interface processual, desconecte-se de um gerenciador de filas usando a chamada MQDISC(Hconn, CompCode, Reason), em que *Hconn* é um identificador para o gerenciador de filas.

Na interface do Java, o gerenciador de filas é representado por um objeto de classe MQQueueManager. Desconecte-se do gerenciador de filas chamando o método disconnect() nessa classe.

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

Modos de conexão do IBM MQ classes for Java

A maneira como você programa o IBM MQ classes for Java tem algumas dependências nos modos de conexão que você deseja usar.

Se você usar conexões do cliente, há uma série de diferenças do IBM MQ MQI client, mas é conceitualmente semelhante. Se usar o modo de ligações, será possível usar ligações de atalho e poderá emitir o comando MQBEGIN. Você especifica qual modo usar configurando variáveis na classe MQEnvironment.

Conexões do cliente do IBM MQ classes for Java

Quando o IBM MQ classes for Java é usado como um cliente, é como o IBM MQ MQI client, mas possui várias diferenças.

Se você estiver programando para *IBM MQ classes for Java* para ser usado como um cliente, esteja ciente das seguintes diferenças:

- Ele suporta apenas TCP/IP.
- Não lê nenhuma variável de ambiente do IBM MQ na inicialização.
- Informações que seriam armazenadas em uma definição de canal e nas variáveis de ambiente podem ser armazenadas em uma classe chamada Environment. Como alternativa, estas informações podem ser transmitidos como parâmetros quando a conexão for feita.
- Condições de erro e exceção são gravadas em um log especificado na classe `MQException`. O destino de erro padrão é o console do Java.
- Somente os atributos a seguir em um arquivo de configuração do cliente IBM MQ são relevantes para o IBM MQ classes for Java. Se você especificar outros atributos, são ineficazes.

Sub-rotina	**Atributo**
Sub-rotina ClientExitPath do arquivo de configuração do cliente	Caminho Padrão das Saídas
Sub-rotina ClientExitPath do arquivo de configuração do cliente	ExitsDefaultPath64
Sub-rotina ClientExitPath do arquivo de configuração do cliente	JavaExitsClasspath

Sub-rotina	Atributo
Sub-rotina MessageBuffer do arquivo de configuração do cliente	MaximumSize
Sub-rotina MessageBuffer do arquivo de configuração do cliente	PurgeTime
Sub-rotina MessageBuffer do arquivo de configuração do cliente	UpdatePercentage
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntRcvBuffSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntSndBuffSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	Connect_Timeout
Sub-rotina TCP do Arquivo de Configuração do Cliente	KeepAlive

- Se estiver conectando a um gerenciador de filas que requer dados de caractere a serem convertidos, então o V7 do Java do cliente agora é capaz de fazer a conversão se o gerenciador de filas não puder fazer isso. A JVM do cliente deve suportar a conversão entre o CCSID do cliente e que do gerenciador de filas.
- A reconexão do cliente automática não é suportada pelo IBM MQ classes for Java.

Quando usado no modo cliente, o *IBM MQ classes for Java* não suporta a chamada MQBEGIN.

Modo de ligações do IBM MQ classes for Java
O modo de ligações do IBM MQ classes for Java difere do modo cliente em três maneiras principais.

Quando usado no modo de ligações, o IBM MQ classes for Java usa a Java Native Interface (JNI) para chamar diretamente para a API do gerenciador de filas existente, em vez de se realizar a comunicação por uma rede.

Por padrão, os aplicativos que usam o IBM MQ classes for Java no modo de ligações se conectam a um gerenciador de filas usando a *ConnectOption*, MQCNO_STANDARD_BINDINGS.

O IBM MQ classes for Java suporta as seguintes *ConnectOptions*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

Para obter informações adicionais sobre *ConnectOptions*, consulte “Conectando-se a um gerenciador de filas usando a chamada MQCONNX” na página 749.

O modo ligações suporta a chamada MQBEGIN para iniciar unidades globais de trabalho que são coordenadas pelo gerenciador de filas em todas as plataformas do IBM MQ for IBM i e do IBM MQ for z/OS.

A maioria dos parâmetros fornecidos pela classe `MQEnvironment` não são relevantes para o modo de ligações e são ignoradas.

Definindo qual conexão IBM MQ classes for Java usar
O tipo de conexão a ser usado é determinado pela configuração de variáveis na classe MQEnvironment.

Duas variáveis são usadas:

MQEnvironment.properties
O tipo de conexão é determinado pelo valor associado com o nome da chave CMQC.TRANSPORT_PROPERTY. Os valores possíveis são os seguintes:

CMQC.TRANSPORT_MQSERIES_BINDINGS
Conectar no modo de ligações

CMQC.TRANSPORT_MQSERIES_CLIENT
Conectar no modo cliente

CMQC.TRANSPORT_MQSERIES
O modo de conexão é determinado pelo valor da propriedade *hostname*

MQEnvironment.hostname
Configure o valor dessa variável como segue:

- Para conexões do cliente, configure o valor dessa variável para o nome do host do servidor IBM MQ ao qual deseja conectar-se
- Para o modo de ligações não configure essa variável ou configure-a como nulo

Operações nos Gerenciadores de Filas

Essa coleção de tópicos descreve como conectar-se a, e desconectar-se de, um gerenciador de filas usando o IBM MQ classes for Java.

Configurando o ambiente do IBM MQ para o IBM MQ classes for Java
Para que um aplicativo estabeleça conexão com um gerenciador de filas no modo cliente, o aplicativo deve especificar o nome do canal, o nome do host e o número da porta.

Nota: As informações neste tópico são relevantes apenas se o seu aplicativo se conectar a um gerenciador de filas no modo cliente. Elas não serão relevantes se a conexão se der no modo de ligações. Consulte: "Modos de conexão para IBM MQ classes for JMS" na página 111

É possível especificar o nome do canal, o nome do host e o número da porta de uma das duas maneiras a seguir: como campos na classe MQEnvironment ou como propriedades do objeto MQQueueManager.

Se você configurar campos na classe MQEnvironment, eles se aplicarão ao seu aplicativo inteiro, exceto onde são substituídos por uma tabela hash de propriedades. Para especificar o nome do canal e o nome do host em MQEnvironment, use o código a seguir:

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

Isto é equivalente à configuração de uma variável de ambiente **MQSERVER**:

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

Por padrão, os clientes do Java tentam se conectar a um listener do IBM MQ na porta 1414. Para especificar uma porta diferente, use o código a seguir:

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

em que nnnn é o número da porta requerido

Se você passar propriedades para um objeto do gerenciador de filas em sua criação, elas se aplicarão apenas a esse gerenciador de filas. Crie entradas em um objeto Hashtable com chaves de **hostname**, **channel** e, opcionalmente, **port**, e com valores apropriados. Para usar a porta padrão, 1414, é possível omitir a entrada **port**. Crie o objeto MQQueueManager usando um construtor que aceita a tabela hash de propriedades.

Identificando uma conexão com o gerenciador de filas configurando um nome de aplicativo

Um aplicativo pode configurar um nome que identifica sua conexão com o gerenciador de filas. Esse nome do aplicativo é mostrado no comando **DISPLAY CONN MQSC/PCF** (em que o campo é chamado **APPLTAG**) ou na exibição IBM MQ Explorer **Conexões de Aplicativos** (em que o campo é chamado **App name**).

Os nomes de aplicativos são limitados a 28 caracteres, portanto os nomes mais longos são truncados. Se um nome de aplicativo não for especificado, um padrão será fornecido. O nome padrão tem como base a classe de chamada (principal), no entanto, se essa informação não estiver disponível, o texto `IBM MQ Client for Java` será usado.

Se o nome da classe de chamada for usado, ele será ajustado removendo os nomes de pacotes iniciais, se necessário. Por exemplo, se a classe de chamada for `com.example.MainApp`, o nome completo será usado, mas se a classe de chamada for `com.example.dictionaryAndThesaurus.multilingual.mainApp`, o nome `multilingual.mainApp` será usado, pois ele é a combinação mais longa do nome da classe e o nome do pacote mais à direita que cabe no comprimento disponível.

Se o próprio nome da classe tiver mais de 28 caracteres de comprimento, ele será truncado para ajuste. Por exemplo, `com.example.mainApplicationForSecondTestCase` se torna `mainApplicationForSecondTest`.

Para configurar um nome de aplicativo na classe MQEnvironment, inclua o nome na tabela hash MQEnvironment.properties, com uma chave de **MQConstants.APPNAME_PROPERTY**, usando o código a seguir:

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

Para configurar um nome de aplicativo na tabela hash de propriedades que é passada para o construtor MQQueueManager, inclua o nome na tabela hash de propriedades com uma chave de **MQConstants.APPNAME_PROPERTY**.

Substituindo propriedades especificadas em um arquivo de configuração do cliente IBM MQ

Um arquivo de configuração do cliente IBM MQ também pode especificar propriedades que são usadas para configurar o IBM MQ classes for Java. No entanto, as propriedades especificadas em um arquivo de configuração do IBM MQ MQI client se aplicam apenas quando um aplicativo se conecta a um gerenciador de filas no modo cliente.

Se necessário, será possível substituir qualquer atributo em um arquivo de configuração do IBM MQ de qualquer uma das seguintes maneiras. As opções são mostradas em ordem de precedência.

- Configure uma propriedade do sistema Java para a propriedade de configuração.
- Configure a propriedade no mapa MQEnvironment.properties.
- No Java5 e liberações posteriores, defina uma variável de ambiente do sistema.

Somente os atributos a seguir em um arquivo de configuração do cliente IBM MQ são relevantes para o IBM MQ classes for Java. Se você especificar ou substituir outros atributos, isso não terá efeito.

Sub-rotina	Atributo
Sub-rotina ClientExitPath do arquivo de configuração do cliente	Caminho Padrão das Saídas
Sub-rotina ClientExitPath do arquivo de configuração do cliente	ExitsDefaultPath64

Sub-rotina	Atributo
Sub-rotina ClientExitPath do arquivo de configuração do cliente	JavaExitsClasspath
Sub-rotina MessageBuffer do arquivo de configuração do cliente	MaximumSize
Sub-rotina MessageBuffer do arquivo de configuração do cliente	PurgeTime
Sub-rotina MessageBuffer do arquivo de configuração do cliente	UpdatePercentage
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntRcvBufSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	ClntSndBufSize
Sub-rotina TCP do Arquivo de Configuração do Cliente	Connect_Timeout
Sub-rotina TCP do Arquivo de Configuração do Cliente	KeepAlive

Conectando-se a um gerenciador de filas no IBM MQ classes for Java
Conecte-se a um gerenciador de filas criando uma nova instância da classe MQQueueManager. Desconecte-se de um gerenciador de filas chamando o método disconnect().

Agora você está pronto para conectar-se a um gerenciador de filas criando uma nova instância da classe MQQueueManager:

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

Para desconectar-se de um gerenciador de filas, chame o método disconnect() no gerenciador de filas:

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

Se você chamar o método disconnect, todas as filas e processos abertos que você acessou usando esse gerenciador de filas serão encerrados. Entretanto, é uma boa prática de programação fechar estes recursos explicitamente quando você terminar de usá-los. Para fazer isso, use o método close() nos objetos relevantes.

O commit() e métodos de recuperação() em um gerenciador de filas são equivalentes às chamadas MQCMIT e MQBACK que são usadas com a interface processual.

Usando uma tabela de definição de canal do cliente com o IBM MQ classes for Java
Um aplicativo cliente IBM MQ classes for Java pode usar as definições de canal de conexão do cliente armazenadas em uma tabela de definição de canal do cliente (CCDT).

Como uma alternativa para criar uma definição de canal de conexão do cliente configurando determinados campos e propriedades do ambiente na classe `MQEnvironment` ou passando-os para um `MQQueueManager` em uma tabela hash de propriedades, um aplicativo cliente IBM MQ classes for Java pode usar definições de canal de conexão do cliente armazenadas em uma tabela de definição de canal do cliente. Essas definições são criadas pelos comandos do IBM MQ Script (MQSC) ou os comandos do IBM MQ Programmable Command Format (PCF) ou usando o IBM MQ Explorer.

Quando o aplicativo criar um objeto `MQQueueManager`, o cliente IBM MQ classes for Java irá procurar a tabela de definição de canal do cliente para uma definição adequada de canal de conexão do cliente e usará a definição de canal para iniciar um canal MQI. Para obter mais informações sobre tabelas de definição de canal do cliente e como construir um, consulte Client Channel Definition Table.

Para usar uma tabela de definição de canal do cliente, um aplicativo deve primeiramente criar um objeto de URL. O objeto de URL contém um localizador uniforme de recursos (URL) que identifica o nome e o local do arquivo contendo a tabela de definição de canal do cliente e especifica como o arquivo pode ser acessado.

Por exemplo, se o arquivo `ccdt1.tab` contiver uma tabela de definição de canal do cliente e estiver armazenado no mesmo sistema no qual o aplicativo está em execução, o aplicativo poderá criar um objeto URL da seguinte maneira:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

Como outro exemplo, suponha que o arquivo `ccdt2.tab` contenha uma tabela de definição de canal do cliente e esteja armazenado em um sistema diferente daquele em que o aplicativo está em execução. Se o arquivo puder ser acessado usando o protocolo FTP, o aplicativo poderá criar um objeto de URL da seguinte maneira:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Depois que o aplicativo criou um objeto de URL, o aplicativo poderá criar um objeto `MQQueueManager` usando um dos construtores que obtém um objeto de URL como um parâmetro. Aqui está um exemplo:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Essa instrução faz com que o cliente IBM MQ classes for Java acesse a tabela de definição de canal do cliente identificada pelo objeto de URL chanTab2, procurar a tabela para uma definição adequado de canal de conexão do cliente e, em seguida, usar a definição de canal para iniciar um canal MQI no gerenciador de filas chamado MARS.

Observe os seguintes pontos que se aplicarão, se um aplicativo usar uma tabela de definição de canal do cliente:

- Quando o aplicativo criar um objeto `MQQueueManager` usando um construtor que usa um objeto de URL como um parâmetro, nenhum nome de canal deverá ser configurado na classe `MQEnvironment`, como um campo ou como uma propriedade de ambiente. Se um nome de canal for configurado, o cliente IBM MQ classes for Java lançará uma `MQException`. A propriedade do campo ou ambiente especificando o nome do canal será considerada para ser configurada, se seu valor for algo diferente de nulo, uma sequência vazia ou uma sequência que contém todos os caracteres em branco.
- O parâmetro **queueManagerName** no construtor `MQQueueManager` pode ter um dos seguintes valores:
  - O nome de um gerenciador de filas
  - Um asterisco (*) seguido pelo nome de um grupo de gerenciadores de filas
  - Um asterisco (*)
  - Nulo, uma sequência vazia ou uma sequência que contém todos os caracteres em branco

  Estes são os mesmos valores que podem ser usados para o parâmetro **QMgrName** em uma chamada MQCONN emitida por um aplicativo cliente que está usando o Message Queue Interface (MQI). Para obter mais informações sobre o significado desses valores, consulte “Visão geral da Message Queue Interface” na página 733.

  Se seu aplicativo usar uma definição do conjunto de conexões, consulte“Controlando o conjunto de conexões padrão em IBM MQ classes for Java” na página 404.
- Quando o cliente IBM MQ classes for Java localizar uma definição adequada de canal de conexão do cliente na tabela de definição de canal do cliente, ele usará somente as informações extraídas a partir dessa definição de canal para iniciar um canal MQI. Quaisquer campos ou propriedades de ambiente relacionados ao canal que o aplicativo possa ser configurado na classe `MQEnvironment` são ignorados.

  Em particular, observe os pontos a seguir se você estiver usando Segurança da Camada de Transporte (TLS):

– Um canal MQI usará o TLS somente se a definição de canal extraída da tabela de definição de canal de cliente especificar o nome de um CipherSpec suportado pelo cliente IBM MQ classes for Java.
– Uma tabela de definição de canal do cliente também contém informações sobre o local dos servidores Lightweight Directory Access Protocol (LDAP) que mantém as listas de revogação de certificado (CRLs). O cliente IBM MQ classes for Java usa apenas essas informações para acessar os servidores LDAP que contêm CRLs.
– Uma tabela de definição de canal do cliente também pode conter o local de um respondente do OCSP. IBM MQ classes for Java não pode usar as informações do OCSP em um arquivo da tabela de definição de canal do cliente. No entanto, é possível configurar o OCSP, conforme descrito na seção Usando o Online Certificate Protocol

Para obter mais informações sobre como usar o TLS com uma tabela de definição de canal de cliente, veja Especificando que um canal MQI usa TLS.

Observe também os pontos a seguir se estiver usando saídas de canal:

– Um canal MQI usa as saídas do canal e dados do usuário associados especificado pela definição de canal extraída da tabela de definição de canal do cliente de preferência para saídas de canal e dados especificados usando outros métodos.
– Uma definição de canal extraída de uma tabela de definição de canal do cliente pode especificar saídas de canal gravadas em Java, C ou C ++. Para obter mais informações sobre como gravar uma saída de canal em Java, consulte “Criando uma saída do canal no IBM MQ classes for Java” na página 398. Para obter mais informações sobre como gravar uma saída de canal em outros idiomas, consulte “Usando saídas do canal não escritas em Java com o IBM MQ classes for Java” na página 401.

A especificação de um intervalo de portas para conexões do cliente IBM MQ classes for Java
É possível especificar uma porta ou um intervalo de portas, que um aplicativo pode ser ligado em uma de duas maneiras.

Quando um aplicativo IBM MQ classes for Java tentar se conectar a um gerenciador de filas do IBM MQ no modo cliente, o firewall poderá permitir apenas aquelas conexões originadas de portas especificadas ou intervalo de portas. Nesta situação, é possível especificar uma porta ou um intervalo de portas ao qual o aplicativo pode ser ligado. É possível especificar a(s) porta(s) das seguintes maneiras:

- É possível configurar o campo localAddressSetting na classe MQEnvironment. Aqui está um exemplo:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- É possível configurar o CMQC.LOCAL_ADDRESS_PROPERTY da propriedade do ambiente. Aqui está um exemplo:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Quando for possível construir o objeto MQQueueManager, será possível transmitir uma hashtable de propriedades que contém um LOCAL_ADDRESS_PROPERTY com o valor "192.0.2.0(2000,3000)"

Em cada um desses exemplos, quando o aplicativo posteriormente se conectar a um gerenciador de filas, o aplicativo será ligado a um endereço IP local e ao número da porta no intervalo de 192.0.2.0(2000) para 192.0.2.0(3000).

Em um sistema com mais de uma interface de rede, também é possível usar o campo localAddressSetting ou o CMQC.LOCAL_ADDRESS_PROPERTY da propriedade do ambiente para especificar qual interface de rede deve ser usada para uma conexão.

Os erros de conexão poderão ocorrer se você restringir o intervalo de portas. Se ocorrer um erro, uma MQException será lançada contendo o código de razão do IBM MQ MQRC_Q_MGR_NOT_AVAILABLE e a seguinte mensagem:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Um erro pode ocorrer se todas as portas no intervalo especificado estiverem em uso ou se o endereço IP especificado, um nome do host ou número de porta não for válido (um número de porta negativo, por exemplo).

Acessar filas, tópicos e processos no IBM MQ classes for Java

Para acessar filas, tópicos e processos, use métodos da classe MQQueueManager. O MQOD (estrutura do descritor de objeto) é reduzido nos parâmetros destes métodos.

Filas

Para abrir uma fila, é possível usar o método accessQueue da classe MQQueueManager. Por exemplo, em um gerenciador de filas chamado queueManager, use o seguinte código:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

O método accessQueue retorna um novo objeto da classe MQQueue.

Quando tiver concluído o uso da fila, use o método close() para fechá-la, como no exemplo a seguir:

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

Também é possível criar uma fila usando o construtor MQQueue. Os parâmetros são exatamente os mesmos que para o método accessQueue, com a adição de um parâmetro do gerenciador de filas. Por exemplo:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

É possível especificar várias opções ao criar filas. Para obter detalhes sobre isso, consulte Class.com.ibm.mq.MQQueue. Construir um objeto de fila desta maneira permite que você grave suas próprias subclasses de MQQueue.

tópicos

De forma semelhante, é possível abrir um tópico usando o método da classe accessTopic da classe MQQueueManager. Por exemplo, em um gerenciador de filas chamado queueManager, use o seguinte código para criar um assinante e publicador:

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

Quando tiver concluído o uso do tópico, use o método close() para fechá-lo.

Também é possível criar um tópico usando o construtor MQTopic. Os parâmetros são exatamente os mesmos que para o método accessTopic, com a adição de um parâmetro do gerenciador de filas. Por exemplo:

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

É possível especificar várias opções ao criar tópicos. Para obter detalhes destes, consulte Classe com.ibm.mq.MQTopic. Construir um objeto de tópico desta maneira permite que você grave suas próprias subclasses de MQTopic.

Um tópico deve ser aberto para publicação ou para assinatura. A classe MQQueueManager tem oito métodos accessTopic e a classe Tópico possui oito construtores. Em cada caso, quatro desses têm um parâmetro **destination** e quatro têm um parâmetro **subscriptionName** (incluindo dois que possuem ambos). Eles podem ser usados somente para abrir o tópico para assinaturas. Os dois métodos restantes possuem um parâmetro **openAs**, e o tópico pode ser aberto para qualquer publicação ou assinatura dependendo do valor do parâmetro **openAs**.

Para criar um tópico como assinante durável, use um método accessTopic da classe MQQueueManager ou um construtor MQTopic que aceite um nome de assinatura e, em qualquer caso, defina a opção CMQC.MQSO_DURABLE.

Processos

Para acessar um processo, use o método accessProcess do MQQueueManager. Por exemplo, em um gerenciador de filas chamado queueManager, use o seguinte código para criar um objeto MQProcess:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Para acessar um processo, use o método accessProcess do MQQueueManager.

O método accessProcess retorna um novo objeto da classe MQProcess.

Quando tiver concluído o uso do objeto do processo, use o método close() para fechá-lo, como no exemplo a seguir:

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

Também é possível criar um processo usando o construtor MQProcess. Os parâmetros são exatamente os mesmos que para o método accessProcess, com a adição de um parâmetro do gerenciador de filas. Por exemplo:

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

Construir um objeto do processo desta maneira permite que você grave suas próprias subclasses de MQProcess.

Manipulando mensagens no IBM MQ classes for Java

As mensagens são representadas pela classe MQMessage. Você colocar e obter mensagens utilizando métodos da classe MQDestination, que possui subclasses de MQQueue e MQTopic.

Coloque mensagens nas filas ou tópicos usando o método put() da classe MQDestination. Você recebe mensagens das filas ou tópicos usando o método get() da classe MQDestination. Diferente da interface processual, em que MQPUT e MQGET colocam e obtêm matrizes de bytes, a linguagem de programação Java coloca e obtém instâncias da classe MQMessage. A classe MQMessage encapsula o buffer de

dados que contém os dados da mensagem reais, junto com todos os parâmetros MQMD (descritor de mensagens) e propriedades de mensagem que descrevem essa mensagem.

Para criar uma nova mensagem, crie uma nova instância da classe MQMessage, e utilizar os métodos writeXXX para colocar dados no buffer de mensagem.

Quando a nova instância de mensagem for criada, todos os parâmetros MQMD serão automaticamente configurados com seus valores padrão, conforme definido em valores iniciais do e declarações de idioma para MQMD. O método put() de MQDestination também usa uma instância da classe MQPutMessageOptions como parâmetro. Esta classe representa a estrutura MQPMO. O exemplo a seguir cria uma mensagem e a coloca em uma fila:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage,pmo);
```

O método get() de MQDestination retorna uma nova instância de MQMessage, que representa a mensagem recém-obtida da fila. Ele também utiliza uma instância da classe MQGetMessageOptions como um parâmetro. Esta classe representa a estrutura de MQGMO.

Você não precisa especificar um tamanho de mensagem máximo, porque o método get() ajusta automaticamente o tamanho de seu buffer interno para encaixar a mensagem recebida. Use os métodos readXXX da classe MQMessage para acessar os dados na mensagem retornada.

O exemplo a seguir mostra como obter uma mensagem de uma fila:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

É possível alterar o formato numérico que os métodos de leitura e gravação usam configurando a variável de membro *encoding*.

É possível alterar o conjunto de caracteres para usar para ler e gravar sequências configurando a variável de membro *characterSet*.

Consulte a classe MQMessage para obter mais informações.

Nota: O método writeUTF() de MQMessage codifica automaticamente o comprimento da sequência, bem como os bytes Unicode que ela contém. Quando sua mensagem for lida por outro programa Java (usando readUTF()), essa é a maneira mais simples de enviar informações de sequência.

Melhorando o desempenho de mensagens não persistentes no IBM MQ classes for Java
Para melhorar o desempenho ao procurar as mensagens ou ao consumir as mensagens não persistentes de um aplicativo cliente, será possível usar a *leitura antecipada*. Os aplicativos clientes que usam MQGET ou consumo assíncrono se beneficiarão das melhorias de desempenho ao procurar mensagens ou ao consumir mensagens não persistentes.

Para obter informações gerais sobre o recurso de leitura antecipada, consulte o tópico relacionado.

No IBM MQ classes for Java, use as propriedades CMQC.MQSO_READ_AHEAD e CMQC.MQSO_NO_READ_AHEAD de um objeto MQQueue ou MQTopic para determinar se os consumidores de mensagens e os navegadores de filas terão permissão para usar a leitura antecipada nesse objeto.

Colocando mensagens de forma assíncrona usando o IBM MQ classes for Java
Para colocar uma mensagem forma assíncrona, configure MQPMO_ASYNC_RESPONSE.

Coloque as mensagens nas filas ou tópicos usando o método put() da classe MQDestination. Para colocar uma mensagem de forma assíncrona, ou seja, permitindo que a operação seja concluída sem aguardar por uma resposta do gerenciador de filas, é possível configurar MQPMO_ASYNC_RESPONSE no campo de opções de MQPutMessageOptions. Para determinar o sucesso ou falha de colocações assíncronas, use a chamada MQQueueManager.getAsyncStatus.

Publicar/assinar em IBM MQ classes for Java
No IBM MQ classes for Java, o tópico é representada pela classe MQTopic e publique para ele usando os métodos MQTopic.put().

Para informações gerais sobre publicar/assinar de IBM MQ, consulte Sistema de mensagens de publicar/assinar.

Manipulando cabeçalhos de mensagem do IBM MQ com o IBM MQ classes for Java
Classes Java são fornecidas representando diferentes tipos de cabeçalho de mensagem. Duas classes auxiliares também são fornecidas.

A interface MQHeader

Objetos de cabeçalho são descritos pela interface MQHeader, que fornece métodos de propósitos gerais para acessar campos de cabeçalho e para ler e gravar conteúdo de mensagem. Cada tipo de cabeçalho tem sua própria classe que implementa a interface MQHeader e inclui métodos getter e setter para campos individuais. Por exemplo, o tipo de cabeçalho MQRFH2 é representado pela classe MQRFH2; o tipo de cabeçalho MQDLH pela classe MQDLH e assim por diante. As classes de cabeçalho executam qualquer conversão de dados necessária automaticamente e podem ler ou gravar dados em qualquer codificação numérica ou conjunto de caracteres (CCSID) especificado.

Importante: As classes de cabeçalhos MQRFH2 tratam a mensagem como um arquivo de acesso aleatório, o que significa que o cursor deve ser posicionado no início da mensagem. Antes de usar uma classe de cabeçalho da mensagem interna, como MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH ou MQXMIT, certifique-se de atualizar a posição do cursor da mensagem para o local correto antes de transmitir a mensagem para a classe..

Classes Auxiliares

Duas classes auxiliares, MQHeaderIterator e MQHeaderList, ajudam na leitura e decodificação (análise) do conteúdo de cabeçalho em mensagens:

- A classe MQHeaderIterator funciona como um java.util.Iterator. Enquanto houver mais cabeçalhos na mensagem, o método next() retorna true e o método nextHeader() ou next() retorna o próximo objeto de cabeçalho.
- A MQHeaderList funciona como um java.util.List. Como a MQHeaderIterator, ela analisa o conteúdo do cabeçalho, mas também permite procurar cabeçalhos específicos, incluir cabeçalhos novos, remover cabeçalhos existentes, atualizar campos de cabeçalho e, em seguida, gravar o conteúdo do cabeçalho de volta em uma mensagem. Como alternativa, é possível criar uma MQHeaderList vazia e, em seguida, preenchê-la com instâncias de cabeçalho e gravá-la em uma mensagem uma vez ou repetidamente.

As classes MQHeaderIterator e MQHeaderList usam as informações de MQHeaderRegistry para saber quais classes de cabeçalho do IBM MQ são associados a determinados tipos e formatos de mensagens.

O MQHeaderRegistry é configurado com conhecimento de todos os formatos e tipos de cabeçalho do IBM MQ e suas classes de implementação, e também é possível registrar seus próprios tipos de cabeçalho.

O suporte é fornecido para os seguintes cabeçalhos do IBM MQ comumente usados

- MQRFH - Regras e formatação de cabeçalho
- MQRFH2 - Como o MQRFH, usado para passar mensagens para e a partir de um broker de mensagem pertencente ao IBM Integration Bus. Também usado para conter as propriedades de mensagem
- MQCIH - Ponte do CICS
- MQDLH - Cabeçalho de mensagens não entregues
- MQIIH - Informações de cabeçalho do IMS
- MQRMH - cabeçalho de mensagem de referência
- MQSAPH - Cabeçalho SAP
- MQWIH - Cabeçalho de informações de trabalho
- MQXQH - Cabeçalho da fila de transmissão
- MQDH - Cabeçalho de distribuição
- MQEPH - Cabeçalho PCF contido

Também é possível definir classes que representam seus próprios cabeçalhos.

Para usar um MQHeaderIterator para obter um cabeçalho RFH2, configure MQGMO_PROPERTIES_FORCE_MQRFH2 em GetMessageOptions ou configure a propriedade da fila PROPCTL para FORCE.

Imprimindo todos os cabeçalhos em uma mensagem usando o IBM MQ classes for Java
Neste exemplo, uma instância de MQHeaderIterator analisa os cabeçalhos em uma MQMessage que foi recebida de uma fila. O objetos MQHeader retornados do método nextHeader() exibem sua estrutura e conteúdo quando seu método toString é chamado.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

Ignorando os cabeçalhos em uma mensagem usando o IBM MQ classes for Java
Neste exemplo, o método skipHeaders () de MQHeaderIterator posiciona o cursor de leitura da mensagem imediatamente após o último cabeçalho.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

Localizando o código de razão em uma mensagem não entregue usando o IBM MQ classes for Java
Neste exemplo, o método de leitura preenche o objeto MQDLH pela leitura da mensagem. Após a operação de leitura, o cursor de leitura da mensagem é posicionado imediatamente após o conteúdo do cabeçalho MQDLH.

As mensagens na fila de mensagens não entregues do gerenciador de filas são prefixadas com um cabeçalho de devoluções (MQDLH). Para decidir como manipular essas mensagens, por exemplo, para determinar se deseja tentar novamente ou descartá-las, um aplicativo de manipulação de devoluções deve ver o código de razão contido no MQDLH.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

Todas as classes de cabeçalho também fornecem um construtor de conveniência para inicializá-las diretamente a partir da mensagem em uma única etapa. Portanto, o código neste exemplo poderia ser simplificado da seguinte forma:

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

Lendo e removendo o cabeçalho de uma mensagem não entregue usando o IBM MQ classes for Java
Neste exemplo, MQDLH é usado para remover o cabeçalho de uma mensagem não entregue.

Um aplicativo de manipulação de mensagens não entregues irá normalmente reenviar mensagens que foram rejeitadas se o código de razão indicar um erro transitório. Antes de reenviar a mensagem, ele deve remover o cabeçalho MQDLH.

Este exemplo executa as etapas a seguir (consulte os comentários no código de exemplo):

1. O MQHeaderList lê a mensagem inteira e cada cabeçalho encontrado na mensagem se torna um item na lista.
2. Mensagens não entregues contêm um MQDLH como seu primeiro cabeçalho, portanto, ele pode ser localizado no primeiro item da lista do cabeçalho. O MQDLH já foi preenchido a partir da mensagem quando o MQHeaderList foi construído, portanto, não há necessidade de chamar seu método de leitura.
3. O código de razão é extraído usando o método getReason() fornecido pela classe MQDLH.
4. O código de razão foi inspecionado e indica que é apropriado para reenviar a mensagem. O MQDLH é removido usando o método MQHeaderList remove().
5. O MQHeaderList grava seu conteúdo restante em um novo objeto de mensagem. A nova mensagem agora contém tudo o que estiver na mensagem original, exceto o MQDLH e pode ser gravada em uma fila. O argumento **true** para o construtor e para o método de gravação indica que o corpo da mensagem deve ser mantido no MQHeaderList e gravado novamente.
6. O campo de formato no descritor de mensagens da nova mensagem agora contém o valor que estava anteriormente no campo de formato do MQDLH. Os dados da mensagem correspondem à codificação numérica e ao CCSID configurado no descritor de mensagens.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.
```

```
MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

Imprimindo o conteúdo de uma mensagem usando o IBM MQ classes for Java
Este exemplo usa MQHeaderList para imprimir o conteúdo de uma mensagem, incluindo seus cabeçalhos.

A saída contém uma visualização de todos os conteúdos do cabeçalho, bem como do corpo da mensagem. A classe MQHeaderList decodifica todos os cabeçalhos de uma vez, enquanto que MQHeaderIterator passa por eles um por vez sob controle do aplicativo. Você pode usar essa técnica para fornecer uma ferramenta de depuração simples ao gravar aplicativos WebSphere MQ.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

Este exemplo também imprime os campos do descritor de mensagens, usando a classe MQMD. O método copyFrom() da classe com.ibm.mq.headers.MQMD preenche o objeto de cabeçalho a partir dos campos do descritor de mensagens da MQMessage em vez de lendo o corpo da mensagem.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

Localizando um tipo específico de cabeçalho em uma mensagem usando o IBM MQ classes for Java
Este exemplo usa o método indexOf(String) de MQHeaderList para localizar um cabeçalho MQRFH2 em uma mensagem, se uma estiver presente.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
        ...
}
```

Analisando um cabeçalho MQRFH2 usando o IBM MQ classes for Java
Este exemplo mostra como acessar um valor de campo conhecido em uma pasta denominada, usando a classe MQRFH2.

A classe MQRFH2 fornece inúmeras maneiras para acessar não apenas os campos na parte fixa da estrutura, mas também o conteúdo da pasta codificada por XML que é transportado dentro do campo NameValueData. Esse exemplo mostra como acessar um valor de campo conhecido em uma pasta denominada – nessa instância, o campo Rto na pasta jms, que representa o nome da fila de resposta em uma mensagem MQ JMS.

```
MQRFH2 rfh = ...
```

```
String value = rfh.getStringFieldValue ("jms", "Rto");
```

Para descobrir o conteúdo de um MQRFH2 (em vez de solicitar campos específicos diretamente), é possível usar o método getFolders para retornar uma lista de MQRFH2.Element, que representa a estrutura de uma pasta que poderia conter campos e outras pastas. A definição de um campo ou pasta para nulo a remove do MQRFH2. Ao manipular o conteúdo da pasta NameValueData desta maneira, o campo StrucLength é automaticamente atualizado de acordo.

Lendo e gravando fluxos de bytes diferentes de objetos MQMessage usando IBM MQ classes for Java
Esses exemplos usam as classes de cabeçalho para analisar e manipular o conteúdo do cabeçalho do IBM MQ quando a origem de dados não for um objeto MQMessage.

É possível usar as classes de cabeçalho para analisar e manipular o conteúdo de cabeçalho do IBM MQ mesmo quando a origem de dados for algo diferente de um objeto MQMessage. A interface MQHeader implementada por toda classe de cabeçalho fornece os métodos `int read (java.io.DataInput message, int encoding, int characterSet)` e `int write (java.io.DataOutput message, int encoding, int characterSet)`. A classe com.ibm.mq.MQMessage implementa as interfaces java.io.DataInput e java.io.DataOutput. Isso significa que é possível usar os dois métodos MQHeader para ler e gravar conteúdo de MQMessage, substituindo a codificação e o CCSID especificados no descritor de mensagens. Isso é útil para mensagens que contêm uma cadeia de cabeçalhos em diferentes codificações.

Também é possível obter objetos DataInput e DataOutput de outros fluxos de dados, por exemplo, fluxos de arquivos ou soquetes ou matrizes de bytes transportadas em mensagens do JMS. As classes java.io.DataInputStream implementam DataInput e as classes java.io.DataOutputStream implementam DataOutput. Este exemplo lê conteúdo de cabeçalho do IBM MQ de uma matriz de bytes:

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

A linha iniciada por MQHeaderIterator poderia ser substituída por

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

Este exemplo grava em uma matriz de bytes usando um DataOutputStream:

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

Ao trabalhar com fluxos dessa maneira, tome cuidado para usar os valores corretos para a codificação e os argumentos de characterSet. Ao ler cabeçalhos, especifique a codificação e o CCSID com o qual o conteúdo de bytes foi gravado originalmente. Ao gravar cabeçalhos, especifique a codificação e o CCSID que deseja produzir. A conversão de dados é executada automaticamente pelas classes de cabeçalho.

Criando classes para novos tipos de cabeçalho usando o IBM MQ classes for Java
É possível criar classes Java para os tipos de cabeçalho não fornecidos com o IBM MQ classes for Java.

Para incluir uma classe Java que represente um novo tipo de cabeçalho que é possível usar da mesma maneira que qualquer classe de cabeçalho fornecido com o IBM MQ classes for Java, você cria uma classe que implementa a interface MQHeader. A abordagem mais simples é estender a classe com.ibm.mq.headers.impl.Header. Este exemplo produz uma classe totalmente funcional que representa a estrutura do cabeçalho MQTM. Não é necessário incluir métodos getter e setter individuais para cada

campo, mas é uma conveniência útil para usuários da classe de cabeçalho. Os métodos getValue e setValue genéricos que usam uma sequência para o nome do campo funcionarão para todos os campos definidos no tipo de cabeçalho. Os métodos read, write e size herdados permitirão que instâncias do novo tipo de cabeçalho sejam lidas e gravadas e calcularão o tamanho do cabeçalho corretamente com base em sua definição de campo. A definição de tipo é criada apenas uma vez, no entanto, muitas instâncias dessa classe de cabeçalho são criadas. Para disponibilizar a nova definição de cabeçalho para decodificar o uso das classes MQHeaderIterator ou MQHeaderList, você deveria registrá-la usando o MQHeaderRegistry. Observe que a classe de cabeçalho MQTM já é fornecida neste pacote e registrado no registro padrão.

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);

    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

Manipulando mensagens PCF com o IBM MQ classes for Java

Classes Java são fornecidas para criar e analisar mensagens estruturadas por PCF e para facilitar o envio de solicitações PCF e a coleta de respostas PCF.

Classes PCFMessage e MQCFGR representam matrizes de estruturas de parâmetros PCF. Elas fornecem métodos de conveniência para inclusão e recuperação de parâmetros PCF.

Estruturas de parâmetros PCF são representadas pelas classes MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL e MQCFGR. Elas compartilham interfaces operacionais básicas:

- Métodos para ler e gravar conteúdo da mensagem: read (), write () e size ()
- Métodos para manipular parâmetros: getValue (), setValue (), getParameter () e outros
- O método enumerador .nextParameter (), que analisa o conteúdo de PCF em um MQMessage

O parâmetro de filtro PCF é usado em comandos inquire para fornecer uma função de filtro. Ele está contido nas classes a seguir:

- MQCFIF - filtro de número inteiro
- MQCFSF - filtro de sequência
- MQCFBF - filtro de byte

Duas classes de agente, PCFAgent e PCFMessageAgent, são fornecidas para gerenciar a conexão com um Gerenciador de filas, a fila do servidor de comandos e uma fila de resposta associada. PCFMessageAgent estende PCFAgent e deve ser normalmente usado preferencialmente. A classe PCFMessageAgent converte os MQMessages recebidos e os transmite de volta ao responsável pela chamada como uma matriz PCFMessage. PCFAgent retorna uma matriz de MQMessages, que você precisa para analisar antes de usar.

Manipulando propriedades de mensagens no IBM MQ classes for Java

As chamadas de função para processar identificadores de mensagens não possuem nenhum IBM MQ classes for Java equivalente. Para configurar, retornar ou excluir propriedades de identificadores de mensagens, use métodos da classe MQMessage.

Para obter informações gerais sobre propriedades de mensagem, consulte "Nomes de propriedades" na página 28.

No acesso IBM MQ classes for Java a mensagens é através da classe MQMessage. Os identificadores de mensagens, portanto, não são fornecidos no ambiente do Java e não há chamadas de função do IBM MQ equivalentes a MQCRTMH, MQDLTMH, MQMHBUF e MQBUFMH

Para configurar as propriedades de identificadores de mensagens na interface processual, use a chamada MQSETMP. No IBM MQ classes for Java, use o método apropriado da classe MQMessage:

- setBooleanProperty
- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property
- setInt8Property
- setLongProperty
- setFloatProperty
- setDoubleProperty
- setStringProperty
- setObjectProperty

Estes são, às vezes, referidos coletivamente como os métodos *set*property*.

Para retornar o valor de propriedades de identificador de mensagens na interface processual, use a chamada MQINQMP. No IBM MQ classes for Java, use o método apropriado da classe MQMessage:

- getBooleanProperty
- getByteProperty
- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty
- getDoubleProperty
- getStringProperty

- getObjectProperty

Estes são, às vezes, referidos coletivamente como os métodos *get*property*.

Para excluir o valor de propriedades de identificadores de mensagens na interface processual, use a chamada MQDLTMP. No IBM MQ classes for Java, use o método deleteProperty da classe MQMessage.

Erros de manipulação em IBM MQ classes for Java

Erros de manipulador que surgem do IBM MQ classes for Java usando os blocos Java `try` e `catch`.

Os métodos na interface do Java não retornam um código de conclusão e código de razão. Em vez disso, eles lançam uma exceção sempre que o código de conclusão e código de razão resultam de uma chamada do IBM MQ e não são ambos zero. Isso simplifica a lógica do programa de forma que não seja necessário verificar os códigos de retorno após cada chamada para o IBM MQ. É possível decidir em quais pontos em seu programa você deseja lidar com a possibilidade de falha. Nestes pontos, é possível cercar seu código com blocos `try` e `catch`, como no exemplo a seguir:

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                           "CC = " + ex.completionCode +
                           "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

Os códigos de razão de chamada IBM MQ relatados de volta em Java exceções para z/OS são documentadas na conclusão da API e códigos de razão...

As exceções que são lançadas enquanto um aplicativo do IBM MQ classes for Java está em execução também são gravadas no log. No entanto, um aplicativo pode chamar o método MQException.logExclude() para evitar que as exceções associadas a um código de razão específico sejam registradas. Talvez você deseja fazer isso em situações em que espera que muitas exceções associadas a um código de razão específico sejam lançadas e não deseje que o log fique cheio com essas exceções. Por exemplo, se seu aplicativo tenta obter uma mensagem de uma fila toda vez que a iteração em um loop acontece e, para a maioria destas tentativas, você espera que nenhuma mensagem adequada esteja na fila, você pode desejar impedir que as exceções associadas ao código de razão MQRC_NO_MSG_AVAILABLE sejam registradas. Se um aplicativo evitou anteriormente que exceções associadas a um código de razão específico fossem registradas, ele pode permitir que essas exceções sejam registradas novamente, chamando o método MQException.logInclude().

Às vezes, o código de razão não transmite todos os detalhes associados ao erro. Para cada exceção que é lançada, um aplicativo deve verificar a exceção vinculada. A exceção vinculada sozinha pode ter outra exceção vinculada e assim as exceções vinculadas formam uma cadeia levando de volta ao problema subjacente original. Uma exceção vinculada é implementada usando o mecanismo de exceção em cadeia da classe java.lang.Throwable e um aplicativo obtém uma exceção vinculada chamando o método Throwable.getCause(). Em uma exceção que é uma instância de MQException, MQException.getCause() recupera a instância subjacente do com.ibm.mq.jmqi.JmqiException e getCause, a partir desta exceção, recupera a java.lang.Exception subjacente que causou o erro.

Obtendo e configurando valores de atributos no IBM MQ classes for Java

Os métodos getXXX() e setXXX() são fornecidos para muitos atributos comuns. Outros podem ser acessados usando os métodos inquire() e set() genéricos.

Para muitos dos atributos comuns, as classes MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess e MQQueueManager contêm os métodos getXXX() e setXXX(). Esses métodos permitem que você obtenha e configure seus valores de atributos. Observe que para MQDestination, MQQueue e MQTopic, os métodos funcionam apenas se você especificar a consulta apropriada e configurar sinalizadores quando abrir o objeto.

Para atributos menos comuns, as classes MQQueueManager, MQDestination, MQQueue, MQTopic e MQProcess herdam de uma classe chamada MQManagedObject. Esta classe define as interfaces inquire() e set().

Quando você cria um novo objeto do gerenciador de filas usando o operador *new*, ele é aberto automaticamente para consulta. Quando você usa o método accessProcess() para acessar um objeto do processo, esse objeto é automaticamente aberto para consulta. Quando você usa o método accessQueue() para acessar um objeto da fila, esse objeto não é aberto automaticamente para operações de consulta ou de configuração. Isso pode ocorrer porque incluir essas opções automaticamente pode causar problemas com alguns tipos de filas remotas. Para usar os métodos inquire, set, getXXX e setXXX em uma fila, deve-se especificar os sinalizadores de consulta e configuração apropriados no parâmetro openOptions do método accessQueue(). O mesmo é verdadeiro para os objetos de destino e de tópico.

Os métodos inquire e set utilizam três parâmetros:

- matriz de seletores
- matriz intAttrs
- matriz charAttrs

Os parâmetros SelectorCount, IntAttrCount e CharAttrLength, que estão localizados em MQINQ, não são necessários, pois o comprimento de uma matriz no Java sempre é conhecido. O exemplo a seguir mostra como fazer uma consulta em uma fila:

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Programas multiencadeados no Java

O ambiente de tempo de execução do Java é inerentemente multiencadeado. O IBM MQ classes for Java permite que um objeto do gerenciador de filas seja compartilhado por vários encadeamentos, mas assegura que todo acesso ao gerenciador de filas de destino seja sincronizado.

Programas multiencadeados são difíceis de evitar em Java. Considere um programa simples que se conecta a um gerenciador de filas e abre uma fila na inicialização. O programa exibe um único botão na tela. Quando um usuário clica nesse botão, o programa busca uma mensagem da fila.

O ambiente de tempo de execução do Java é inerentemente multiencadeado. Portanto, a inicialização do seu aplicativo ocorre em um encadeamento e o código que é executado em resposta ao pressionamento do botão é executado em um encadeamento separado (o encadeamento da interface com o usuário).

Com o IBM MQ MQI client baseado em C, isso pode causar um problema, porque há limitações para o compartilhamento de manipulações por vários encadeamentos. O IBM MQ classes for Java afrouxa esta restrição, permitindo que um objeto gerenciador de filas (e sua fila, tópico e objetos de processo associados) seja compartilhado por vários encadeamentos.

A implementação do IBM MQ classes for Java assegura que, para uma conexão específica (instância do objeto MQQueueManager), todo acesso ao gerenciador de filas de destino do IBM MQ seja sincronizado. Um encadeamento que deseja emitir uma chamada a um gerenciador de filas é bloqueado até que todas as outras chamadas em andamento para essa conexão sejam concluídas. Se você requerer acesso simultâneo ao mesmo gerenciador de filas a partir de múltiplos encadeamentos em seu programa,

crie um novo objeto MQQueueManager para cada encadeamento que requer acesso simultâneo. (Isto é equivalente a emitir uma chamada MQCONN separada para cada encadeamento.)

Nota: As instâncias da classe `com.ibm.mq.MQGetMessageOptions` não devem ser compartilhadas entre encadeamentos que estão solicitando mensagens simultaneamente. Instâncias dessa classe são atualizadas com dados durante a solicitação MQGET correspondente, que pode resultar em consequências inesperadas quando vários encadeamentos são operacionais simultaneamente na mesma instância do objeto.

Usando saídas de canal em IBM MQ classes for Java

Uma visão geral de como usar saídas do canal em um aplicativo usando o IBM MQ classes for Java.

Os tópicos a seguir descrevem como gravar uma saída do canal no Java, como designá-la e como transmitir dados para ela. Eles, em seguida, descrevem como usar saídas de canal gravadas em C e como usar uma sequência de saídas do canal.

Seu aplicativo deve ter a permissão de segurança correta para carregar a classe de saída do canal.

Criando uma saída do canal no IBM MQ classes for Java

É possível fornecer suas próprias saídas do canal definindo uma classe de Java que implemente uma interface apropriada.

Para implementar uma saída, defina uma nova classe Java que implemente a interface apropriada. Três interfaces de saída são definidas no pacote com.ibm.mq.exits:

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

Nota: Saídas do canal são suportadas apenas para conexões do cliente; elas não são suportadas para conexões de ligações. Não é possível usar uma saída do canal do Java fora do IBM MQ classes for Java, por exemplo, se você estiver usando um aplicativo cliente gravado em C.

Qualquer criptografia TLS definida para uma conexão é executada *após* as saídas de envio e de segurança serem chamadas. Da mesma forma, a decriptografia é executada *antes* das saídas de recebimento e de segurança serem chamadas.

A amostra a seguir define uma classe que implementa todas as três interfaces:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                       MQCD channelDefinition,
                                       ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                        MQCD channelDefinition,
                                        ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

Cada saída transmitiu um objeto MQCXP e um objeto MQCD. Estes objetos representam as estruturas MQCXP e MQCD definidas na interface processual.

Toda classe de saída gravada deve ter um construtor. Este pode ser o construtor padrão ou um que obtenha um argumento de sequência. Se ele obtiver uma sequência, os dados do usuário serão transmitidos para a classe de saída quando ela for criada. Se a classe de saída contiver um construtor padrão e um construtor de argumento único, o construtor de argumento único terá prioridade.

Para as saídas de envio e de segurança, seu código de saída deve retornar os dados que deseja enviar para o servidor. Para uma saída de recebimento, seu código de saída deve retornar os dados modificados que você deseja que o IBM MQ interprete.

O corpo de saída mais simples possível é:

```
{ return agentBuffer; }
```

Não feche o gerenciador de filas de dentro de uma saída do canal.

Usando classes de saída do canal existentes

Em versões do IBM MQ anteriores à 7.0, você deve implementar estas saídas usando as interfaces MQSendExit, MQReceiveExit e MQSecurityExit, como no exemplo a seguir. Este método permanece válido, mas o novo método é preferido para funcionalidade e desempenho aprimorados.

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

Designando uma saída do canal no IBM MQ classes for Java
É possível designar uma saída do canal usando o IBM MQ classes for Java.

Não há nenhum equivalente direto para o canal do IBM MQ no IBM MQ classes for Java. As saídas de canal são designadas a um MQQueueManager. Por exemplo, tendo definido uma classe que implementa a interface WMQSecurityExit, um aplicativo pode usar a saída de segurança em uma de quatro maneiras:

- Ao designar uma instância da classe para o campo MQEnvironment.channelSecurityExit antes de criar um objeto MQQueueManager
- Configurar o campo MQEnvironment.channelSecurityExit como uma sequência que representa a classe de saída de segurança antes de criar um objeto MQQueueManager
- Ao criar um par de chave/valor na hashtable de propriedades transmitida para MQQueueManager com uma chave de CMQC.SECURITY_EXIT_PROPERTY
- Usando uma Tabela de Definição de Canal do Cliente (CCDT)

Qualquer saída designada configurando o campo MQEnvironment.channelSecurityExit para uma sequência, criando um par de chave/valor na hashtable de propriedades ou usando um CCDT, deve ser gravada com um construtor padrão. Uma saída designada como uma instância de uma classe não precisa de um construtor padrão, dependendo do aplicativo.

Um aplicativo pode usar uma saída de envio ou de recebimento de forma semelhante. Por exemplo, o fragmento de código a seguir mostra como usar a segurança, saídas de envio e recebimento que são implementados na classe MyMQExits, que foi definida anteriormente, usando MQEnvironment:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Se mais de um método for usado para designar uma saída do canal, a ordem de precedência será a seguinte:

1. Se a URL de uma CCDT for transmitida para o MQQueueManager, o conteúdo da CCDT determina as saídas do canal a serem usadas e quaisquer definições de saída em MQEnvironment ou hashtable de propriedades são ignoradas.
2. Se nenhuma URL CCDT for transmitida, as definições de saída de MQEnvironment e hashtable serão mescladas
   - Se o mesmo tipo de saída for definido em ambos MQEnvironment e a hashtable, a definição na hashtable será usada.
   - Se tipos de saída novos e antigos equivalentes forem especificados (por exemplo, o campo sendExit, que só pode ser usado para o tipo de saída usado em versões anteriores a IBM WebSphere MQ 7.0 e o campo channelSendExit, que pode ser usado para qualquer saída de envio), a nova saída (channelSendExit) será usada em vez da saída antiga.

Se você tiver declarado uma saída do canal como uma sequência, deverá ativar o IBM MQ para localizar o programa de saída do canal. É possível fazer isso de várias maneiras, dependendo do ambiente no qual o aplicativo está em execução e sobre como os programas de saída de canal são compactados.

- Para um aplicativo que está em execução em um servidor de aplicativos, deve-se armazenar os arquivos no diretório mostrado em Tabela 58 na página 401 ou compactados em arquivos JAR referenciados por **exitClasspath**.
- Para um aplicativo que não está em execução em um servidor de aplicativos, as regras a seguir se aplicam:
  - Se as classes de saída do canal forem compactadas em arquivos JAR separados, esses arquivos JAR deverão ser incluídos no **exitClasspath**.
  - Se as classes de saída do canal não forem compactadas em arquivos JAR, os arquivos de classe podem ser armazenados no diretório mostrado em Tabela 58 na página 401 ou em qualquer diretório no caminho de classe do sistema da JVM ou **exitClasspath**.

A propriedades **exitClasspath** pode ser especificada de quatro maneiras. Em ordem de prioridade, essas maneiras são as seguintes:

1. A propriedade do sistema com.ibm.mq.exitClasspath (definida na linha de comandos usando a opção `-D`)
2. A sub-rotina exitPath do arquivo `mqclient.ini`
3. Uma entrada de hashtable com a chave CMQC.EXIT_CLASSPATH_PROPERTY
4. A variável MQEnvironment **exitClasspath**

Separe vários caminhos usando o caractere java.io.File.pathSeparator.

Tabela 58. O diretório para programas de saída de canal	
Plataforma	**Diretório**
AIX AIX Linux Linux	`/var/mqm/exits` (programas de saída de canal de 32 bits) `/var/mqm/exits64` (programas de saída de canal de 64 bits)
Windows	*install_data_dir* \exits

Nota: *install_data_dir* é o diretório que você escolheu para os arquivos de dados do IBM MQ durante a instalação. O diretório padrão é `C:\ProgramData \IBM \MQ`.

Passando dados para saídas do canal no IBM MQ classes for Java
É possível passar saídas de canal e retornar dados de saídas de canal para seu aplicativo.

O parâmetro agentBuffer

Para uma saída de envio, o parâmetro *agentBuffer* contém os dados que estão prestes a serem enviados. Para uma saída de recebimento ou uma saída de segurança, o parâmetro *agentBuffer* contém os dados que acabam de ser recebidos. Você não precisa de um parâmetro de comprimento, porque a expressão agentBuffer.limit() indica o comprimento da matriz.

Para as saídas de envio e de segurança, seu código de saída deve retornar os dados que deseja enviar para o servidor. Para uma saída de recebimento, seu código de saída deve retornar os dados modificados que você deseja que o IBM MQ interprete.

O corpo de saída mais simples possível é:

```
{ return agentBuffer; }
```

Saídas de canal são chamadas com um buffer que tem uma matriz auxiliar. Para melhor desempenho, a saída deve retornar um buffer com uma matriz auxiliar.

Dados do usuário

Se um aplicativo se conecta a um gerenciador de filas, configurando channelSecurityExit, channelSendExit ou channelReceiveExit, 32 bytes de dados do usuário podem ser passados para a classe de saída do canal apropriada quando chamada, usando os campos channelSecurityExitUserData, channelSendExitUserData ou channelReceiveExitUserData. Esses dados do usuário estão disponíveis para a classe de saída do canal, mas são atualizados sempre que a saída for chamada. Quaisquer mudanças feitas nos dados do usuário na saída do canal serão, portanto, perdidas. Se deseja fazer mudanças persistentes nos dados em uma saída de canal, use exitUserArea de MQCXP. Os dados nesse campo são mantidos entre as chamadas da saída.

Se o aplicativo configurar securityExit, sendExit ou receiveExit, nenhum dado do usuário poderá ser passado para essas classes de saída do canal.

Se um aplicativo usar uma tabela de definição de canal de cliente (CCDT) para se conectar a um gerenciador de filas, quaisquer dados do usuário especificados em uma definição de canal de conexão de cliente serão passados para classes de saída do canal quando forem chamadas. Para obter mais informações sobre como usar uma tabela de definição de canal do cliente, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for Java” na página 383.

Usando saídas do canal não escritas em Java com o IBM MQ classes for Java
Como usar programas de saída do canal escritos em C a partir de um aplicativo Java.

No IBM MQ, é possível especificar o nome de um programa de saída do canal escrito em C como uma String passada para os campos channelSecurityExit, channelSendExit ou channelReceiveExit no objeto

MQEnvironment ou propriedades Hashtable. No entanto, não é possível usar uma saída do canal escrita em Java em um aplicativo escrito em outra linguagem.

Especifique o nome do programa de saída no formato `library(function)` e assegure que o local do programa de saída esteja especificado conforme descrito em Caminho para saídas.

Para obter informações sobre como escrever uma saída do canal em C, consulte “Programas de Saída de Canal para Canais de Mensagens” na página 973.

Usando uma sequência de saídas de envio ou recebimento de canal no IBM MQ classes for Java
Um aplicativo IBM MQ classes for Java pode usar uma sequência de saídas de envio ou recebimento de canal executadas sucessivamente.

Para usar uma sequência de saídas de envio, um aplicativo pode criar um List ou String contendo as saídas de envio. Se um List for usado, cada elemento de List poderá ser qualquer um dos seguintes:

- Uma instância de uma classe definida pelo usuário que implementa a interface WMQSendExit
- Uma instância de uma classe definida pelo usuário que implementa a interface MQSendExit (para uma saída de envio gravada em Java)
- Uma instância da classe MQExternalSendExit (para uma saída de envio não gravada em Java)
- Uma instância da classe MQSendExitChain
- Uma instância da classe String

Um List não pode conter outro List.

O aplicativo pode usar uma sequência de saídas de recebimento de maneira semelhante.

Se um String for usado, ele deverá consistir de uma ou mais definições de saída separadas por vírgula, cada uma das quais poderá ser o nome de uma classe Java ou um programa C no formato `library(function)`.

O aplicativo então designará o objeto List ou String ao campo MQEnvironment.channelSendExit antes de criar um objeto MQQueueManager.

O contexto de informações transmitidas às saídas está unicamente no domínio das saídas. Por exemplo, se uma saída Java e uma saída C estiverem encadeadas, a presença da saída Java não terá nenhum efeito na saída C.

Usando as classes de cadeia de saída

Em versões anteriores a IBM WebSphere MQ 7.0, duas classes foram fornecidas para permitir sequências de saídas:

- MQSendExitChain, que implementa a interface MQSendExit
- MQReceiveExitChain, que implementa a interface MQReceiveExit

O uso dessas classes permanece válido, mas o novo método é preferencial. O uso das interfaces do IBM MQ Classes for Java significa que o aplicativo ainda tem uma dependência do `com.ibm.mq.jar`. Se o novo conjunto de interfaces no pacote `com.ibm.mq.exits` for usado, não haverá dependência do `com.ibm.mq.jar`.

Para usar uma sequência de saídas de envio, um aplicativo criou uma lista de objetos, em que cada objeto era um dos seguintes:

- Uma instância de uma classe definida pelo usuário que implementa a interface MQSendExit (para uma saída de envio gravada em Java)
- Uma instância da classe MQExternalSendExit (para uma saída de envio não gravada em Java)
- Uma instância da classe MQSendExitChain

O aplicativo criou um objeto MQSendExitChain transmitindo esta lista de objetos como um parâmetro no construtor. O aplicativo teria então designado o objeto MQSendExitChain para o campo MQEnvironment.sendExit antes de criar um objeto MQQueueManager.

Compactação de canal no IBM MQ classes for Java

A compactação de dados que fluem em um canal pode melhorar o desempenho do canal e reduzir o tráfego de rede. IBM MQ classes for Java usa a função de compactação construída em IBM MQ.

Usando a função fornecida com o IBM MQ, é possível compactar os dados que fluem em canais de mensagens e canais MQI e, em qualquer tipo de canal, é possível compactar dados de cabeçalho e dados de mensagens independentemente um do outro. Por padrão, nenhum dados é compactado em um canal. Para obter uma descrição completa da compactação de canal, incluindo como ela é implementada no IBM MQ, consulte Compactação de dados (COMPMSG) e Compactação de cabeçalho (COMPHDR).

Um aplicativo IBM MQ classes for Java especifica as técnicas que podem ser usadas para compactar o cabeçalho ou dados da mensagem em uma conexão do cliente criando um objeto java.util.Collection. Cada técnica de compactação é um objeto Integer na coleta e a ordem na qual o aplicativo inclui as técnicas de compactação para a coleta é a ordem na qual as técnicas de compactação serão negociadas com o gerenciador de filas quando a conexão do cliente for iniciada. O aplicativo pode então designar a coleta para o campo hdrCompList, para os dados do cabeçalho ou o msgCompList, para dados da mensagem, na classe MQEnvironment. Quando o aplicativo estiver pronto, ele poderá iniciar a conexão do cliente criando um objeto MQQueueManager.

Os seguintes fragmentos de código ilustram a abordagem descrita. O primeiro fragmento de código mostra como implementar a compactação de dados de cabeçalho:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

O segundo fragmento de código mostra como implementar a compactação dos dados da mensagem:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_LZ4HIGH));
:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

No segundo exemplo, as técnicas de compactação serão negociadas na ordem RLE, em seguida, ZLIBHIGH, quando a conexão do cliente for iniciada. A técnica de compactação selecionada não pode ser mudada durante o tempo de vida do objeto MQQueueManager.

As técnicas de compactação para os dados de cabeçalho e de mensagens que são suportadas pelo gerenciador de filas e o cliente em uma conexão do cliente são transmitidas para uma saída de canal como coletas nos campos hdrCompList e msgCompList de um objeto MQChannelDefinition. As técnicas reais que estão sendo atualmente usadas para compactar os dados de cabeçalho e mensagens em uma conexão do cliente são transmitidas para uma saída de canal nos campos CurHdrCompression e CurMsgCompression de um objeto MQChannelExit.

Se a compactação for usada em uma conexão do cliente, os dados serão compactados antes de quaisquer saídas de envio do canal serem processadas e extraídas após quaisquer saídas de recebimento de canal serem processadas. Os dados transmitidos para enviar e receber saídas está, portanto, em um estado compactado.

Para obter mais informações sobre como especificar técnicas de compactação e sobre técnicas de compactação disponíveis, consulte Classe com.ibm.mq.MQEnvironment e Interface com.ibm.mq.MQC.

Compartilhando uma conexão TCP/IP em IBM MQ classes for Java

Várias instâncias de um canal MQI podem ser feitas para compartilhar uma única conexão TCP/IP.

No IBM MQ classes for Java, usa-se a variável MQEnvironment.sharingConversations para controlar o número de conversas que podem compartilhar uma única conexão TCP/IP.

O atributo SHARECNV é uma abordagem de melhor esforço para o compartilhamento de conexão. Portanto, quando um valor SHARECNV maior que 0 é usado com o IBM MQ classes for Java, não é garantido que um novo pedido de conexão irá sempre compartilhar uma conexão já estabelecida.

Definição do conjunto de conexões em IBM MQ classes for Java

IBM MQ classes for Java permite conexões sobressalentes para ser agrupadas para reutilização.

IBM MQ classes for Java fornece suporte adicional para aplicativos que lidam com várias conexões para os gerenciadores de filas do IBM MQ. Quando uma conexão não for mais necessária, em vez de destruí-la, ela poderá ser agrupada e reutilizada posteriormente. Isso pode fornecer um aprimoramento de desempenho substancial para aplicativos e middleware conectados de forma serial aos gerenciadores de filas arbitrários.

O IBM MQ fornece um conjunto de conexões padrão. Os aplicativos podem ativar ou desativar esse conjunto de conexões registrando e removendo o registro de tokens através da classe MQEnvironment. Se o conjunto estiver ativo quando o IBM MQ classes for Java construir um objeto MQQueueManager, ele irá procurar esse conjunto padrão e reutilizará qualquer conexão adequada. Quando uma chamada MQQueueManager.disconnect() ocorrer, a conexão subjacente será retornada ao conjunto.

Como alternativa, os aplicativos podem construir um conjunto de conexões MQSimpleConnectionManager para um uso específico. Em seguida, o aplicativo pode especificar esse conjunto durante a construção de um objeto MQQueueManager ou transmitir esse conjunto para MQEnvironment para usar como o conjunto de conexões padrão.

Para evitar que conexões usem muitos recursos, é possível limitar o número total de conexões que um objeto MQSimpleConnectionManager pode manipular e o tamanho do conjunto de conexões. A configuração de limites será útil se houver demandas conflitantes para conexões dentro de uma JVM.

Por padrão, o método getMaxConnections() retorna o valor de zero, o que significa que não há limite para o número de conexões que o objeto MQSimpleConnectionManager pode manipular. É possível configurar um limite usando o método setMaxConnections(). Se você configurar um limite e o limite for atingido, uma solicitação para uma conexão adicional poderá fazer com que uma MQException seja lançada, com um código de razão de MQRC_MAX_CONNS_LIMIT_REACHED.

Controlando o conjunto de conexões padrão em IBM MQ classes for Java

Este exemplo mostra como usar o conjunto de conexões padrão.

Considere o aplicativo de exemplo a seguir, MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 usa uma lista de gerenciadores de filas locais a partir da linha de comandos, se conecta a cada sucessivamente e executa alguma operação. No entanto, quando a linha de comandos listar o mesmo gerenciador de filas várias vezes, será mais eficiente para se conectar apenas uma vez e reutilizar essa conexão várias vezes.

IBM MQ classes for Java fornece um conjunto de conexões padrão que é possível usar para fazer isso. Para ativar o conjunto, use um dos métodos MQEnvironment.addConnectionPoolToken(). Para desativar o conjunto, use MQEnvironment.removeConnectionPoolToken().

O aplicativo de exemplo a seguir, MQApp2, é funcionalmente idêntico ao MQApp1, mas se conecta apenas uma vez para cada gerenciador de filas.

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

A primeira linha em negrito ativa o conjunto de conexões padrão registrando um objeto MQPoolToken com MQEnvironment.

O construtor MQQueueManager agora procura esse conjunto para uma conexão apropriada e apenas criará uma conexão com o gerenciador de filas se ele não puder localizar uma existente. A chamada qmgr.disconnect() retorna a conexão ao conjunto para reutilização posterior. Essas chamadas de API são as mesmas que o aplicativo de amostra MQApp1.

A segunda linha destacada desativa o conjunto de conexões padrão, que destrói quaisquer conexões do gerenciador de filas armazenadas no conjunto. Isso é importante porque, caso contrário, o aplicativo finalizaria com um número de conexões do gerenciador de filas ativas no conjunto. Esta situação poderia causar erros que apareceriam nos logs do gerenciador de filas.

Se um aplicativo usar uma tabela de definição de canal do cliente (CCDT) para se conectar a um gerenciador de filas, o construtor MQQueueManager procurará primeiramente a tabela para uma definição de canal de conexão do cliente adequada. Se um for localizado, o construtor irá procurar o conjunto de conexões padrão para uma conexão que pode ser usada para o canal. Se o construtor não puder localizar uma conexão adequada no conjunto, ele irá procurar então na tabela de definições de canal do cliente pela próxima definição adequada do canal de conexão do cliente e continuará conforme descrito anteriormente. Se o construtor concluir sua procura da tabela de definição de canal do cliente e não localizar nenhuma conexão adequada no conjunto, o construtor iniciará uma segunda procura da tabela. Durante essa procura, o construtor tentará criar uma nova conexão para cada definição adequada de canal de conexão do cliente, por sua vez, e usará a primeira conexão que ele gerencia para criar.

O conjunto de conexões padrão armazena um máximo de dez conexões não usadas e mantém essas conexões ativas por um máximo de cinco minutos. O aplicativo pode alterar isto (para obter detalhes, consulte “Fornecendo um conjunto de conexões diferente no IBM MQ classes for Java” na página 406).

Em vez de usar MQEnvironment para fornecer um MQPoolToken, o aplicativo pode construir seu próprio:

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

Alguns aplicativos ou fornecedores de middleware fornecem subclasses de MQPoolToken para transmitir informações a um conjunto de conexões customizado. Eles podem ser construídos e transmitidos para addConnectionPoolToken() desta maneira para que as informações extras possam ser transmitidas para o conjunto de conexões.

O conjunto de conexões padrão e vários componentes em IBM MQ classes for Java
Este exemplo mostra como incluir ou remover MQPoolTokens de um conjunto estático de objetos MQPoolToken registrados.

MQEnvironment contém um conjunto estático de objetos MQPoolToken registrados. Para incluir ou remover MQPoolTokens desse conjunto, use os métodos a seguir:

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

Um aplicativo pode consistir em muitos componentes que existem independentemente e executam trabalho usando um gerenciador de filas. Nesse aplicativo, cada componente deve incluir um MQPoolToken no conjunto de MQEnvironment durante seu ciclo de vida.

Por exemplo, o aplicativo de exemplo MQApp3 cria dez encadeamentos e inicia cada um. Cada encadeamento registra seu próprio MQPoolToken, aguarda um período de tempo, em seguida, conecta-se ao gerenciador de filas. Depois que o encadeamento é desconectado, ele remove seu próprio MQPoolToken.

O conjunto de conexões padrão permanece ativo enquanto houver pelo menos um token no conjunto de MQPoolTokens, portanto, permanecerá ativo na duração desse aplicativo. O aplicativo não precisa manter um objeto principal em controle geral dos encadeamentos.

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

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

Fornecendo um conjunto de conexões diferente no IBM MQ classes for Java
Este exemplo mostra como usar a classe **com.ibm.mq.MQSimpleConnectionManager** para fornecer um conjunto de conexões diferente.

Essa classe fornece recursos básicos para definição do conjunto de conexões e os aplicativos podem usar essa classe para customizar o comportamento do conjunto.

Após ser instanciada, um MQSimpleConnectionManager pode ser especificado no construtor MQQueueManager. O MQSimpleConnectionManager, em seguida, gerencia a conexão subjacente ao MQQueueManager construído. Se o MQSimpleConnectionManager contiver uma conexão agrupada adequada, essa conexão será reutilizada e retornada ao MQSimpleConnectionManager após uma chamada MQQueueManager.disconnect().

O fragmento de código a seguir demonstra esse comportamento:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

A conexão que é forjada durante o primeiro construtor MQQueueManager é armazenada em myConnMan após a chamada qmgr.disconnect(). A conexão é, então, reutilizada durante a segunda chamada ao construtor MQQueueManager.

A segunda linha ativa o MQSimpleConnectionManager. A última linha desativa MQSimpleConnectionManager, destruindo quaisquer conexões mantidas no conjunto. Um MQSimpleConnectionManager está, por padrão, em MODE_AUTO, que é descrito posteriormente nesta seção.

Um MQSimpleConnectionManager aloca conexões com base nas usadas mais recentemente e destrói conexões com base nas usadas menos recentemente. Por padrão, uma conexão será destruída se não tiver sido usada por cinco minutos ou se houver mais de dez conexões não usadas no conjunto. É possível alterar esses valores chamando MQSimpleConnectionManager.setTimeout().

Também é possível configurar um MQSimpleConnectionManager para uso como o conjunto de conexões padrão, para ser usado quando nenhum Connection Manager for fornecido no construtor MQQueueManager.

O aplicativo a seguir demonstra isso:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

As linhas em negrito criam e configuram um objeto MQSimpleConnectionManager. A configuração faz o seguinte:

- Finaliza conexões que não são usadas para uma hora
- Limita o número de conexões gerenciadas por myConnMan para 75
- Limita o número de conexões não utilizadas no conjunto para 50
- Configura MODE_AUTO, que é o padrão. Isso significa que o conjunto está ativo somente se ele for o gerenciador de conexões padrão e houver pelo menos um token no conjunto de MQPoolTokens mantido por MQEnvironment.

O novo MQSimpleConnectionManager é, então, configurado como o gerenciador de conexões padrão.

Na última linha, o aplicativo chama MQApp3.main(). Isso executa inúmeros encadeamentos, em que cada um deles usa IBM MQ independentemente. Esses encadeamentos usam myConnMan quando forjam conexões.

Coordenação JTA/JDBC usando o IBM MQ classes for Java

O IBM MQ classes for Java suporta o método MQQueueManager.begin(), que permite que o IBM MQ aja como um coordenador para um banco de dados que fornece um driver compatível tipo JDBC 2 ou JDBC tipo 4.

Esse suporte não está disponível em todas as plataformas. Para verificar quais plataformas suportam coordenação de JDBC, consulte Requisitos do sistema para IBM MQ.

Para usar o suporte XA-JTA, deve-se usar a biblioteca de comutação de JTA especial. O método de usar essa biblioteca varia dependendo se estiver usando o Windows ou uma das outras plataformas.

Configurando a coordenação JTA/JDBC no Windows

A biblioteca XA é fornecida como uma DLL com um nome do formato `jdbcxxx.dll`.

O `jdbcora12.dll` fornecido tem compatibilidade com o Oracle 12C, para uma instalação do servidor IBM MQ para Windows.

Em sistemas Windows, a biblioteca XA é fornecida como uma DLL completa. O nome desta DLL é `jdbcxxx.dll`, em que `xxx` indica o banco de dados para o qual a biblioteca de comutação foi compilada. Esta biblioteca está no diretório `java\lib\jdbc` ou `java\lib64\jdbc` de sua instalação do IBM MQ classes for Java. Deve-se declarar a biblioteca XA, também descrita como o arquivo de carregamento do comutador, para o gerenciador de filas. Use o IBM MQ Explorer. Especifique os detalhes do arquivo de carregamento do comutador no painel de propriedades do gerenciador de filas, sob o gerenciador de recursos XA. Deve-se fornecer somente o nome da biblioteca. Por exemplo:

Para um banco de dados Db2, configure o campo SwitchFile como: `dbcdb2`

Para um banco de dados Oracle, configure o campo SwitchFile como: `jdbcora`

Notas:

1. O Oracle 12C é suportado pelo IBM MQ classes for Java somente no IBM MQ for Windows.
2. A versão suportada do Oracle 12C é a 12.1.0.1.0 Enterprise Edition e fix packs futuros.
3. Os bancos de dados Oracle de 64 bits no Windows de 64 bits requerem o cliente do Oracle de 32 bits.
4. Usando o IBM MQ classes for Java, o IBM MQ pode agir como um coordenador de transação. No entanto, não é possível participar de uma transação no estilo de JTA.

Configurando a coordenação de JTA/JDBC em plataformas diferentes do Windows

Arquivos de objeto são fornecidos. Vincule o apropriado usando o makefile fornecido e declare-o para o gerenciador de filas usando o arquivo de configuração.

Para cada sistema de gerenciamento de banco de dados, o IBM MQ fornece dois arquivos de objeto. Deve-se vincular um arquivo de objeto para criar uma biblioteca de comutação de 32 bits e vincular o outro arquivo de objeto para criar uma biblioteca de comutação de 64 bits. Para Db2, o nome de cada arquivo de objeto é `jdbcdb2.o`. Para Oracle, o nome de cada arquivo de objeto é `jdbcora.o`.

Deve-se vincular cada arquivo de objeto usando o makefile apropriado fornecido com o IBM MQ. Uma biblioteca de comutação requer outras bibliotecas, que podem ser armazenadas em locais diferentes em sistemas diferentes. No entanto, uma biblioteca de comutação não pode utilizar a variável de ambiente do caminho da biblioteca para localizar essas bibliotecas porque a biblioteca de comutação é carregada pelo gerenciador de filas, que é executado em um ambiente `setuid`. O makefile fornecido, portanto, assegura que uma biblioteca de comutação contém os nomes de caminhos completos dessas bibliotecas.

Para criar uma biblioteca de comutação, insira um comando **make** com o formato a seguir. Para criar uma biblioteca de comutadores de 32 bits, digite o comando no diretório `/java/lib/jdbc` da instalação do IBM MQ. Para criar uma biblioteca de comutadores de 64 bits, digite o comando no diretório `/java/lib64/jdbc`.

```
make DBMS
```

em que *DBMS* é o sistema de gerenciamento de banco de dados para o qual você está criando a biblioteca de comutação. Os valores válidos são db2 para Db2 e `oracle` para Oracle.

Nota:

- Para executar aplicativos de 32 bits, deve-se criar uma biblioteca de comutação de 32 bits e uma de 64 bits para cada sistema de gerenciamento de banco de dados que você está usando. Para executar aplicativos de 64 bits, é necessário criar somente uma biblioteca de comutação de 64 bits. Para Db2, o nome de cada biblioteca de comutação é `jdbcdb2` e para Oracle, o nome de cada biblioteca de comutação é `jdbcora`. Os makefiles asseguram que as bibliotecas de comutação de 32 bits e de 64 bits sejam armazenadas em diferentes diretórios do IBM MQ. Uma biblioteca de comutadores de 32 bits é armazenada no diretório `/java/lib/jdbc` e uma biblioteca de comutadores de 64 bits é armazenada no diretório `/java/lib64/jdbc`.
- Uma vez que é possível instalar o Oracle em qualquer lugar em um sistema, os makefiles usam a variável de ambiente **ORACLE_HOME** para localizar onde o Oracle está instalado.
- Se IBM MQ for instalado em um local diferente do local padrão, altere o valor de **MQ_INSTALLATION_PATH** no makefile.

Após a criação das bibliotecas de comutação para Db2, Oracle ou ambos, deve-se declará-las ao seu gerenciador de filas. Se o arquivo de configuração do gerenciador de filas (qm.ini) já contém sub-rotinas `XAResourceManager` para bancos de dados Db2 ou Oracle, deve-se substituir a entrada `SwitchFile` em cada sub-rotina por um dos seguintes:

Para um banco de dados Db2

```
SwitchFile=jdbcdb2
```

Para um banco de dados Oracle

```
SwitchFile=jdbcora
```

Não especifique o nome do caminho completo da biblioteca de comutação de 32 bits ou de 64 bits. Especifique somente o nome da biblioteca.

Se o arquivo de configuração do gerenciador de filas não já contém sub-rotinas `XAResourceManager` para bancos de dados Db2 ou Oracle, ou se você deseja incluir sub-rotinas `XAResourceManager` adicionais, consulte Administrando IBM MQ para informações sobre como construir uma sub-rotina `XAResourceManager`. No entanto, cada entrada `SwitchFile` em uma nova sub-rotina `XAResourceManager` deve ser exatamente conforme descrito anteriormente para um banco de dados Db2 or Oracle. Deve-se incluir também a entrada `ThreadOfControl=PROCESS`.

Após atualizar o arquivo de configuração do gerenciador de filas e certificar-se de que todas as variáveis de ambiente de banco de dados apropriadas foram configuradas, será possível reiniciar o gerenciador de filas.

Usando a coordenação de JTA/JDBC
Codifique suas chamadas API como no exemplo fornecido.

A sequência básica de chamadas API para um aplicativo de usuário é:

```
 qMgr = new MQQueueManager("QM1")
 Connection con = qMgr.getJDBCConnection( xads );
 qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

 qMgr.commit() or qMgr.backout();
 con.close()
 qMgr.disconnect()
```

`xads` na chamada getJDBCConnection é uma implementação específica do banco de dados da interface XADataSource, que define os detalhes do banco de dados ao qual se conectar. Consulte a documentação de seu banco de dados para determinar como criar um objeto XADataSource apropriado para passar no getJDBCConnection.

Deve-se também atualizar o caminho de classe com os arquivos jar específicos do banco de dados apropriados executando o trabalho do JDBC.

Se você precisar se conectar a diversos bancos de dados, deverá chamar getJDBCConnection várias vezes para executar a transação em várias conexões diferentes.

Há duas formas de getJDBCConnection, refletindo as duas formas de XADataSource.getXAConnection:

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

Esses métodos declaram Exceção em suas cláusulas throws para evitar problemas com o verificador da JVM para clientes que não estão usando as funções JTA. A exceção real lançada é javax.transaction.xa.XAException que requer que o arquivo jta.jar seja incluído no caminho de classe para os programas que não o requeriam antes.

Para usar o suporte JTA/JDBC, deve-se incluir a instrução a seguir em seu aplicativo:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Problemas e limitações conhecidos com a coordenação de JTA/JDBC
Alguns dos problemas e limitações de suporte JTA/JDBC dependem do sistema de gerenciamento de banco de dados em uso, por exemplo, drivers JDBC testados se comportam de forma diferente quando o banco de dados é encerrado enquanto um aplicativo está em execução. Se a conexão com o banco de dados que um aplicativo está usando é quebrada, há etapas que o aplicativo pode executar para restabelecer uma nova conexão com o gerenciador de filas e o banco de dados, para que seja possível usar essas novas conexões para executar o trabalho transacional necessário.

Como o suporte JTA/JDBC faz chamadas para drivers JDBC, a implementação desses drivers JDBC pode ter um efeito significativo no comportamento do sistema. Especificamente, drivers JDBC testados se comportam de forma diferente quando o banco de dados é encerrado enquanto um aplicativo está em execução.

Importante: Sempre evite encerrar abruptamente um banco de dados enquanto houver aplicativos que estão mantendo conexões abertas com ele.

Nota: Um aplicativo IBM MQ classes for Java deve se conectar usando o modo de ligações para fazer o IBM MQ agir como um coordenador de banco de dados.

Diversas sub-rotinas XAResourceManager
O uso de mais de uma sub-rotina XAResourceManager em um arquivo de configuração do gerenciador de filas, `qm.ini`, não é suportado. Qualquer sub-rotina XAResourceManager diferente da primeira será ignorada.

Db2

Às vezes, o Db2 retorna um erro SQL0805N. Esse problema pode ser resolvido com o comando CLP a seguir:

```
DB2 bind @db2cli.lst blocking all grant public
```

Para obter mais informações, consulte a documentação do Db2.

A sub-rotina XAResourceManager deve ser configurada para usar ThreadOfControl=PROCESS. Para Db2 8.1 e superior, isso não corresponde ao encadeamento padrão de configuração de controle para Db2, portanto, toc=p deve ser especificado na sequência aberta de XA. Uma sub-rotina XAResourceManager de exemplo para o Db2 com a coordenação de JTA/JDBC é a seguinte:

```
XAResourceManager:
```

```
Name=jdbcdb2
SwitchFile=jdbcdb2
XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
ThreadOfControl=PROCESS
```

Isso não evita que os aplicativos Java que usam a coordenação de JTA/JDBC sejam eles mesmos multiencadeados.

Oracle

Chamar o método JDBC Connection.close() após MQQueueManager.disconnect() gera uma SQLException. Chame Connection.close() antes de MQQueueManager.disconnect() ou omita a chamada a Connection.close().

Manipulando problemas com conexões com o banco de dados

Quando um aplicativo IBM MQ classes for Java usa o suporte JTA/JDBC que é fornecido pelo IBM MQ, ele geralmente executa as seguintes etapas:

1. Cria um novo objeto MQQueueManager para representar uma conexão com o gerenciador de filas que agirá como o gerenciador de transações.
2. Constrói um objeto XADataSource que contém detalhes sobre como se conectar ao banco de dados que será inscrito na transação.
3. Chama o método MQQueueManager.getJDBCConnection(XADataSource) passando o XADataSource que foi criado anteriormente. Isso faz com que o IBM MQ classes for Java para estabeleça uma conexão com o banco de dados.
4. Chama o método MQQueueManager.begin() para iniciar a transação XA.
5. Executa o sistema de mensagens e o trabalho do banco de dados.
6. Quando todo o trabalho necessário foi concluído, chame o método MQQueueManager.commit (). Isso concluirá a transação XA.
7. Se uma nova transação XA é necessária nesse ponto, o aplicativo pode repetir as etapas 4, 5 e 6.
8. Quando o aplicativo é concluído, ele deve fechar a conexão com o banco de dados que foi criada na etapa 3 e, em seguida, chamar o método MQQueueManager.disconnect() para desconectar do gerenciador de filas.

O IBM MQ classes for Java mantém uma lista interna de todas as conexões do banco de dados que são criadas quando um aplicativo chama MQQueueManager.getJDBCConnection(XADataSource). Se um gerenciador de filas precisa se comunicar com o banco de dados durante o processamento da transação XA, o processamento a seguir ocorre:

1. O gerenciador de filas chama no IBM MQ classes for Java, transmitindo os detalhes da chamada XA que precisam ser passados para o banco de dados.
2. O IBM MQ classes for Java, então, consulta a conexão apropriada na lista e, em seguida, usa essa conexão para o fluxo de chamada XA para o banco de dados.

Se a conexão com o banco de dados for perdida em qualquer ponto durante esse processamento, o aplicativo deverá:

1. Voltar qualquer trabalho existente que foi feito sob a transação, chamando o método MQQueueManager.backout().
2. Fechar a conexão de banco de dados. Isso deve fazer com que o IBM MQ classes for Java remova os detalhes da conexão de dados quebrada de sua lista interna.
3. Desconectar do gerenciador de filas, chamando o método MQQueueManager.disconnect().
4. Estabelecer uma nova conexão com o gerenciador de filas, construindo um novo objeto MQQueueManager.
5. Criar uma nova conexão com o banco de dados, chamando o método MQQueueManager.getJDBCConnection(XADataSource).
6. Executar o trabalho transacional novamente.

Isso permite que o aplicativo restabeleça uma nova conexão com o gerenciador de filas e com o banco de dados e, em seguida, use essas conexões para executar o trabalho transacional necessário.

Suporte a Segurança da Camada de Transporte (TLS) no IBM MQ classes for Java

Os aplicativos cliente IBM MQ classes for Java suportam a criptografia do TLS. Você requer que um provedor JSSE use a criptografia do TLS.

Aplicativos cliente IBM MQ classes for Java que usam TRANSPORT(CLIENT) suportam a criptografia do TLS. O TLS fornece criptografia de comunicação, autenticação e integridade da mensagem. É geralmente usada para comunicações seguras entre qualquer dois pontos na Internet ou em uma intranet.

O IBM MQ classes for Java usa Java Secure Socket Extension (JSSE) para manipular a criptografia do TLS e, portanto, requer um provedor JSSE. JVMs JSE v1.4 têm um provedor JSSE integrado. Detalhes sobre como gerenciar e armazenar certificados podem variar de provedor para provedor. Para obter informações sobre isso, consulte a documentação do seu provedor JSSE.

Esta seção supõe que seu provedor JSSE esteja instalado e configurado corretamente e que os certificados adequados tenham sido instalados e disponibilizados para seu provedor JSSE.

Se o aplicativo cliente IBM MQ classes for Java usar uma tabela de definição de canal do cliente (CCDT) para se conectar a um gerenciador de filas, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for Java” na página 383.

Ativando o TLS no IBM MQ classes for Java

Para ativar o TLS, você especifica um CipherSuite. Existem duas maneiras de especificar um CipherSuite.

O TLS é suportado somente para conexões do cliente. Para ativar o TLS, deve-se especificar o CipherSuite a ser usado ao se comunicar com o gerenciador de filas e esse CipherSuite deve corresponder ao CipherSpec configurado no canal de destino. Além disso, o CipherSuite nomeado deve ser suportado pelo provedor JSSE. No entanto, CipherSuites são diferentes dos CipherSpecs e, portanto, têm nomes diferentes. O “CipherSpecs e CipherSuites TLS no IBM MQ classes for Java” na página 416 contém uma tabela mapeando os CipherSpecs suportados pelo IBM MQ para seus CipherSuites equivalentes, conforme conhecidos pelo JSSE.

Para ativar o TLS, especifique o CipherSuite usando a variável de membro estático sslCipherSuite de MQEnvironment. O exemplo a seguir conecta-se a um canal SVRCONN denominado SECURE.SVRCONN.CHANNEL, que foi configurado para requerer TLS com um CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA256:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Embora o canal tenha um CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA256, o aplicativo Java deve especificar um CipherSuite de SSL_RSA_WITH_AES_128_CBC_SHA256. Consulte “CipherSpecs e CipherSuites TLS no IBM MQ classes for Java” na página 416 para obter uma lista de mapeamentos entre CipherSpecs e CipherSuites.

Um aplicativo também pode especificar um CipherSuite configurando a propriedade do ambiente CMQC.SSL_CIPHER_SUITE_PROPERTY.

Como alternativa, use o Client Channel Definition Table (CCDT). Para obter mais informações, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for Java” na página 383

Se você requer uma conexão de cliente para usar um CipherSuite que é suportado pelo provedor JSSE FIPS IBM Java (IBMJSSEFIPS), um aplicativo pode configurar o campo sslFipsRequired na classe MQEnvironment como `true`. Como alternativa, o aplicativo pode configurar a propriedade do ambiente CMQC.SSL_FIPS_REQUIRED_PROPERTY. O valor padrão é `false`, que significa que uma conexão do cliente pode usar qualquer CipherSuite que seja suportado pelo IBM MQ.

Se um aplicativo usar mais de uma conexão do cliente, o valor do campo sslFipsRequired que é usado quando o aplicativo cria a primeira conexão do cliente determinará o valor que é usado quando o

aplicativo cria qualquer conexão do cliente subsequente. Portanto, quando o aplicativo cria uma conexão de cliente subsequente, o valor do campo sslFipsRequired é ignorado. Deve-se reiniciar o aplicativo se desejar usar um valor diferente para o campo sslFipsRequired.

Para conectar-se com êxito usando TLS, o armazenamento confiável JSSE deve ser configurado com certificados raiz de autoridade de certificação a partir dos quais o certificado apresentado pelo gerenciador de filas pode ser autenticado. Da mesma forma, se SSLClientAuth no canal SVRCONN tiver sido configurado como MQSSL_CLIENT_AUTH_REQUIRED, o keystore JSSE deverá conter um certificado de identificação que seja confiável pelo gerenciador de filas.

Referências relacionadas

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Usando o nome distinto do gerenciador de filas no IBM MQ classes for Java

O gerenciador de filas se identifica usando um certificado TLS, que contém um nome distinto (DN). Um aplicativo cliente do IBM MQ classes for Java pode usar este DN para assegurar que ele esteja se comunicando com o gerenciador de filas correto.

Um padrão de DN é especificado usando a variável sslPeerName de MQEnvironment. Por exemplo, configurar:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

permite que a conexão seja bem-sucedida somente se o gerenciador de filas apresentar um certificado com um Nome Comum iniciando QMGR., e pelo menos dois nomes de Unidade Organizacional, o primeiro dos quais deve ser IBM , e o segundo WebSphere.

Se sslPeerName estiver configurado, as conexões serão bem-sucedidas somente se ele estiver configurado com um padrão válido e o gerenciador de filas apresentar um certificado correspondente.

Um aplicativo também pode especificar o nome distinto do gerenciador de filas, configurando a propriedade de ambiente CMQC.SSL_PEER_NAME_PROPERTY. Para obter mais informações sobre nomes distintos, consulte Nomes distintos.

Usando listas de revogação de certificado no IBM MQ classes for Java

Especifique as listas de revogação de certificado a serem usadas pela classe java.security.cert.CertStore. O IBM MQ classes for Java verifica, então, certificados com relação à CRL especificada.

Uma lista de revogação de certificados (CRL) é um conjunto de certificados que foram revogados, pela autoridade de certificação emissora ou pela organização local. CRLs geralmente são hospedadas em servidores LDAP. Com o Java 2 v1.4, um servidor de CRL pode ser especificado no momento de conexão e o certificado apresentado pelo gerenciador de filas é verificado com relação à CRL antes que a conexão seja permitida. Para obter mais informações sobre listas de revogação de certificado e o IBM MQ, consulte Trabalhando com Listas de revogação de certificados e Listas de revogação de autoridades e Acessando as CRLs e ARLs com o IBM MQ classes for Java e IBM MQ classes for JMS.

Nota: Para usar um CertStore com sucesso com uma CRL hospedada em um servidor LDAP, certifique-se de que o Java Software Development Kit (SDK) seja compatível com a CRL. Alguns SDKs requerem que a CRL esteja em conformidade com o RFC 2587, que define um esquema para o LDAP v2. A maioria dos servidores LDAP v3 usa RFC 2256 em vez disso.

As CRLs a serem usadas são especificadas por meio da classe java.security.cert.CertStore. Consulte a documentação sobre essa classe para obter detalhes completos sobre como obter as instâncias de CertStore. Para criar um CertStore com base em um servidor LDAP, primeiro, crie uma instância LDAPCertStoreParameters inicializada com as configurações do servidor e da porta a serem usadas. Por exemplo:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Tendo criado uma instância CertStoreParameters, use o construtor estático em CertStore para criar um CertStore do tipo LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Outros tipos de CertStore (por exemplo, Collection) também são suportados. Normalmente, há vários servidores CRL configurados com informações idênticas da CRL para fornecer redundância. Quando houver um objeto CertStore para cada um desses servidores CRL, coloque-os todos em um Collection apropriado. O exemplo a seguir mostra os objetos CertStore colocados em uma ArrayList:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

Esse Collection pode ser configurado para a variável estática MQEnvironment, sslCertStores, antes de se conectar para ativar a verificação de CRL:

```
MQEnvironment.sslCertStores = crls;
```

O certificado apresentado pelo gerenciador de filas quando uma conexão que está sendo configurada for validada da seguinte forma:

1. O primeiro objeto CertStore no Collection identificado por sslCertStores é usado para identificar um servidor de CRL.
2. Uma tentativa é feita para contatar o servidor de CRL.
3. Se a tentativa for bem-sucedida, uma correspondência do certificado é procurada no servidor.
   a. Se o certificado tiver sido revogado, o processo de procura terminou e a solicitação de conexão falhará com o código de razão MQRC_SSL_CERTIFICATE_REVOKED.
   b. Se o certificado não for localizado, o processo de procura termina e a conexão tem permissão para continuar.
4. Se a tentativa de contatar o servidor não for bem-sucedida, o próximo objeto CertStore será usado para identificar um servidor CRL e o processo é repetido a partir da etapa 2.

   Se esse era o último CertStore no Collection ou se o Collection não contiver nenhum objeto CertStore, o processo de procura falhou e a solicitação de conexão falhará com o código de razão MQRC_SSL_CERT_STORE_ERROR.

O objeto Collection determina a ordem na qual CertStores são usados.

Collection of CertStores também pode ser configurado usando CMQC.SSL_CERT_STORE_PROPERTY. Como conveniência, essa propriedade também permite que um único CertStore seja especificado sem ser um membro de um Collection.

Se sslCertStores for configurado como null, nenhuma verificação de CRL será executada. Essa propriedade será ignorada se sslCipherSuite não estiver configurado.

Renegociando a chave secreta no IBM MQ classes for Java
Um aplicativo cliente IBM MQ classes for Java pode controlar quando a chave secreta usada para criptografia em uma conexão do cliente será renegociada em termos de número total de bytes enviados e recebidos.

O aplicativo pode fazer isso de uma das maneiras a seguir: se o aplicativo usar mais de uma dessas maneiras, as regras de precedência usuais se aplicam.

- Configurando o campo sslResetCount na classe MQEnvironment.
- Configurando uma propriedade de ambiente MQC.SSL_RESET_COUNT_PROPERTY em um objeto Hashtable. O aplicativo designa, então, a hashtable para o campo `properties` na classe MQEnvironment ou passa a hashtable para um objeto MQQueueManager em seu construtor.

O valor do campo sslResetCount ou a propriedade de ambiente MQC.SSL_RESET_COUNT_PROPERTY representa o número total de bytes enviados e recebidos pelo código do cliente IBM MQ classes for Java antes que a chave secreta seja renegociada. O número de bytes enviados é o número antes da criptografia e o número de bytes recebidos é o número após a decriptografia. O número de bytes também inclui informações de controle enviadas e recebidas pelo cliente IBM MQ classes for Java.

Se a contagem de reconfiguração for zero, que é o valor padrão, a chave secreta nunca será renegociada. A contagem de reconfiguração será ignorada se nenhum CipherSuite for especificado.

Fornecendo um SSLSocketFactory customizado em IBM MQ classes for Java
Se você usar um JSSE Socket Factory customizado, configure o MQEnvironment.sslSocketFactory para o objeto de factory customizado. Detalhes variam entre diferentes implementações de JSSE.

Diferentes implementações de JSSE podem fornecer diferentes recursos. Por exemplo, uma implementação de JSSE especializada pode permitir a configuração de um modelo específico de hardware de criptografia. Além disso, alguns provedores JSSE permitem a customização de keystores e de armazenamentos confiáveis por programa ou permitem que a opção de certificado de identidade do keystore seja alterada. No JSSE, todas essas customizações são abstraídas em uma classe de factory, javax.net.ssl.SSLSocketFactory.

Consulte a documentação de seu JSSE para obter detalhes sobre como criar uma implementação de SSLSocketFactory customizada. Os detalhes variam de provedor para provedor, mas uma sequência típica de etapas pode ser:

1. Criar um objeto SSLContext usando um método estático em SSLContext
2. Inicializar esse SSLContext com implementações de KeyManager e TrustManager apropriadas (criadas a partir de suas próprias classes de factory)
3. Criar um SSLSocketFactory a partir do SSLContext

Quando tiver um objeto SSLSocketFactory, configure o MQEnvironment.sslSocketFactory para o objeto de factory customizado. Por exemplo:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

O IBM MQ classes for Java usa esse SSLSocketFactory para se conectar ao gerenciador de filas do IBM MQ. Essa propriedade também pode ser configurada usando CMQC.SSL_SOCKET_FACTORY_PROPERTY. Se sslSocketFactory estiver configurado como null, o SSLSocketFactory padrão da JVM será usado. Essa propriedade será ignorada se sslCipherSuite não estiver configurado.

Ao usar SSLSocketFactories customizados, considere o efeito de compartilhamento de conexão TCP/IP. Se o compartilhamento de conexão for possível, então, um novo soquete não será solicitado do SSLSocketFactory fornecido, mesmo se o soquete produzido fosse diferente de alguma maneira no contexto de uma solicitação de conexão subsequentes. Por exemplo, se um certificado de cliente diferente precisar ser apresentado em uma conexão subsequente, então, o compartilhamento de conexões não deverá ser permitido.

Fazendo mudanças para o keystore ou armazenamento confiável do JSSE em IBM MQ classes for Java
Se você mudar o keystore ou armazenamento confiável JSSE, deverá executar determinadas ações para que as mudanças entrem em vigor.

Se você mudar o conteúdo do keystore ou do armazenamento confiável JSSE ou o local do arquivo de keystore ou de armazenamento confiável, os aplicativos IBM MQ classes for Java que estão em execução no momento não selecionarão automaticamente as mudanças. Para que as mudanças entrem em vigor, as seguintes ações devem ser executadas:

- Os aplicativos devem fechar todas as suas conexões e destruir quaisquer conexões não usadas em conjuntos de conexões.
- Se seu provedor JSSE armazenar em cache informações do keystore e do armazenamento confiável, essas informações deverão ser atualizadas.

Após essas ações terem sido executadas, os aplicativos poderão, então, recriar suas conexões.

Dependendo de como você projeta seus aplicativos e sobre a função fornecida pelo seu provedor JSSE, pode ser possível executar estas ações sem parar e reiniciar seus aplicativos. No entanto, parar e reiniciar o aplicativo poderá ser a solução mais simples.

Manipulação de erro ao usar TLS com o IBM MQ classes for Java
Vários códigos de razão podem ser emitidos pelo IBM MQ classes for Java ao conectar-se a um gerenciador de filas usando TLS.

Eles são explicados na lista a seguir:

MQRC_SSL_NOT_ALLOWED
A propriedade sslCipherSuite foi configurada, mas a conexão de ligações foi usada. Somente a conexão do cliente suporta TLS.

MQRC_JSSE_ERROR
O provedor JSSE relatou um erro que não pôde ser manipulado pelo IBM MQ. Isso pode ter sido causado por um problema de configuração com o JSSE ou porque o certificado apresentado pelo gerenciador de filas não pôde ser validado. A exceção produzida pelo JSSE pode ser recuperada usando o método getCause() em MQException.

MQRC_SSL_INITIALIZATION_ERROR
Uma chamada MQCONN ou MQCONNX foi emitida com opções de configuração de TLS especificadas, mas ocorreu um erro durante a inicialização do ambiente TLS.

MQRC_SSL_PEER_NAME_MISMATCH
O padrão de DN especificado na propriedade sslPeerName não correspondia ao DN apresentado pelo gerenciador de filas.

MQRC_SSL_PEER_NAME_ERROR
O padrão de DN especificado na propriedade sslPeerName não era válido.

MQRC_UNSUPPORTED_CIPHER_SUITE
O CipherSuite denominado no sslCipherSuite não foi reconhecido pelo provedor JSSE. Uma lista completa de CipherSuites suportados pelo provedor JSSE pode ser obtida por um programa, usando o método SSLSocketFactory.getSupportedCipherSuites(). Uma lista de CipherSuites que podem ser usados para se comunicar com o IBM MQ pode ser localizada em "CipherSpecs e CipherSuites TLS no IBM MQ classes for Java" na página 416.

MQRC_SSL_CERTIFICATE_REVOKED
O certificado apresentado pelo gerenciador de filas foi localizado em uma CRL especificada com a propriedade sslCertStores. Atualize o gerenciador de filas para usar certificados confiáveis.

MQRC_SSL_CERT_STORE_ERROR
Nenhum dos CertStores fornecidos pôde ser procurado pelo certificado apresentado pelo gerenciador de filas. O método MQException.getCause() retorna o erro que ocorreu ao faze a primeira tentativa de procura de CertStore. Se a exceção causal for NoSuchElementException, ClassCastException ou NullPointerException, verifique se o Collection especificado na propriedade sslCertStores contém pelo menos um objeto CertStore válido.

CipherSpecs e CipherSuites TLS no IBM MQ classes for Java
A capacidade de aplicativos IBM MQ classes for Java para estabelecer conexões com um gerenciador de filas, depende do CipherSpec especificado na extremidade do servidor do canal MQI e o CipherSuite especificado na extremidade do cliente.

A tabela a seguir lista os CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes.

Deprecated É necessário revisar o tópico CipherSpecs descontinuados para ver se algum dos CipherSpecs, listados na tabela a seguir, foi descontinuado pelo IBM MQ e, se sim, em qual atualização o CipherSpec foi descontinuado.

Importante: Os CipherSuites listados são aqueles suportados pelo IBM Java Runtime Environment (JRE) fornecido com IBM MQ. Os CipherSuites listados incluem aqueles suportados pelo Oracle Java JRE. Para

obter mais informações sobre como configurar seu aplicativo para usar um Oracle Java JRE, consulte Configurando seu aplicativo para usar IBM Java ou Oracle Java CipherSuite mapeamentos.

A tabela também indica o protocolo usado para a comunicação e se o CipherSuite está em conformidade com o padrão FIPS 140-2 ou não.

Nota: No AIX, Linux, and Windows, IBM MQ fornece conformidade FIPS 140-2 por meio do módulo criptográfico IBM Crypto for C (ICC) . O certificado deste módulo foi movido para o status Histórico. Os clientes devem visualizar o IBM Crypto for C (ICC) certificado e estar ciente de qualquer aviso fornecido pelo NIST Um módulo FIPS 140-3 de substituição está atualmente em andamento e seu status pode ser visualizado procurando por ele na NIST CMVP modules in process list.

O IBM MQ Operator 3.2.0 e a imagem do contêiner do gerenciador de filas 9.4.0.0 em diante são baseados no UBI 9 A conformidade do FIPS 140-3 está pendente atualmente e seu status pode ser visualizado procurando "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" no NIST CMVP modules in process list.

Ciphersuites denotados como compatíveis com FIPS 140-2 podem ser usados se o aplicativo não tiver sido configurado para impor a conformidade com FIPS 140-2, mas se a conformidade com FIPS 140-2 tiver sido configurada para o aplicativo (consulte as seguintes notas sobre configuração), apenas aqueles que estiverem marcados como compatíveis com FIPS 140-2 CipherSuites poderão ser configurados; tentar usar outro CipherSuites resulta em erro.

Nota: Cada JRE pode ter vários provedores de segurança criptográficos, com cada um podendo contribuir com uma implementação do mesmo CipherSuite. No entanto, nem todos os provedores de segurança são certificados pelo FIPS 140-2. Se a conformidade com FIPS 140-2 não é imposta para um aplicativo, então, é possível que uma implementação não certificada do CipherSuite possa ser usada. As implementações podem não funcionar em conformidade com FIPS 140-2, mesmo que o CipherSuite teoricamente atenda ao nível de segurança mínimo exigido pelo padrão. Consulte as notas a seguir para obter mais informações sobre a configuração do cumprimento FIPS 140-2 nos aplicativos IBM MQ Java.

Para obter mais informações sobre conformidade de FIPS 140-2 e Conjunto B para CipherSpecs e CipherSuites, veja Especificando CipherSpecs. Também pode ser necessário conhecer as informações referentes às Normas Federais de Processamento de Informações dos EUA.

Para usar o conjunto completo de CipherSuites e para operar com FIPS 140-2 e/ou conformidade Suite-B, um JRE adequado é necessário.IBM Java 7 Service Refresh 4 Fix Pack 2 ou um nível superior de IBM JRE fornece o suporte apropriado para o TLS 1.2 CipherSuites listado em Tabela 59 na página 418.

Para poder usar Cifras TLS 1.3 , o JRE em execução em seu aplicativo deve suportar TLS 1.3.

Nota: Para usar alguns CipherSuites, o 'irrestrito' política de arquivos precisam ser configurados no JRE. Para obter mais detalhes sobre como os arquivos de políticas são configurados em um SDK ou JRE, consulte o tópico *Arquivos de políticas do IBM SDK* na *Referência de segurança para o IBM SDK, Java Technology Edition* para a versão que você está usando.

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	no
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sim
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no
TLS_RSA_WITH_3DES_EDE_CBC_SHA "2" na página 437	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	não "4" na página 437

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	não "4" na página 437
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	não "4" na página 437

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	não "4" na página 437
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	não "4" na página 437

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	não "4" na página 437
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	não "4" na página 437

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	no
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	no
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	sim
TLS_AES_128_GCM_SHA256 "3" na página 437	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_AES_256_GCM_SHA384 "3" na página 437	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	no
TLS_CHACHA20_POLY1305_SHA256 "3" na página 437	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_AES_128_CCM_SHA256 "3" na página 437	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	no
TLS_AES_128_CCM_8_SHA256 "3" na página 437	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	no
QUALQUER "3" na página 437	*ANY	*ANY	Múltiplos	no
ANY_TLS13 "3" na página 437	*TLS13	*TLS13	TLS V13	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)				
CipherSpec **"1" na página 437**	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ANY_TLS12_OR_HIGHER "3" na página 437	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 e superior	no
ANY_TLS13_OR_HIGHER "3" na página 437	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 e acima	no

Notas:

1. Este é o valor configurado em um canal no IBM MQ, incluindo em um CCDT (binário ou JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.
3. Para poder usar Cifras TLS v1.3 , o Java runtime environment (JRE) que está executando seu aplicativo deve suportar TLS v1.3.
4. V 9.4.0 V 9.4.0 No IBM MQ 9.4.0, o JRE do IBM Java 8 remove o suporte para a troca de chave RSA ao operar no modo FIPS.

Configurando Ciphersuites e a conformidade com FIPS em um aplicativo IBM MQ classes for Java

- Um aplicativo que usa IBM MQ classes for Java pode usar um dos dois métodos para configurar o CipherSuite para uma conexão:

  - Configure o campo sslCipherSuite na classe MQEnvironment para o nome do CipherSuite.
  - Configure o CMQC.SSL_CIPHER_SUITE_PROPERTY da propriedade na hashtable de propriedades transmitido para o construtor MQQueueManager para o nome CipherSuite.
- Um aplicativo que usa o IBM MQ classes for Java pode usar um dos dois métodos para cumprir a conformidade com FIPS 140-2:
  - Configure o campo sslFipsRequired para true na classe MQEnvironment.
  - Configure a propriedade CMQC.SSL_FIPS_REQUIRED_PROPERTY na hashtable de propriedades passada para o construtor do MQQueueManager para true.

Configurando seu aplicativo para usar mapeamentos do IBM Java ou Oracle Java CipherSuite

V 9.4.0 No IBM MQ 9.4.0, um Cipher pode ser definido como o nome CipherSpec ou CipherSuite e é manipulado corretamente pelo IBM MQ.

Nota: Removed A Java Propriedade do sistema `com.ibm.mq.cfg.useIBMCipherMappings`, que controlava quais mapeamentos eram usados em versões anteriores de IBM MQ, não é mais necessária, e é removida do produto em IBM MQ 9.4.0

limitações de interoperabilidade

Determinados CipherSuites podem ser compatíveis com mais de um CipherSpec IBM MQ, dependendo do protocolo em uso. No entanto, apenas a combinação CipherSuite/CipherSpec que usa a versão de TLS especificada na Tabela 1 é suportada. Tentando usar as combinações não suportadas de CipherSuites e CipherSpecs falhará com uma exceção apropriada. Instalações usando qualquer uma dessas combinações CipherSuite/CipherSpec deve mover para uma combinação suportada.

A tabela a seguir mostra os CipherSuites para o qual essa limitação aplica-se.

Tabela 60. CipherSuites e suas CipherSpecs suportados e não suportados

CipherSuite	Suportado TLS CipherSpec	CipherSpec SSL não suportado
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na página 438	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Nota:

1. Deprecated Esse CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.

Executando aplicativos IBM MQ classes for Java

Se você gravar um aplicativo (uma classe que contenha um método main()), usando o modo de ligações ou de cliente, execute o programa usando o interpretador Java.

Use o comando:

```
java -Djava.library.path= library_path MyClass
```

em que *library_path* é o caminho para as bibliotecas do IBM MQ classes for Java. Para obter mais informações, consulte "IBM MQ classes for Java bibliotecas" na página 364.

Tarefas relacionadas

Rastreando aplicativos do IBM MQ classes for Java

Rastreando o Adaptador de recursos IBM MQ

Comportamento dependente do ambiente do IBM MQ classes for Java

O IBM MQ classes for Java permite criar aplicativos que podem ser executados com relação a versões diferentes do IBM MQ. Esta coleção de tópicos descreve o comportamento de classes Java dependentes dessas versões diferentes.

O IBM MQ classes for Java fornece um núcleo de classes, que fornecem função e comportamento consistente em todos os ambientes. Os recursos fora desse núcleo dependem do recurso do gerenciador de filas ao qual o aplicativo está conectado.

Exceto quando indicado aqui, o comportamento exibido é conforme descrito na Referência de aplicativos MQI apropriada para o gerenciador de filas.

Classes principais em IBM MQ classes for Java

IBM MQ classes for Java contém um conjunto principal de classes, que pode ser usado em todos os ambientes.

O seguinte conjunto de classes é considerado de classes principais e pode ser usado em todos os ambientes com apenas as variações menores listadas em “Restrições e variações para as classes principais do IBM MQ classes for Java” na página 440.

- MQEnvironment
- MQException
- MQGetMessageOptions

  Excluindo:

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentação
- MQManagedObject

  Excluindo:

  - inquire()
  - set()
- MQMessage

  Excluindo:

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

  Excluindo:

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

  Excluindo:
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQTopic
- MQC

Nota:

1. Algumas constantes não são incluídas no núcleo (consulte "Restrições e variações para as classes principais do IBM MQ classes for Java" na página 440 para obter detalhes); não as use em programas completamente portáteis.
2. Algumas plataformas não suportam todos os modos de conexão. Nessas plataformas, é possível usar apenas as classes principais e as opções relacionadas aos modos suportados.

Restrições e variações para as classes principais do IBM MQ classes for Java

As classes principais geralmente se comportam de forma consistente em todos os ambientes, mesmo se as chamadas MQI equivalentes normalmente tiverem diferenças de ambiente. O comportamento é como se um gerenciador de filas AIX, Linux ou Windows for usado, exceto para as pequenas restrições e variações a seguir.

Restrições para valores MQGMO_* no IBM MQ classes for Java
Determinados valores MQGMO_* não são suportados por todos os gerenciadores de fila.

O uso dos valores MQGMO_* a seguir pode resultar em uma MQException sendo emitida a partir de um MQQueue.get():

MQGMO_SYNCPOINT_IF _PERSISTENT
MQGMO_MARK_SKIP _BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_ AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

Além disso, MQGMO_SET_SIGNAL não é suportado quando usado a partir do Java.

Restrições para valores MQPMRF_* em IBM MQ classes for Java
Elas são usadas apenas ao colocar mensagens em uma lista de distribuição e são suportadas apenas por gerenciadores de filas que suportam listas de distribuição. Por exemplo, gerenciadores de filas do z/OS não suportam listas de distribuição.

Restrições para valores de MQPMO_* no IBM MQ classes for Java
Determinados valores de MQPMO_* não são suportados por todos os gerenciadores de filas

O uso dos valores de MQPMO_* a seguir pode resultar em uma MQException sendo emitida a partir de um MQQueue.put() ou um MQQueueManager.put():

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

Restrições e variações para valores de MQCNO_* no IBM MQ classes for Java
Determinados valores de `MQCNO_*` não são suportados.

- A reconexão automática do cliente não é suportada pelo IBM MQ classes for Java Qualquer valor de `MQCNO_RECONNECT_*` que você configurar, a conexão continua a se comportar como se `MQCNO_RECONNECT_DISABLED` estivesse configurado.
- `MQCNO_FASTPATH` é ignorado em gerenciadores de filas que não suportam `MQCNO_FASTPATH`. Ele também é ignorado pelas conexões do cliente.

Restrições para valores MQRO_* no IBM MQ classes for Java
As seguintes opções de relatório podem ser configuradas.

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

Para obter mais informações, consulte Relatório.

Miscellaneous differences between IBM MQ classes for Java on z/OS and other platforms
IBM MQ for z/OS behaves differently from IBM MQ on other platforms in some areas.

BackoutCount
A z/OS queue manager returns a maximum BackoutCount of 255, even if the message has been backed out more than 255 times.

Default dynamic queue prefix
When connected to a z/OS queue manager using a bindings connection, the default dynamic queue prefix is CSQ.*. Otherwise, the default dynamic queue prefix is AMQ.*.

MQQueueManager constructor
Client connect is not supported on z/OS. Attempting to connect with client options results in an MQException with MQCC_FAILED and MQRC_ENVIRONMENT_ERROR.

The MQQueueManager constructor might also fail with MQRC_CHAR_CONVERSION_ERROR (if it fails to initialize conversion between the IBM-1047 and ISO8859-1 code pages), or MQRC_UCS2_CONVERSION_ERROR (if it fails to initialize conversion between the queue manager's code page and Unicode). If your application fails with one of these reason codes, ensure that the National Language Resources component of Language Environment is installed, and ensure that the correct conversion tables are available.

Conversion tables for Unicode are installed as part of the z/OS C/C++ optional feature. See the *z/OS C/C++ Programming Guide*, SC09-4765, for more information about enabling UCS-2 conversions.

Recursos fora das classes principais do IBM MQ classes for Java

O IBM MQ classes for Java contêm determinadas funções que são especificamente projetadas para usar extensões de API não suportadas por todos os gerenciadores de filas. Esta coleção de tópicos descreve como eles se comportam ao usar um gerenciador de filas que não os suporta.

Variações na opção de construtor MQQueueManager

Alguns dos construtores MQQueueManager incluem um argumento de número inteiro opcional. Alguns valores desse argumento não são aceitos em todas as plataformas.

Quando um construtor MQQueueManager inclui um argumento de número inteiro opcional, ele é mapeado para o campo de opções MQCNO da MQI e é usado para alternar entre conexão normal e de atalho. Este formato estendido do construtor é aceito em todos os ambientes, se as únicas opções usadas forem MQCNO_STANDARD_BINDING ou MQCNO_FASTPATH_BINDING. Quaisquer outras opções fazem o construtor falhar com MQRC_OPTIONS_ERROR. A opção de atalho CMQC.MQCNO_FASTPATH_BINDING é honrada somente com uma conexão de ligações para um gerenciador de filas que a suporte. Em outros ambientes, ela será ignorada.

Restrições no método MQQueueManager.begin()

Esse método pode ser usado apenas com relação a um gerenciador de filas do IBM MQ em sistemas AIX, Linux, and Windows no modo de ligação Caso contrário, ele falha com MQRC_ENVIRONMENT_ERROR.

Consulte “Coordenação JTA/JDBC usando o IBM MQ classes for Java” na página 408 para obter mais detalhes.

Variações nos campos MQGetMessageOptions

Alguns gerenciadores de filas não suportam a estrutura MQGMO Versão 2, portanto, deve-se configurar alguns campos para seus valores padrão.

Ao usar um gerenciador de filas que não suporta a estrutura MQGMO Versão 2, deixe os campos a seguir definidos para seus valores padrão:

GroupStatus
SegmentStatus
Segmentação

Além disso, o campo MatchOptions suporta apenas MQMO_MATCH_MSG_ID e MQMO_MATCH_CORREL_ID. Se você colocar valores não suportados nesses campos, o MQDestination.get() subsequente falhará com MQRC_GMO_ERROR. Se o gerenciador de filas não suportar a estrutura MQGMO Versão 2, esses campos não serão atualizados após um MQDestination.get() bem-sucedido.

Restrições em listas de distribuição em IBM MQ classes for Java

Nem todos os gerenciadores de fila permitem abrir uma MQDistributionList.

As classes a seguir são usadas para criar listas de distribuição:

MQDistributionList
MQDistributionListItem
MQMessageTracker

É possível criar e preencher MQDistributionLists e MQDistributionListItems em qualquer ambiente, mas nem todos os gerenciadores de filas permitem abrir uma MQDistributionList. Em particular, os gerenciadores de filas do z/OS não suportam listas de distribuição. Tentativa de abrir um MQDistributionList ao usar esse gerenciador de filas resulta em MQRC_OD_ERROR.

Variações nos campos MQPutMessageOptions

Se um gerenciador de filas não suportar listas de distribuição, determinados campos MQPMO serão tratados de forma diferente.

Quatro campos no MQPMO são renderizados como as variáveis de membro a seguir na classe MQPutMessageOptions:

knownDestCount
unknownDestCount
invalidDestCount
recordFields

Esses campos são destinados principalmente para uso com listas de distribuição. No entanto, um gerenciador de filas que suporta listas de distribuição também preenche os campos DestCount após um MQPUT para uma única fila. Por exemplo, se a fila for resolvida para uma fila local, knownDestCount será configurado para 1 e os outros dois campos de contagem serão configurados para 0.

Se o gerenciador de filas não suporta listas de distribuição, esses valores serão simuladas da seguinte forma:

- Se o put() for bem-sucedido, unknownDestCount será configurado para 1 e os outros serão configurados para 0.
- Se o put() falhar, invalidDestCount será configurado para 1 e os outros serão configurados para 0.

A variável recordFields é usada com listas de distribuição. Um valor pode ser gravado em recordFields a qualquer momento, independentemente do ambiente. Ele é ignorado se o objeto MQPutMessageOptions for usado em um MQDestination.put() ou MQQueueManager.put() subsequente, em vez de MQDistributionList.put().

Restrições em campos do MQMD com IBM MQ classes for Java
Alguns campos do MQMD com respeito a segmentação de mensagens devem ser deixados com seus valores padrão ao usar um gerenciador de filas que não suporta segmentação.

Os seguintes campos MQMD são amplamente envolvidos com segmentação de mensagens:

GroupId
MsgSeqNumber
Offset
MsgFlags
OriginalLength

Se um aplicativo configura qualquer um desses campos do MQMD para valores diferentes de seus padrões e, em seguida, executa um put() ou get() em um gerenciador de filas que não os suporta, put() ou get() gerarão uma MQException com MQRC_MD_ERROR. Um put() ou get() bem-sucedido com esse gerenciador de filas sempre deixa os campos do MQMD configurados para seus valores padrão. Não envie uma mensagem agrupada ou segmentada para um aplicativo do Java que é executado em um gerenciador de filas que não suporta agrupamento e segmentação de mensagens.

Se um aplicativo do Java tenta efetuar get() em uma mensagem a partir de um gerenciador de filas que não suporta esses campos, e a mensagem física a ser recuperada for parte de um grupo de mensagens segmentadas (ou seja, ele possui valores não padrão para os campos do MQMD), ele será recuperado sem erro. No entanto, os campos do MQMD no MQMessage não são atualizados, a propriedade de formato MQMessage é definida como MQFMT_MD_EXTENSION e os dados da mensagem verdadeira são prefixadas com uma estrutura MQMDE que contém os valores para os novos campos.

Restrições para o IBM MQ classes for Java sob CICS Transaction Server

No ambiente do CICS Transaction Server for z/OS, somente o encadeamento principal (primeiro) é permitido para emitir chamadas do CICS ou IBM MQ.

Observe que as classes do IBM MQ JMS não são suportadas para uso em um aplicativo CICS Java.

Portanto, não é possível compartilhar objetos MQQueueManager ou MQQueue entre encadeamentos nesse ambiente ou criar um novo MQQueueManager em um encadeamento filho.

z/OS "Miscellaneous differences between IBM MQ classes for Java on z/OS and other platforms" na página 441 identifica algumas restrições e variações que se aplicam ao IBM MQ classes for Java quando estiver executando com relação a um gerenciador de filas do z/OS. Além disso, quando estiver executando no CICS, os métodos de controle de transação em MQQueueManager não são suportados.

Em vez de emitir MQQueueManager.commit () ou MQQueueManager.backout (), os aplicativos usam os métodos de sincronização de tarefas JCICS, Task.commit () e Task.rollback (). A classe Tarefa é fornecida por JCICS no pacote com.ibm.cics.server.

Usando o adaptador de recursos do IBM MQ

O adaptador de recursos permite que aplicativos que estão em execução em um servidor de aplicativos acesse recursos do IBM MQ. Ele suporta a comunicação de entrada e de saída.

O que o adaptador de recursos contém

A partir do IBM MQ 9.3.0, o Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Além do adaptador de recursos que suporta Java EE e JMS 2.0, IBM MQ 9.3.0 e mais tarde fornecem um adaptador de recursos que suporta Jakarta Messaging

JM 3.0 **IBM MQ adaptador de recursos para Jakarta Messaging**

O Jakarta Connectors Architecture fornece uma maneira padrão de conectar aplicativos que estão em execução em um ambiente do Jakarta EE para um Enterprise Information System (EIS) como IBM MQ ou Db2. O adaptador de recursos IBM MQ para Jakarta Messaging implementa as interfaces Jakarta Connectors 2.0.0 e contém o IBM MQ classes for Jakarta Messaging. Ele permite que aplicativos Jakarta Messaging e beans acionados por mensagens (MDBs) em execução em um servidor de aplicativos, acessem os recursos de um gerenciador de filas do IBM MQ . O adaptador de recursos suporta tanto o domínio ponto a ponto e o domínio de publicar/assinar.

JMS 2.0 **IBM MQ adaptador de recursos para JMS 2.0**

O Java Platform, Enterprise Edition Connector Architecture (JCA) fornece uma maneira padrão de conectar aplicativos que estão em execução em um ambiente do Java EE a um Enterprise Information System (EIS), como IBM MQ ou Db2. O adaptador de recursos IBM MQ para JMS 2.0 implementa as interfaces JCA 1.7 e contém o IBM MQ classes for JMS. Ele permite que os aplicativos JMS e os beans acionados por mensagens (MDBs), em execução em um servidor de aplicativos, acessem os recursos de um gerenciador de filas do IBM MQ. O adaptador de recursos suporta tanto o domínio ponto a ponto e o domínio de publicar/assinar.

O adaptador de recursos do IBM MQ suporta dois tipos de comunicação entre um aplicativo e um gerenciador de filas:

Comunicação de saída

Um aplicativo inicia uma conexão com um gerenciador de filas e, em seguida, envia mensagens do JMS para destinos do JMS e recebe mensagens do JMS a partir de destinos do JMS de maneira síncrona.

Comunicação de entrada

Uma mensagem do JMS que chega em um destino do JMS é entregue a um MDB, que processa a mensagem de forma assíncrona.

O adaptador de recursos também contém o IBM MQ classes for Java. As classes estão automaticamente disponíveis para aplicativos que estão em execução em um servidor de aplicativos no qual o adaptador de recursos foi implementado e permitem que os aplicativos que estão em execução nesse servidor de aplicativos usam a API do IBM MQ classes for Java quando eles estão acessando recursos de um gerenciador de filas do IBM MQ.

O uso do IBM MQ classes for Java em um ambiente do Java EE é suportado com restrições. Para obter informações sobre essas restrições, consulte “Executando aplicativos IBM MQ classes for Java no Java EE” na página 355.

Qual versão do adaptador de recursos usar

A versão do adaptador de recursos que você usa depende se você está implementando-o em um servidor de aplicativos que suporta Jakarta EE ou Java EE:

JM 3.0 **Jakarta EE**

Em IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado. O adaptador de recursos IBM MQ para Jakarta Messaging deve ser implementado em um servidor de aplicativos que suporte Jakarta EE.

Java EE 7

O adaptador de recursos do IBM MQ 8.0 e mais recente suporta o JCA v1.7 e fornece suporte ao JMS 2.0. Esse adaptador de recursos precisa ser implementado dentro de um servidor de aplicativos Java EE 7 e posterior (consulte “Instrução de suporte do adaptador de recursos do IBM MQ” na página 446).

É possível instalar o adaptador de recursos da IBM MQ 8.0 ou mais recente em qualquer servidor de aplicativos que seja certificado como compatível com a especificação do Java Platform, Enterprise Edition 7. Usando o adaptador de recursos IBM MQ 8.0 ou posterior, um aplicativo pode se conectar a um gerenciador de filas usando o transporte BINDINGS ou CLIENT.

Importante: O adaptador de recursos do IBM MQ 8.0 ou mais recente pode ser implementado somente em um servidor de aplicativos que suporte o JMS 2.0.

Usando o adaptador de recursos com o WebSphere Application Server traditional

O adaptador de recursos IBM MQ é pré-instalado no WebSphere Application Server traditional 9.0 ou posterior. Portanto, não há nenhum requisito para instalar um novo adaptador de recursos.

JM 3.0 O WebSphere Application Server traditional não suporta atualmente Jakarta EE. Consulte a instrução de suporte do adaptador de recursos IBM MQ .

Nota: Um adaptador de recursos do IBM MQ 9.0 ou mais recente pode se conectar no modo de transporte CLIENT ou BINDINGS a qualquer gerenciador de filas do IBM MQ em serviço.

Usando o adaptador de recursos com o WebSphere Liberty

Para conectar-se ao IBM MQ a partir do WebSphere Liberty, deve-se usar o adaptador de recursos do IBM MQ. Como o Liberty não contém o adaptador de recursos do IBM MQ, deve-se obtê-lo separadamente da Fix Central.

A versão do adaptador de recursos que você usa depende se você está implementando em uma versão do Liberty que suporta Jakarta EE ou Java EE.

Para obter mais informações sobre como fazer download e instalar o adaptador de recursos, veja “Instalando o adaptador de recursos no Liberty” na página 453.

Conceitos relacionados

“Configurando o adaptador de recursos para comunicação de entrada” na página 461
Para configurar a comunicação de entrada, defina as propriedades de um ou mais objetos ActivationSpec.

“Configurando o adaptador de recursos para comunicação de saída” na página 480
Para configurar a comunicação de saída, defina as propriedades de um objeto ConnectionFactory e um objeto de destino administrado.

“Usando IBM MQ classes for JMS/Jakarta Messaging” na página 83
IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os Java provedores de sistemas de mensagens fornecidos com IBM MQ. Assim como a implementação das interfaces definidas nas especificações JMS e Jakarta Messaging , esses provedores de sistemas de mensagens incluem dois conjuntos de extensões na API do sistema de mensagens do Java

“Usando o IBM MQ classes for Java” na página 353
Use o IBM MQ em um ambiente Java. O IBM MQ classes for Java permite que um aplicativo Java se conecte ao IBM MQ como um cliente IBM MQ ou se conecte diretamente a um gerenciador de filas do IBM MQ.

Referências relacionadas

Configurando o servidor de aplicativos para usar o nível de manutenção mais recente do adaptador de recursos

Determinação de problemas para o adaptador de recursos do IBM MQ

Tópicos doWebSphere Application Server

Mantendo o adaptador de recursos do IBM MQ

Implementando aplicativos JMS no Liberty para usar o provedor de sistemas de mensagens do IBM MQ

Instrução de suporte do adaptador de recursos do IBM MQ

O adaptador de recursos do IBM MQ que deve ser usado para comunicação entre um aplicativo e um gerenciador de filas depende se você está usando a API Jakarta Messaging 3.0 ou a API JMS 2.0 .

JMS 2.0 IBM MQ 8.0 ou posterior vem com um adaptador de recursos que implementa a especificação JMS 2.0 . Ele só pode ser implementado em um servidor de aplicativos que é compatível com o Java Platform, Enterprise Edition 7 (Java EE 7) e, portanto, suporta JMS 2.0. Uma lista de servidores de aplicativos certificados para Java Platform, Enterprise Edition é mantida no website do Oracle.

JM 3.0 A partir do IBM MQ 9.3.0, o Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Além do adaptador de recursos que suporta Java EE e JMS 2.0, IBM MQ 9.3.0 e mais tarde fornecem um adaptador de recursos que suporta Jakarta Messaging Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando as classes do IBM MQ para JMS.

Implementação em WebSphere Liberty

O WebSphere Liberty 8.5.5 Fix Pack 6 e mais recente e o WebSphere Application Server Liberty 9.0 e mais recente são servidores de aplicativos certificados Java EE 7, de modo que o adaptador de recursos do IBM MQ 9.0 pode ser implementado neles.

Para usar o adaptador de recursos IBM MQ para Jakarta Messaging com Liberty, deve-se usar uma versão do Liberty que suporte Jakarta EE.

O WebSphere Liberty tem os seguintes recursos disponíveis para trabalhar com adaptadores de recursos:

- JM 3.0 O recurso messaging-3.0 para permitir trabalhar com adaptadores de recursos do Jakarta Messaging 3.0 .
- O recurso wmqJmsClient-2.0 para permitir o trabalho com os adaptadores de recursos do JMS 2.0.
- O recurso wmqJmsClient-1.1 para permitir o trabalho com os adaptadores de recursos do JMS 1.1.

Importante:

- JM 3.0 O adaptador de recursos IBM MQ para Jakarta Messaging deve ser implementado em uma versão do Liberty que suporte Jakarta EE. Esse adaptador de recursos não pode ser usado com versões de Liberty que suportam a especificação Java EE mais antiga não Jakarta EE..
- JMS 2.0 O adaptador de recursos IBM MQ 8.0 ou posterior que suporta JMS 2.0 deve ser implementado com o recurso wmqJmsClient-2.0 .

Implementação em WebSphere Application Server traditional

O WebSphere Application Server traditional 9.0 é fornecido com um adaptador de recursos do IBM MQ 9.0 já instalado. Portanto, não há nenhum requisito para instalar um novo adaptador de recursos. O adaptador de recursos instalado pode se conectar no modo de transporte CLIENT ou BINDINGS a quaisquer gerenciadores de filas em execução em uma versão suportada do IBM MQ. Para obter mais informações, consulte “Conectividade com gerenciadores de filas do IBM MQ 8.0 ou mais recente” na página 447.

Importante:

- O adaptador de recursos do IBM MQ 9.0 não pode ser implementado em versões do WebSphere Application Server traditional antes do IBM MQ 9.0, pois essas versões não são certificadas pelo Java EE 7.
- JM 3.0 O WebSphere Application Server traditional não suporta atualmente Jakarta EE.

Para obter mais informações sobre as versões do adaptador de recursos que são fornecidas com o WebSphere Application Server, consulte a nota técnica Qual versão do WebSphere MQ Resource Adapter (RA) é fornecida com o WebSphere Application Server?

Usando o adaptador de recursos com outros servidores de aplicativos

Para todos os outros servidores de aplicativos compatíveis com Java EE 7 ou Jakarta EE , os problemas que ocorrem após a conclusão bem-sucedida do IBM MQ Resource Adapter Installation Verification Test (IVT) podem ser relatados ao IBM para a investigação do IBM MQ rastreio do produto e outras IBM MQ informações de diagnóstico. Se o IVT do adaptador de recursos do IBM MQ não puder ser executado com êxito, quaisquer problemas encontrados provavelmente serão causados por implementação incorreta ou definições de recursos incorretas que são específicas do servidor de aplicativos e os problemas devem ser investigados usando a documentação do servidor de aplicativos e a organização de suporte para esse servidor de aplicativos.

Java Tempo de execução

O Java Runtime (JRE) que é usado para executar o servidor de aplicativos deve ser um que seja suportado com o IBM MQ 9.0 ou Cliente mais recente. Para obter informações adicionais, consulte Requisitos do sistema para IBM MQ. (Selecione qual versão e sistema operacional ou relatório de componente você deseja ver e, em seguida, siga o link **Java** que é listado na guia **Software suportado**).

Conectividade com gerenciadores de filas do IBM MQ 8.0 ou mais recente

A gama completa de funcionalidades do JMS 2.0 está disponível ao se conectar a um gerenciador de filas do IBM MQ 8.0 ou mais recente usando o adaptador de recursos que foi implementado em um servidor de aplicativos certificado por Java EE 7. Para fazer uso da funcionalidade do JMS 2.0, o adaptador de recursos precisa se conectar ao gerenciador de filas usando o modo normal do provedor de sistemas de mensagens do IBM MQ. Para obter mais informações, veja Configurando a propriedade JMS**PROVIDERVERSION**.

JM 3.0 A gama completa de funcionalidades do Jakarta Messaging 3.0 está disponível ao se conectar a um gerenciador de filas do IBM MQ 9.3 ou mais recente usando o adaptador de recursos que foi implementado em um servidor de aplicativos certificado por Jakarta EE.

Extensões de MQ

A especificação do JMS 2.0 introduz mudanças em como certos comportamentos funcionam. Como o IBM MQ 8.0 ou mais recente implementa essa especificação, há mudanças no comportamento entre o IBM MQ 8.0 e versões mais recentes e anteriores do produto. No IBM MQ 8.0 ou mais recente, o IBM MQ classes for JMS inclui suporte para a propriedade do sistema Java `com.ibm.mq.jms.SupportMQExtensions` que, quando for configurado como TRUE, fará com que essas versões do IBM MQ revertam esses comportamentos para aqueles do IBM WebSphere MQ 7.5 ou versões anteriores. O valor padrão da propriedade é FALSE.

O adaptador de recursos do IBM MQ 9.0 ou mais recente também inclui uma propriedade do adaptador de recursos chamada `supportMQExtensions` que tem o mesmo efeito e valor padrão que a propriedade de sistema `com.ibm.mq.jms.SupportMQExtensions` Java. Essa propriedade do adaptador de recursos é configurada como falsa no `ra.xml` por padrão.

Se a propriedade do adaptador de recursos e a propriedade de sistema Java estão configuradas, a propriedade de sistema tem precedência.

Observe que, dentro do adaptador de recursos que já está implementado no WebSphere Application Server traditional 9.0, essa propriedade é automaticamente configurada para TRUE para a migração de auxílio.

Para obter informações adicionais, consulte "Propriedade SupportMQExtensions" na página 334.

Problemas Gerais

A intercalação de sessão não é suportada

Alguns servidores de aplicativos fornecem um recurso chamado intercalação de sessão, em que a mesma sessão do JMS pode ser usada em várias transações, embora seja inscrita somente uma vez. O adaptador de recursos do IBM MQ não suporta esse recurso, que pode levar aos problemas a seguir:

```
Uma tentativa de colocar uma mensagem em uma fila do MQ falha com o código de razão 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Chamadas para xa_close() falham com o código de razão -3 (XAER_PROTO) e um FDC com ID de análise AT040010 é gerado no gerenciador de filas do IBM MQ que está sendo acessado a partir do servidor de aplicativos. Para obter informações sobre como desativar esse recurso, consulte a documentação do servidor de aplicativos.

Especificação do Java Transaction API (JTA) de como os recursos XA são recuperados para a recuperação de transação XA

A seção 3.4.8 da especificação de JTA não define um mecanismo específico pelo qual os recursos XA são recriados para executar a recuperação transacional de XA. Assim, cabe a cada gerenciador de transações individual (e, portanto, o servidor de aplicativos) decidir como os recursos XA envolvidos em uma transação XA são recuperados. É possível que, para alguns servidores de aplicativos, o adaptador de recursos do IBM MQ 9.0 não implemente os mecanismos específicos do servidor de aplicativos que são usados para executar a recuperação transacional XA.

Correnpondendo conexões em um ManagedConnectionFactory

Um servidor de aplicativos pode chamar o método matchManagedConnections em uma instância ManagedConnectionFactory fornecida pelo adaptador de recursos do IBM MQ. Uma ManagedConnection será retornada apenas se o método localizar uma que corresponda aos argumentos **`javax.security.auth.Subject`** e **`javax.resource.spi.ConnectionRequestInfo`** que foram passados para o método pelo servidor de aplicativos.

Limitações do adaptador de recursos do IBM MQ

O adaptador de recursos do IBM MQ é suportado em todas as plataformas do IBM MQ. No entanto, quando você usa o adaptador de recursos do IBM MQ, alguns recursos do IBM MQ estão indisponíveis ou são limitados.

O adaptador de recursos do IBM MQ tem as limitações a seguir:

- Desde o IBM MQ 8.0, o adaptador de recursos é um adaptador de recursos Java Platform, Enterprise Edition 7 (Java EE 7) que fornece a função do JMS 2.0. Consequentemente, o adaptador de recursos do IBM MQ 8.0 ou mais recente deve ser instalado em um servidor de aplicativos Java EE 7 ou mais recente certificado. Ele pode se conectar no modo de transporte do cliente ou de ligações a qualquer gerenciador de filas de serviço.
- Ao executar dentro do servidor de aplicativos do WebSphere Liberty, o IBM MQ classes for Java estabilizado não é suportado. Dentro de outros servidores de aplicativos, o IBM MQ classes for Java não é recomendado para uso. Consulte a IBM nota técnica Usando Interfaces do WebSphere MQ Java em Ambientes J2EE/JEE para obter detalhes de IBM MQ classes for Java considerações em Java EE.
- Ao executar dentro do servidor de aplicativos do WebSphere Liberty no z/OS, o recurso wmqJmsClient-2.0 deve ser usado. O suporte genérico do JCA não é possível para o z/OS.
- O adaptador de recursos do IBM MQ não suporta programas de saída do canal escritos em linguagens diferentes de Java.

- Enquanto um servidor de aplicativos está em execução, o valor da propriedade sslFipsRequired deve ser true para todos os recursos do JCA ou false para todos os recursos do JCA. Este será um requisito mesmo se os recursos do JCA não forem usados simultaneamente. Se a propriedade sslFipsRequired tiver diferentes valores para diferentes recursos do JCA, o IBM MQ emitirá o código de razão MQRC_UNSUPPORTED_CIPHER_SUITE, mesmo se uma conexão do TLS não estiver sendo usada.
- Não é possível especificar mais de um keystore para um servidor de aplicativos. Se forem feitas conexões com mais de um gerenciador de filas, todas as conexões devem usar o mesmo keystore. Essa limitação não se aplica ao WebSphere Application Server.
- Se você usar uma tabela de definição de canal de cliente (CCDT) com mais de uma definição de canal de conexão de cliente apropriada, no caso de uma falha, o adaptador de recursos pode selecionar uma definição de canal diferente e, portanto, um gerenciador de filas diferente na CCDT, o que causaria problemas para a recuperação de transação. O adaptador de recursos não toma nenhuma ação para evitar que essa configuração seja usada e é sua responsabilidade evitar configurações que possam causar problemas para a recuperação da transação.
- A funcionalidade de nova tentativa de conexão não é suportada para conexões de saída ao executar em um contêiner Java EE (EJB/Servlet). A nova tentativa de conexão não é suportada para o JMS de saída quando o adaptador é usado em um contexto de contêiner do JEE , independentemente da configuração de transação ou para uso não transacionado.
- A reautenticação, conforme definido na Seção 9.1.9 da especificação Java EE Connector Architecture versão 1.7, de conexões JMS, não é suportada. O arquivo `ra.xml` dentro do adaptador de recursos do IBM MQ deve ter a propriedade chamada **`reauthentication-support`** configurada para o valor `false`. Uma tentativa feita pelo servidor de aplicativos para reautenticar uma conexão JMS faz com que o adaptador de recursos do IBM MQ lance uma exceção javax.resource.spi.SecurityException com o código de mensagem MQJCA1028.

Tarefas relacionadas

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Referências relacionadas

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

WebSphere Application Server e o adaptador de recursos do IBM MQ

O adaptador de recursos do IBM MQ é usado por aplicativos que executam o sistema de mensagens do JMS com o provedor do sistema de mensagens do IBM MQ no WebSphere Application Server.

Importante: Não use o adaptador de recursos do IBM MQ com WebSphere Application Server 6.0 ou WebSphere Application Server 6.1

WebSphere Application Server traditional 9.0 inclui uma versão do adaptador de recursos IBM MQ 9.0. O adaptador de recursos do IBM MQ 9.0 ou mais recente não pode ser implementado em versões anteriores do WebSphere Application Server, uma vez que essas versões não são certificadas pelo Java EE 7.

JM 3.0 O WebSphere Application Server traditional não suporta atualmente Jakarta EE. Consulte a instrução de suporte do adaptador de recursos IBM MQ .

Se quiser usar um aplicativo JMS para acessar os recursos de um gerenciador de filas do IBM MQ no WebSphere Application Server, use o provedor do sistema de mensagens do IBM MQ no WebSphere Application Server. O provedor de mensagens do IBM MQ contém uma versão do IBM MQ classes for JMS. Para obter mais informações, consulte a nota técnica Qual versão do WebSphere MQ Resource Adapter (RA) é fornecida com o WebSphere Application Server?.

Importante: Não inclua nenhum dos arquivos JAR IBM MQ classes for JMS ou IBM MQ classes for Java em seu aplicativo. Isso pode resultar em ClassCastExceptions e pode ser difícil de manter.

Liberty e o adaptador de recursos do IBM MQ

O adaptador de recursos IBM MQ pode ser instalado no WebSphere Liberty usando um recurso Liberty . O recurso usado depende de qual versão do adaptador de recursos você está instalando. Como alternativa, é possível, sujeito a algumas restrições, instalar o adaptador de recursos usando o suporte genérico do Java Platform, Enterprise Edition Connector Architecture (Java EE JCA).

Restrições gerais ao instalar o adaptador de recursos no Liberty

As restrições a seguir se aplicam ao adaptador de recursos ao usar o recurso wmqJmsClient-1.1 ou wmqJmsClient-2.0 e também ao usar o suporte genérico do JCA:

- O IBM MQ classes for Java não é suportado no Liberty. Ele não deve ser usado com o recurso de sistema de mensagens do IBM MQ Liberty ou com o suporte JCA genérico. Para obter mais informações, consulte Usando Interfaces do WebSphere MQ Java em J2EE/JEE .
- O adaptador de recursos do IBM MQ tem um tipo de transporte de BINDINGS_THEN_CLIENT. Esse tipo de transporte não é suportado no recurso de sistema de mensagens do IBM MQ Liberty.
- Antes do IBM MQ 9.0, o recurso Advanced Message Security (AMS) não estava incluído no recurso do sistema de mensagens do IBM MQ Liberty. No entanto, o AMS é suportado com um adaptador de recursos do IBM MQ 9.0 ou mais recente.

Nota: Em IBM MQ versões maiores que IBM MQ 9.0.0.6 e IBM MQ 9.1.0.1 você deve usar o recurso transportSecurity-1.0 em vez do recurso ssl-1.0 .

Para obter informações adicionais, consulte:

Ativando a comunicação SSL no Liberty
Padrões SSL em Liberty
Segurança de Transporte 1.0

Restrições ao usar os recursos do Liberty

Com o WebSphere Liberty 8.5.5 Fix Pack 2 para WebSphere Liberty 8.5.5 Fix Pack 5 inclusive, somente o recurso wmqJmsClient-1.1 estava disponível e somente o JMS 1.1 poderia ser usado. O WebSphere Liberty 8.5.5 Fix Pack 6 incluiu o recurso wmqJmsClient-2.0 para que o JMS 2.0 pudesse ser usado.

JM 3.0 Em IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado. Para usar o adaptador de recursos IBM MQ para Jakarta Messaging com Liberty, deve-se usar uma versão do Liberty que suporte Jakarta EE. Deve-se usar o adaptador de recursos para Jakarta Messaging com o recurso Liberty generic messaging-3.0 .

O recurso que deve ser usado depende de qual versão do adaptador de recursos você está usando:

- O IBM MQ 8.0.0 Fix Pack 3 e o adaptador de recursos do IBM MQ 8.0 mais recente podem ser usados somente com o recurso wmqJmsClient-2.0.
- O adaptador de recursos do IBM MQ 9.0 pode ser usado somente com o recurso wmqJmsClient-2.0.
- JM 3.0 O recurso messaging-3.0 permite trabalhar com adaptadores de recursos Jakarta Messaging 3.0 .

Restrições ao usar o suporte JCA genérico

Se você estiver usando o suporte JCA genérico, as restrições a seguir se aplicarão:

- Deve-se especificar o nível de JMS ao usar o suporte genérico do JCA O JMS 2.0 e JCA 1.7 devem ser usados somente com os adaptadores de recursos do IBM MQ 8.0.0 Fix Pack 3 e IBM MQ 8.0 mais recente.
- Não é possível executar o adaptador de recursos do IBM MQ no z/OS usando suporte genérico do JCA. Para executar o adaptador de recursos do IBM MQ no z/OS, ele deve ser executado com o recurso wmqJmsClient-1.1 ou wmqJmsClient-2.0.

- O local do adaptador de recursos é especificado usando o elemento xml a seguir:

```
JM 3.0 <resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

```
JMS 2.0 <resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Importante: O valor da tag ID pode ser qualquer coisa EXCETO `wmqJms`. Se você usar `wmqJms` como o ID, o Liberty não será capaz de carregar adequadamente o adaptador de recursos. Isso ocorre porque `wmqJms` é o ID usado internamente para se referir ao recurso específico para o IBM MQ. Ele realmente cria uma NullPointerException.

Os exemplos a seguir mostram alguns fragmentos de um arquivo `server.xml` ao executar JMS 2.0:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

Sugestão: Observe o uso dos recursos jca-1.7 e jms-2.0 e a falta do recurso wmqJmsClient-2.0.

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Sugestão: Observe o uso de `mqJms` para o ID, que é preferencial. Não use `wmqJms`.

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

Sugestão: Observe o classloaderProviderRef de volta para o adaptador de recursos por meio do ID `mqJms`; isso permite que classes específicas do IBM MQsejam carregadas.

Restrições ao Rastrear Usando o Suporte Genérico JCA

O rastreio e a criação de log não são integrados ao sistema de rastreio do Liberty. Em vez disso, o rastreio do adaptador de recurso do IBM MQ deve ser ativado usando as propriedades do sistema Java ou um arquivo de configuração IBM MQ classes for JMS , conforme descrito em Rastreio IBM MQ classes for JMS aplicativos Para obter detalhes sobre como configurar propriedades do sistema Java em Liberty, consulte a WebSphere Liberty documentação.

Por exemplo, para ativar o rastreio do adaptador de recursos IBM MQ em Liberty 19.0.0.9, inclua uma entrada no arquivo Liberty `jvm.options`:

1. Crie um arquivo de texto denominado `jvm.options`
2. Insira as seguintes opções da JVM para ativar o rastreio, um por linha, neste arquivo:

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. Para aplicar essas configurações a um único servidor, salve `jvm.options` em:

   ```
   ${server.config.dir}/jvm.options
   ```

Para aplicar essas mudanças em todos os Liberty, salve `jvm.options` em:

```
${wlp.install.dir}/etc/jvm.options
```

Isso terá efeito para todas as JVMs que não possuem um arquivo `jvm.options` definido localmente.

4. Reinicie o servidor para ativar as mudanças

Isso resulta em um rastreio sendo gravado em um arquivo de rastreio chamado `MQRA-WLP_<process identifier>.trc` no diretório `<path_to_trace_to>`.

Suporte integral do Liberty XA com tabelas de definição de canal do cliente

Ao usar WebSphere Liberty 18.0.0.2 ou posterior, é possível usar grupos de gerenciadores de filas dentro da tabela de definição de canal de cliente (CCDT) em conjunto com transações XA. Isso significa que agora é possível fazer uso da distribuição e da disponibilidade da carga de trabalho, fornecida pelos grupos de gerenciadores de filas, ao mesmo tempo em que se mantém a integridade da transação.

No evento de erros de conectividade para um gerenciador de filas, o gerenciador de filas precisa ficar disponível novamente para que a transação possa ser resolvida. A recuperação da transação é gerenciada pelo Liberty e você pode precisar configurar o gerenciador de transações, para que um período de tempo apropriado seja permitido para que os gerenciadores de filas se tornem disponíveis novamente. Para obter mais informações, consulte Gerenciador de transações (transação) na documentação do produto WebSphere Liberty.

Esse é um recurso do lado do cliente, ou seja, é necessário um adaptador de recursos, não um gerenciador de filas.

Instalando o adaptador de recursos do IBM MQ

O adaptador de recursos do IBM MQ é fornecido como um archive de recursos (RAR). Instale o arquivo RAR em seu servidor de aplicativos. Pode ser necessário incluir diretórios no caminho do sistema.

Sobre esta tarefa

O adaptador de recursos IBM MQ é fornecido como um arquivo RAR (Resource Archive):

- JM 3.0 Para Jakarta Messaging 3.0, esse arquivo é chamado de `wmq.jakarta.jmsra.rar` O arquivo RAR contém o IBM MQ classes for Jakarta Messaging e a implementação do IBM MQ das interfaces Jakarta Connectors Architecture (JCA).
- JMS 2.0 Para JMS 2.0, esse arquivo é chamado de `wmq.jmsra.rar` O arquivo RAR contém o IBM MQ classes for JMS e a implementação do IBM MQ das interfaces Java EE Connector Architecture (JCA).

Quando você instala o adaptador de recursos como parte da instalação do produto IBM MQ , o arquivo RAR é instalado com IBM MQ classes for JMS no diretório mostrado em Tabela 61 na página 452

Tabela 61. Diretório IBM MQ contendo o arquivo RAR para cada plataforma

Plataforma	**Diretório**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Deve-se usar o adaptador de recursos do IBM MQ para se conectar ao IBM MQ de um servidor de aplicativos. Dependendo de qual servidor de aplicativos você está usando, o adaptador de recursos pode ser pré-instalado ou pode ser necessário instalá-lo você mesmo.

Tabela 62. Instalação do adaptador de recursos em um servidor de aplicativos

Servidor de aplicativos	**Pré-instalado ou precisa ser instalado?**
WebSphere Application Server traditional 9.0	O adaptador de recursos do IBM MQ 9.0 é pré-instalado no WebSphere Application Server traditional 9.0. Portanto, não é necessário instalar um novo adaptador de recursos no WebSphere Application Server traditional 9.0.
WebSphere Liberty	O WebSphere Liberty não contém o adaptador de recursos do IBM MQ, por isso, deve-se obtê-lo separadamente no Fix Central.
Outro servidor de aplicativos Java EE ou Jakarta EE	Obtenha o adaptador de recursos separadamente do Fix Central, como para WebSphere Liberty.

Procedimento

- Se você estiver se conectando ao IBM MQ a partir do WebSphere Libertyou a outro servidor de aplicativos Java EE ou Jakarta EE , faça download e instale o IBM MQ adaptador de recursos conforme descrito em “Instalando o adaptador de recursos no Liberty” na página 453.
- Linux AIX

  Para conexões de ligações em AIX and Linux sistemas, certifique-se de que o diretório que contém as bibliotecas Java Native Interface (JNI) esteja no caminho do sistema.

  Para obter o local desse diretório, que também contém as bibliotecas do IBM MQ classes for JMS, consulte “Configurando as bibliotecas do Java Native Interface (JNI)” na página 97.

  Windows No Windows, esse diretório é incluído automaticamente no caminho do sistema durante a instalação do IBM MQ classes for JMS.

  Sugestão: Como uma alternativa para configurar o caminho do sistema, o adaptador de recursos do IBM MQ tem uma propriedade chamada nativeLibraryPath que pode ser usada para especificar o local da biblioteca JNI. Por exemplo, no WebSphere Liberty, isso seria configurado conforme mostrado no exemplo a seguir:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

  Transações são suportadas no modo cliente e no modo ligações.

Instalando o adaptador de recursos no Liberty

Para se conectar ao IBM MQ a partir de WebSphere Libertyou a outros servidores de aplicativos Java EE ou Jakarta EE , deve-se usar o adaptador de recursos do IBM MQ Como o Liberty não contém o adaptador de recursos do IBM MQ, deve-se obtê-lo separadamente da Fix Central.

Antes de começar

Nota: As informações neste tópico não se aplicam ao WebSphere Application Server traditional 9.0. O adaptador de recursos do IBM MQ 9.0 é pré-instalado no WebSphere Application Server traditional 9.0. Portanto, não há nenhum requisito para instalar um novo adaptador de recursos nesse caso.

Antes de iniciar essa tarefa, certifique-se de ter um Java runtime environment (JRE) instalado em sua máquina e que o JRE foi incluído no caminho do sistema.

O instalador do Java que é usado nesse processo de instalação não requer que seja executado como raiz ou qualquer usuário específico. O único requisito é que o usuário como ele é executado tenha acesso de gravação para o diretório que você deseja que os arquivos entrem.

Para versões de Liberty a WebSphere Liberty 8.5.5 Fix Pack 1, se um EJB for implementado usando unicamente a configuração dentro do `ejb-jar.xml`, a versão de WebSphere Application Server que o Perfil Liberty está usando deverá ter o APAR PM89890 aplicado. Esse método de configuração é usado para o programa de verificação de instalação (IVT) do adaptador de recursos, por isso, esse APAR é necessário para que o IVT seja executado.

JM 3.0 Em IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado. Para usar o adaptador de recursos IBM MQ para Jakarta Messaging com Liberty, deve-se usar uma versão do Liberty que suporte Jakarta EE. Por exemplo, é possível usar o recurso Liberty generic messaging-3.0 .

Sobre esta tarefa

O arquivo JAR do adaptador de recursos que pode ser transferido por download da Fix Central é executável. Quando você executa esse arquivo executável, ele exibe o contrato de licença do IBM MQ, que deve ser aceito. Ele pede um diretório no qual instalar o adaptador de recursos do IBM MQ. O arquivo RAR do adaptador de recursos e o programa de teste de verificação de instalação (IVT) são, então, instalados nesse diretório. É possível aceitar o padrão ou especificar outro diretório, que pode ser o diretório dos adaptadores de recursos de um servidor de aplicativos ou qualquer outro diretório em seu sistema. Se não existir, o diretório será criado como parte da instalação.

Antes do IBM MQ 9.0, o nome do arquivo a ser transferido por download estava no formato de *V.R.M.F*-WS-MQ-Java-InstallRA.jar, por exemplo 8.0.0.6-WS-MQ-Java-InstallRA.jar. A partir do IBM MQ 9.0, o formato do nome do arquivo é *V.R.M.F*-IBM-MQ-Java-InstallRA.jar, por exemplo 9.0.0.0-IBM-MQ-Java-InstallRA.jar.

Depois de ter transferido por download e instalado o adaptador de recursos, você está pronto para configurá-lo em WebSphere Liberty.

Procedimento

1. Faça download do adaptador de recursos do IBM MQ por meio da Fix Central.
   a) Clique neste link: Adaptador de recursos do IBM MQ.
   b) Localize o adaptador de recursos para a sua versão do IBM MQ na lista de correções disponíveis exibida.
      Por exemplo:

      ```
      release level: 9.1.4.0-IBM-MQ-Java-InstallRA
      Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
      Servers
      ```

      Em seguida, clique no arquivo de adaptador de recursos e siga o processo de download.
2. Inicie a instalação inserindo o seguinte comando a partir do diretório no qual você transferiu o arquivo por download.
   No IBM MQ 9.0, o formato do comando é como a seguir:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   em que *V.R.M.F* é o número de Versão, Liberação, Modificação e Fix Pack e *V.R.M.F*-IBM-MQ-Java-InstallRA.jar é o nome do arquivo que foi transferido por download do Fix Central.
   Por exemplo, para instalar o adaptador de recursos IBM MQ para a liberação IBM MQ 9.1.4, o comando a seguir seria usado:

   ```
   java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
   ```

Nota: Para realizar essa instalação, deve-se ter um JRE instalado em sua máquina e incluído no caminho do sistema.

Ao inserir o comando, as seguintes informações são exibidas:

```
Para que seja possível usar, extrair ou instalar o IBM MQ 9.1, deve-se aceitar
os termos de 1. IBM Contrato Internacional de Licença para Avaliação de
Programas 2. Contrato Internacional de Licença do Programa IBM e adicional
. Leia os contratos de licença a seguir com atenção.

O contrato de licença é exibido separadamente usando a opção
--viewLicenseAgreement.
Pressione Enter para exibir os termos de licença agora, ou 'x' para ignorar.
```

3. Revise e aceite os termos de licença:
   a) Para exibir a licença, pressione Enter.

      Como alternativa, pressionar x ignora a exibição da licença.

      Após a exibição da licença ou imediatamente após a seleção de x, a seguinte mensagem aparece para informar que é possível optar por exibir os termos de licença adicionais:

      ```
      As informações adicionais sobre licenças são exibidas separadamente usando a opção
      --viewLicenseInfo.
      Pressione Enter para exibir informações adicionais sobre licenças agora ou 'x' para ignorar.
      ```

   b) Para exibir os termos de licença adicionais, pressione Enter.

      Como alternativa, pressionar x ignora a exibição dos termos de licença adicionais.

      Após a exibição dos termos de licença adicionais ou imediatamente após a seleção de x, a seguinte mensagem será exibida, solicitando que você aceite o contrato de licença:

      ```
      Ao escolher a opção "Concordo" abaixo, você concorda com os termos do
      contrato de licença e com os termos não IBM, se aplicável. Se você não
      concordar, selecione "Eu não concordo".

      Selecione [1] Eu concordo ou [2] Eu não concordo:
      ```

   c) Para aceitar o contrato de licença e continuar selecionando o diretório de instalação, selecione 1.

      Como alternativa, se você selecionar 2, a instalação será finalizada imediatamente.

      Se você selecionou 1, a mensagem a seguir aparece, solicitando que você selecione um diretório de instalação de destino:

      ```
      Insira um diretório para arquivos do produto ou mantenha em branco para aceitar o valor
      padrão.
      O diretório de destino padrão é H:\Liberty\WMQ
      Diretório de destino para arquivos do produto?
      ```

4. Especifique o diretório de instalação para o adaptador de recursos:
   - Se você deseja instalar o adaptador de recursos no local padrão, pressione Enter sem especificar um valor.
   - Se você deseja instalar o adaptador de recursos em um local diferente do padrão, especifique o nome do diretório no qual você deseja instalar o adaptador de recursos e, em seguida, pressione Enter.

   Após os arquivos serem instalados no local selecionado, uma mensagem de confirmação será exibida, conforme mostrado no exemplo a seguir:

   ```
   Extraindo arquivos para H:\Liberty\WMQ\wmq
   Todos os arquivos do produto foram extraídos com sucesso.
   ```

   Durante a instalação, um novo diretório com o nome `wmq` é criado dentro do diretório de instalação selecionado e os arquivos a seguir são instalados no diretório `wmq`:

   - O programa de teste de verificação de instalação `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) ou `wmq.jmsra.ivt` (JMS 2.0).
   - O arquivo RAR IBM MQ`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 ou `wmq.jmsra.rar` (JMS 2.0).

5. JMS 2.0

   Opcional: Configure o adaptador de recursos Java EE 7 (JMS 2.0) em WebSphere Liberty Profile.

As etapas que devem ser executadas para configurar o adaptador de recursos no Liberty são as seguintes. Para obter mais informações, consulte a documentação do produto WebSphere Application Server.

a) Inclua o recurso wmqJmsClient-2.0 no arquivo `server.xml` para permitir o trabalho com o adaptador de recursos IBM MQ.

   Para obter informações adicionais, consulte “Qual versão do adaptador de recursos usar” na página 444.

b) Inclua uma referência no arquivo `wmq.jmsra.rar` (JMS 2.0) que foi instalado.

Uma configuração de exemplo para suportar servlets e MDBs, com o JNDI, pode ser semelhante ao seguinte:

```
 <featureManager>
    <feature>wmqJmsClient-2.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

6. JM 3.0

Opcional: Configure o adaptador de recursos Jakarta EE 9 (Jakarta Messaging 3.0) em WebSphere Liberty Profile.

As etapas que devem ser executadas para configurar o adaptador de recursos no Liberty são as seguintes. Para obter mais informações, consulte a documentação do produto WebSphere Application Server.

a) Inclua o recurso wmqJmsClient-3.0 no arquivo `server.xml` para permitir trabalhar com o adaptador de recursos IBM MQ .

   Para obter informações adicionais, consulte “Qual versão do adaptador de recursos usar” na página 444.

b) Inclua uma referência no arquivo `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) que foi instalado.

Uma configuração de exemplo para suportar servlets e MDBs, com o JNDI, pode ser semelhante ao seguinte:

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

Nota: Se você estiver usando Open Liberty, em vez de WebSphere Liberty Profile, será necessário usar o recurso de suporte do adaptador de recursos genérico "messagingClient-3.0" no lugar de "wmqJmsClient-3.0" e outros aspectos da configuração serão diferentes. Consulte a documentação do Open Liberty para obter mais informações.

Configurando o adaptador de recursos do IBM MQ

Para configurar o adaptador de recursos do IBM MQ, você define vários recursos do Java Platform, Enterprise Edition Connector Architecture (JCA) e, opcionalmente, as propriedades do sistema. Deve-se também configurar o adaptador de recursos para executar o programa de teste de verificação da instalação (IVT). Isso é importante porque o serviço IBM pode exigir que este programa seja executado para indicar que qualquer servidor de aplicativos não IBM foi configurado corretamente.

Antes de começar

Esta tarefa supõe que você já esteja familiarizado com o JMS e IBM MQ classes for JMS. Muitas das propriedades usadas para configurar o adaptador de recursos do IBM MQ são equivalentes às propriedades de objetos do IBM MQ classes for JMS e têm a mesma função.

Sobre esta tarefa

Cada servidor de aplicativos fornece seu próprio conjunto de interfaces de administração. Alguns servidores de aplicativos fornecem interfaces gráficas com o usuário para definir recursos de JCA, mas outros requerem que o administrador grave planos de implementação XML. É, portanto, fora do escopo desta documentação fornecer informações sobre como configurar o adaptador de recursos do IBM MQ para cada servidor de aplicativos.

As etapas a seguir, portanto, focam apenas naquilo que é necessário configurar. Consulte a documentação fornecida com seu servidor de aplicativos para obter informações sobre como configurar um adaptador de recursos do JCA.

Procedimento

Defina os recursos JCA a seguir nas categorias a seguir:

- Defina as propriedades do objeto ResourceAdapter.

  Essas propriedades, que representam as propriedades globais do adaptador de recursos, como o nível de rastreio de diagnóstico, estão descritas em "Configuração para propriedades do objeto ResourceAdapter" na página 458.
- Defina as propriedades de um objeto ActivationSpec.

  Estas propriedades determinam como um MDB é ativado para comunicação de entrada. Para obter informações adicionais, consulte "Configurando o adaptador de recursos para comunicação de entrada" na página 461.
- Defina as propriedades de um objeto ConnectionFactory.

  O servidor de aplicativos usa essas propriedades para criar um objeto JMS ConnectionFactory para comunicação de saída. Para obter mais informações, consulte "Configurando o adaptador de recursos para comunicação de saída" na página 480.
- Defina as propriedades de um objeto de destino administrado.

  O servidor de aplicativos usa essas propriedades para criar uma um objeto Queue do JMS ou objeto Topic do JMS para comunicação de saída. Para obter informações adicionais, consulte "Configurando o adaptador de recursos para comunicação de saída" na página 480.
- Opcional: Defina um plano de implementação para o adaptador de recursos.

  O arquivo RAR do adaptador de recursos IBM MQ contém um arquivo chamado `META-INF/ra.xml`, que contém um descritor de implementação para o adaptador de recursos. Esse descritor de implementação é definido pelo esquema XML no https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd e contém informações sobre o adaptador de recursos e os serviços que ele fornece. Um servidor de aplicativos também pode requerer um plano de implementação para o adaptador de recursos. Este plano de implementação é específico ao servidor de aplicativos.

Especifique propriedades do sistema JVM conforme necessário:

- Se você estiver usando a Segurança da Camada de Transporte (TLS), especifique os locais do arquivo keystore e do arquivo de armazenamento confiável como propriedades do sistema JVM, como no exemplo a seguir:

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  Essas propriedades não podem ser propriedades de um objeto ActivationSpec ou ConnectionFactory e não é possível especificar mais de um keystore para um servidor de aplicativos. As propriedades

se aplicam a toda JVM e pode, portanto, afetar o servidor de aplicativos se outros aplicativos em execução no servidor de aplicativos estiverem usando conexões TLS. O servidor de aplicativos pode também reconfigurar essas propriedades para valores diferentes. Para obter mais informações sobre o uso de TLS com o IBM MQ classes for JMS, veja “Usando TLS com o IBM MQ classes for JMS” na página 259.

- Opcional: Se necessário, configure o adaptador de recursos para registrar mensagens de aviso ao seu log de saída padrão do servidor de aplicativos.

  Os logs do adaptador de recursos, avisos e mensagens de erro usam o mesmo mecanismo do IBM MQ classes for JMS. Para obter mais informações, consulte Erros de log para IBM MQ classes for JMS. Isso significa que, por padrão, as mensagens vão para um arquivo chamado `mqjms.log`. Para configurar o adaptador de recursos para registrar adicionalmente as mensagens de aviso no log de saída padrão do servidor de aplicativos, configure a propriedade de sistema JVM a seguir para seu servidor de aplicativos:

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  Esta é a mesma propriedade que aquela usada para controlar o rastreio para o IBM MQ classes for JMS. Assim como o IBM MQ classes for JMS, é possível usar uma propriedade do sistema apontando para o arquivo `jms.config` (consulte “O arquivo de configuração IBM MQ classes for JMS/Jakarta Messaging” na página 100). Para obter informações sobre como configurar uma propriedade de sistema JVM, consulte a documentação do servidor de aplicativos.

Configure o adaptador de recursos para executar o teste de verificação de instalação

- Configure o adaptador de recursos para executar o programa de teste de verificação da instalação (IVT) fornecido com o adaptador de recursos do IBM MQ.

  Para obter informações sobre o que precisa ser configurado a fim de executar o programa IVT, consulte “Verificando a instalação do adaptador de recursos” na página 501.

  Isso é importante porque o serviço IBM pode exigir que este programa seja executado para indicar que qualquer servidor de aplicativos não IBM foi configurado corretamente.

  Importante: Deve-se configurar o adaptador de recursos antes de poder executar o programa.

Configuração para propriedades do objeto ResourceAdapter

O objeto ResourceAdapter engloba as propriedades globais do adaptador de recursos do IBM MQ, como o nível de rastreio de diagnóstico. Para definir essas propriedades, use os recursos do seu adaptador de recursos, conforme descrito na documentação fornecida com seu servidor de aplicativos.

O objeto ResourceAdapter possui dois conjuntos de propriedades:

- Propriedades associadas ao rastreio de diagnóstico
- Propriedades associadas ao conjunto de conexões gerenciado pelo adaptador de recursos

A maneira na qual você define essas propriedades depende das interfaces de administração que o seu servidor de aplicativos fornece. Se você estiver usando o WebSphere Application Server traditional, veja “Configuração do WebSphere Application Server traditional” na página 460 ou se estiver usando o WebSphere Liberty, veja “Configuração do WebSphere Liberty” na página 460. Para outros servidores de aplicativos, veja a documentação do produto para seu servidor de aplicativos.

Para obter mais informações sobre a definição das propriedades associadas ao rastreamento de diagnóstico, consulte Rastreando o adaptador de recursos do IBM MQ

O adaptador de recursos gerencia um conjunto de conexões internas de conexões do JMS usadas para entregar mensagens a MDBs. O Tabela 63 na página 459 lista as propriedades do objeto ResourceAdapter que estão associadas ao conjunto de conexões.

Tabela 63. Propriedades do Objeto ResourceAdapter que estão associadas ao conjunto de conexões			
Nome da propriedade	**tipo**	**Valor padrão**	**Descrição**
maxConnections	Sequência	50	O número máximo de conexões com um gerenciador de filas do IBM MQ e o número máximo de MDBs implementados.
connectionConcurrency	Sequência	1	O número máximo de MDBs para compartilhar uma conexão JMS. O compartilhamento de conexões não é possível e essa propriedade sempre tem o valor 1.
reconnectionRetryCount	Sequência	5	O número máximo de tentativas feitas pelo adaptador de recursos para se reconectar a um gerenciador de filas do IBM MQ se uma conexão falhar.
reconnectionRetryInterval	Sequência	300 000	O tempo, em milissegundos, que o adaptador de recursos aguarda antes de tentar reconectar a um gerenciador de filas do IBM MQ.
startupRetryCount	Sequência	0	O número padrão de vezes para tentar e conectar um MDB na inicialização, se o gerenciador de filas não estiver em execução quando o servidor de aplicativos é iniciado.
startupRetryInterval	Sequência	30.000	O tempo de suspensão padrão entre as tentativas de conexão de inicialização (em milissegundos).
supportMQExtensions	Sequência	false	Reverte o comportamento do IBM MQ JMS para o comportamento pré-JMS 2.0. Para obter mais informações, consulte “Propriedade SupportMQExtensions” na página 334.
nativeLibraryPath	Sequência	<vazio>	O caminho a ser usado para carregar a biblioteca JNI do IBM MQ para permitir conexões do modo de ligações. Windows No Windows, o caminho do sistema também precisa conter o local da instalação do IBM MQ correspondente.

Quando um MDB é implementado no servidor de aplicativos, uma nova conexão do JMS é criada e uma conversa iniciada com o gerenciador de filas, contanto que o número máximo de conexões especificado pela propriedade maxConnection não seja excedido. O número máximo de MDBs é igual a, portanto, o número máximo de conexões. Se o número de MDBs implementados atingir esse máximo, qualquer tentativa de implementar outro MDB falhará. Se um MDB for interrompido, sua conexão poderá ser usada por outro MDB.

Em geral, se vários MDBs tiverem que ser implementados, o valor da propriedade maxConnections deverá ser aumentado.

As propriedades reconnectionRetryCount e reconnectionRetryInterval controlam o comportamento do adaptador de recursos quando as conexões com um gerenciador de filas do IBM MQ falharem devido a uma falha de rede, por exemplo. Quando uma conexão falha, o adaptador de recursos suspende a entrega de mensagens para todos os MDBs fornecidos por essa conexão por um intervalo especificado pela propriedade reconnectionRetryInterval. O adaptador de recursos, então, tenta se reconectar ao

gerenciador de filas. Se a tentativa falhar, o adaptador de recursos fará outras tentativas de reconexão nos intervalos especificados pela propriedade reconnectionRetryInterval até o limite imposto pela propriedade reconnectionRetryCount ser atingido. Se todas as tentativas falharem, a entrega será interrompida permanentemente até os MDBs serem reiniciados manualmente.

Em geral, o objeto ResourceAdapter não requer administração. No entanto, para ativar o rastreio de diagnóstico em sistemas AIX and Linux, por exemplo, é possível definir as seguintes propriedades:

```
traceEnabled:    true
traceLevel:      10
```

Essas propriedades não têm efeito se o adaptador de recursos não foi iniciado, que é o caso, por exemplo, quando aplicativos usando recursos do IBM MQ estão em execução apenas no contêiner do cliente. Nessa situação, é possível configurar as propriedades para rastreio de diagnóstico como propriedades de sistema da Java Virtual Machine (JVM). É possível configurar as propriedades usando o sinalizador -D no comando **java**, como no seguinte exemplo:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

Não é necessário definir todas as propriedades do objeto ResourceAdapter. Quaisquer propriedades deixadas sem especificação usam seus valores padrão. Em um ambiente gerenciado, é melhor não misturar as duas maneiras de especificar propriedades. Se você misturá-las, as propriedades do sistema JVM terão precedência sobre as propriedades do objeto ResourceAdapter.

Configuração do WebSphere Application Server traditional

As mesmas propriedades estão disponíveis para o adaptador de recursos no WebSphere Application Server traditional, mas elas devem ser configuradas dentro do painel de propriedades do adaptador de recursos (consulte Configurações do provedor JMS na documentação do produto WebSphere Application Server traditional. O rastreio é controlado pela seção de diagnósticos da configuração do WebSphere Application Server traditional. Para obter mais informações, veja Trabalhando com provedores de diagnóstico na documentação do produto WebSphere Application Server traditional.

Configuração do WebSphere Liberty

O adaptador de recursos é configurado usando elementos XML no arquivo `server.xml`, conforme mostrado no exemplo a seguir:

JM 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

O rastreio é ativado incluindo este elemento XML:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

Configurando o adaptador de recursos para comunicação de entrada

Para configurar a comunicação de entrada, defina as propriedades de um ou mais objetos ActivationSpec.

As propriedades de um objeto ActivationSpec determinam como um bean acionado por mensagens (MBD) recebe mensagens JMS de uma fila IBM MQ. O comportamento transacional do MDB é definido em seu descritor de implementação.

Um objeto ActivationSpec possui dois conjuntos de propriedades:

- Propriedades que são usadas para criar uma conexão do JMS em um gerenciador de filas do IBM MQ
- Propriedades que são usadas para criar um consumidor de conexão do JMS que entrega mensagens de forma assíncrona conforme elas chegam em uma fila especificada

A maneira na qual você define as propriedades de um objeto ActivationSpec depende das interfaces de administração fornecidas por seu servidor de aplicativos.

Propriedades usadas para criar uma conexão do JMS com um gerenciador de filas do IBM MQ

Todas as propriedades no Tabela 64 na página 461 são opcionais.

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
applicationName	Sequência	• O nome da classe de chamada, se estiver disponível, ajustado para não ter mais de 28 caracteres. Se ela não estiver disponível, a sequência `WebSphere MQ Client for Java` será usada.	O nome pelo qual um aplicativo é registrado com o gerenciador de filas. Esse nome do aplicativo é mostrado no comando **DISPLAY CONN MQSC/PCF** (em que o campo é chamado **APPLTAG**) ou na exibição IBM MQ Explorer **Conexões de Aplicativos** (em que o campo é chamado **App name**).
brokerCCDurSubQueue [1]	Sequência	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila a partir da qual um consumidor de conexão recebe mensagens de assinaturas duráveis
brokerCCSubQueue [1]	Sequência	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila a partir da qual um consumidor de conexão recebe mensagens de assinaturas não duráveis
brokerControlQueue [1]	Sequência	• **SYSTEM.BROKER.CONTROL.QUEUE** • Um nome da fila	O nome da fila de controle do broker
brokerQueueManager [1]	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas no qual o broker está em execução

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
brokerSubQueue [1]	Sequência	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila a partir da qual um consumidor de mensagens não duráveis recebe as mensagens
brokerVersion [1]	Sequência	• **unspecified** – Quando o broker for migrado da V6 para V7, configure essa propriedade de modo que os cabeçalhos RFH2 não sejam mais usados. Após a migração, essa propriedade não é mais relevante. • **V1** - Para usar um broker de publicação/assinatura do IBM MQ. Esse valor é o valor padrão se TRANSPORT for configurado para BIND ou CLIENT. • **V2** – Para usar um broker do IBM Integration Bus no modo nativo. Este valor é o valor padrão, se TRANSPORT estiver definido como DIRECT ou DIRECTHTTP.	A versão do broker que está sendo usada
ccdtURL	Sequência	• **null** • Um localizador uniforme de recursos (URL)	Uma URL que identifica o nome e o local do arquivo contendo a tabela de definição de canal do cliente (CCDT) e especifica como o arquivo pode ser acessado
CCSID	Sequência	• **819** • Um identificador de conjunto de caracteres codificados suportados pela máquina virtual Java (JVM)	O identificador do conjunto de caracteres codificados para uma conexão
channel	Sequência	• **SYSTEM.DEF.SVRCONN** • O nome de um canal de MQI	O nome do canal de MQI a ser usado
cleanupInterval [1]	int	• **3 600 000** • Um inteiro positivo	O intervalo, em milissegundos, entre as execuções em segundo plano do utilitário de limpeza de publicar/assinar
cleanupLevel [1]	Sequência	• **SAFE** • Nenhum • STRONG • FORCE • NONDUR	O nível de limpeza para um armazenamento de assinatura baseado no broker
clientID	Sequência	• **null** • Um identificador de cliente	O identificador do cliente para uma conexão

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
cloneSupport	Sequência	• **DISABLED** – Apenas uma instância de um assinante de tópico durável pode ser executada de cada vez. • ENABLED - Duas ou mais instâncias do mesmo assinante de tópico durável podem ser executadas simultaneamente, mas cada instância deve ser executada em uma máquina virtual Java (JVM) separada.	Se duas ou mais instâncias do mesmo assinante de tópico durável puderem ser executadas simultaneamente
connectionFactoryLookup	Sequência	• **null** • O nome JNDI para um objeto ConnectionFactory	Se essa propriedade for configurada, o ActivationSpec procurará um objeto JMS ConnectionFactory com o nome JNDI especificado no namespace JNDI do servidor de aplicativos e, em seguida, usará as propriedades desse objeto para criar uma conexão do JMS para um gerenciador de filas do IBM MQ com uma exceção. A única propriedade do ActivationSpec que será usada ao criar a conexão do JMS é o clientID. Para obter mais informações, consulte “Propriedades ActivationSpec connectionFactoryLookup e destinationLookup” na página 476.

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
connectionNameList	Sequência	• **localhost(1414)** • Uma sequência composta de itens separados por vírgulas, em que cada item tem o formato: `HOSTNAME(PORT)` em que *HOSTNAME* é um nome DNS ou um endereço IP.	Uma lista de nomes de conexão TCP/IP usada para comunicações de entrada. Quando especificado, **connectionNameList** substitui as propriedades **hostname** e **port**. Essa propriedade é usada para se reconectar a gerenciadores de filas de várias instâncias. **connectionNameList** é semelhante em forma a **localAddress**, mas não deve ser confundida com ela. **localAddress** especifica as características das comunicações locais, enquanto **connectionNameList** especifica como alcançar um gerenciador de filas remotas.
dynamicallyBalanced [4]	Booleana	• **falso** • true	Se este MDB pode ser solicitado para receber mensagens de um gerenciador de filas diferente como parte do balanceamento de aplicativos em um cluster uniforme.
failIfQuiesce	Booleana	• **true** • false	Indica se as chamadas para certos métodos falharão se o gerenciador de filas estiver em um estado de quiesce
headerCompression	Sequência	• **NONE** • SYSTEM - A compactação do cabeçalho da mensagem de RLE é executada	Uma lista de técnicas que podem ser usadas para compactar os dados do cabeçalho em uma conexão

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
hostName	Sequência	• **localhost** • Um nome do host • Um endereço IP	O nome do host ou o endereço IP do sistema em que o gerenciador de filas reside. As propriedades **hostname** e **port** são substituídas pela propriedade **connectionNameList** quando especificado.
localAddress	Sequência	• **null** • Uma sequência no formato: `[ host_name ][(low_port [, high_port ])]` em que *host_name* é um nome do host ou endereço IP, *low_port* e *high_port* são números de porta TCP e colchetes denotam um componente opcional	Para uma conexão com um gerenciador de filas, esta propriedade especifica um ou ambos os seguintes procedimentos: • A interface de rede local a ser usada • A porta local ou um intervalo de portas locais a ser usado **localAddress** é semelhante em forma a **connectionNameList**, mas não deve ser confundida com ela. **localAddress** especifica as características das comunicações locais, enquanto **connectionNameList** especifica como alcançar um gerenciador de filas remotas.
messageCompression	Sequência	• **NONE** • Uma lista de um ou mais dos seguintes valores separados por caracteres em branco: RLE ZLIBFAST ZLIBHIGH V 9.4.0 LZ4FAST V 9.4.0 LZ4HIGH	Uma lista de técnicas que podem ser usadas para compactar os dados da mensagem em uma conexão

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
messageRetention [1]	Booleana	• **true** – Mensagens indesejadas permanecem na fila de entrada • false – As mensagens não desejadas são tratadas de acordo com suas opções de disposição	Indica se o consumidor da conexão manterá mensagens indesejadas na fila de entrada
messageSelection [1]	Sequência	• **CLIENT** • BROKER	Determina se a seleção de mensagem é feita pelo IBM MQ classes for JMS ou pelo broker. A seleção de mensagens pelo broker não é suportada quando brokerVersion tem o valor 1.
senha	Sequência	• **null** • Uma senha	A senha padrão a ser usada ao criar uma conexão com o gerenciador de filas
pollingInterval [1]	int	• **5000** • Qualquer número inteiro positivo	Se cada listener de mensagem dentro de uma sessão não tiver mensagens adequadas em sua fila, este valor será o intervalo máximo, em milissegundos, que decorrerá antes que cada listener da mensagem tente novamente obter uma mensagem de sua fila. Se ocorrer com frequência o fato de nenhuma mensagem adequada estar disponível para qualquer um dos listeners da mensagem em uma sessão, considere aumentar o valor desta propriedade. Esta propriedade é relevante apenas se TRANSPORT tiver o valor BIND ou CLIENT.
port	int	• **1414** • Um número da porta TCP	A porta na qual o gerenciador de filas atende. As propriedades **hostname** e **port** são substituídas pela propriedade **connectionNameList** quando especificado.

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
providerVersion	sequência	• **unspecified** • Uma sequência em um dos formatos a seguir – V.R.M.F – V.R.M – V.R – V em que V, R, M e F são valores de números inteiros maiores ou iguais a zero.	A versão, liberação, nível de modificação e fix pack do gerenciador de filas ao qual o MDB pretende se conectar.
queueManager	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas ao qual se conectar
receiveExit [3]	Sequência	• **null** • Uma sequência composta por um ou mais itens separados por vírgulas, em que cada item é o nome completo de uma classe que implementa a interface do IBM MQ classes for Java, MQReceiveExit	Identifica um programa de saída de recebimento de canal ou uma sequência de programas de saída de recebimento a ser executada na sucessão
receiveExitInit	Sequência	• **null** • Uma sequência composta de um ou mais itens de dados do usuário separados por vírgulas	Os dados do usuário que são transmitidos para programas de saída de recebimento do canal quando são chamados

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
rescanInterval [1]	int	• **5000** • Qualquer número inteiro positivo	Quando um consumidor de mensagens no domínio ponto a ponto usa um seletor de mensagens para selecionar quais mensagens deseja receber, o IBM MQ classes for JMS procura na fila IBM MQ mensagens adequadas na sequência determinada pelo atributo **MsgDeliverySequence** da fila. Quando o IBM MQ classes for JMS localiza uma mensagem adequada e a entrega ao consumidor, o IBM MQ classes for JMS retoma a procura para a próxima mensagem adequada a partir de sua posição atual na fila. O IBM MQ classes for JMS continua a procura na fila desta maneira até atingir o final da fila ou até que o intervalo de tempo em milissegundos, conforme determinado pelo valor dessa propriedade, tenha expirado. Em cada caso, o IBM MQ classes for JMS retorna ao início da fila para continuar sua procura e um novo intervalo de tempo começa.
securityExit [3]	Sequência	• **null** • O nome completo de uma classe que implementa a interface do IBM MQ classes for Java, MQSecurityExit	Identifica um programa de saída de segurança do canal
securityExitInit	Sequência	• **null** • Uma sequência de dados do usuário	Os dados do usuário que são transmitidos para um programa de saída de segurança do canal quando são chamados

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
sendExit [3]	Sequência	• **null** • Uma sequência composta por um ou mais itens separados por vírgulas, em que cada item é o nome completo de uma classe que implementa a interface do IBM MQ classes for Java, MQSendExit	Identifica um programa de saída de envio de canal ou uma sequência de programas de saída de envio a ser executada na sucessão
sendExitInit	Sequência	• **null** • Uma sequência composta de um ou mais itens de dados do usuário separados por vírgulas	Os dados do usuário que são transmitidos para programas de saída de envio do canal quando são chamados
shareConvAllowed	Booleana	• NO-Uma conexão do cliente não pode compartilhar seu soquete.. • **YES** -Uma conexão do cliente pode compartilhar seu soquete	Se uma conexão do cliente puder compartilhar seu soquete com outras conexões do JMS de nível superior a partir do mesmo processo para o mesmo gerenciador de filas, se as definições de canal corresponderem
sparseSubscriptions [1]	Booleana	• **false** – As assinaturas recebem mensagens de correspondência frequentes. • true – As assinaturas recebem mensagens de correspondência não frequentes. Esse valor requer que a fila de assinaturas possa ser aberta para procurar.	Controla a política de recuperação de mensagens de um objeto TopicSubscriber
sslCertStores	Sequência	• **null** • Uma sequência de uma ou mais URLs de LDAP separadas por espaços em branco. Cada URL de LDAP possui o formato: `ldap://host_name [: port ]` em que *host_name* é um nome de host ou endereço IP, *port* é um número de porta TCP e colchetes denotam um componente opcional.	Os servidores Lightweight Directory Access Protocol (LDAP) que retêm as listas de revogação de certificado (CRLs) para uso em uma conexão TLS
sslCipherSuite	Sequência	• **null** • O nome de um CipherSuite	O CipherSuite a ser usado para uma conexão TLS
sslFipsRequired [2]	Booleana	• **falso** • true	Indica se uma conexão TLS deve usar um CipherSuite suportado pelo provedor do IBM Java JSSE FIPS (IBMJSSEFIPS)

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
sslPeerName	Sequência	• **null** • Um modelo para nomes distintos	Para uma conexão TLS, um modelo que é usado para verificar o nome distinto no certificado digital fornecido pelo gerenciador de filas
sslResetCount	int	• **0** • Um número inteiro no intervalo de 0 – 999999999	O número total de bytes enviados e recebidos por uma conexão TLS antes de as chaves secretas usadas pelo TLS serem renegociadas
sslSocketFactory	Sequência	Uma sequência que representa o nome de classe completo de uma classe que fornece uma implementação da interface javax.net.ssl.SSLSocketFactory. Opcionalmente, incluindo um argumento a ser transmitido para o método construtor, entre parênteses.	Quaisquer conexões estabelecidas no escopo do objeto administrado usam soquetes obtidos a partir desta implementação da interface SSLSocketFactory.
statusRefreshInterval [1]	int	• **60000** • Qualquer número inteiro positivo	O intervalo, em milissegundos, entre atualizações da transação de execução longa, que detecta quando um assinante perde sua conexão com o gerenciador de filas. Essa propriedade só será relevante se **subscriptionStore** tiver o valor QUEUE.
subscriptionStore [1]	Sequência	• **Broker** • MIGRATE • FILA	Determina onde o IBM MQ classes for JMS armazena dados persistentes sobre assinaturas ativas

Tabela 64. Propriedades de um objeto ActivationSpec que são usadas para criar uma conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
transportType	Sequência	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	Indica se uma conexão com um gerenciador de filas usa o modo cliente ou o modo de ligações. Se o valor `BINDINGS_THEN_CLIENT` for especificado, o adaptador de recursos tentará primeiro fazer uma conexão no modo de ligações. Se essa tentativa de conexão falhar, o adaptador de recursos tentará estabelecer uma conexão do modo cliente. z/OS Se uma especificação de ativação que está em execução em um sistema WebSphere Application Server for z/OS tiver sido configurada para usar o modo de transporte `BINDINGS_THEN_CLIENT` e uma conexão estabelecida anteriormente for interrompida, quaisquer tentativas de reconexão pela especificação de ativação primeiro tentará usar o modo de transporte `BINDINGS`. Se a tentativa de conexão de modo de transporte `BINDINGS` for malsucedida, a especificação de ativação tentará subsequentemente uma conexão de modo de transporte `CLIENT`.
nome do usuário	Sequência	• **null** • Um nome de usuário	O nome do usuário padrão a ser usado ao criar uma conexão com um gerenciador de filas
wildcardFormat	Sequência	• CHAR - Reconhece somente os caracteres curingas, conforme usados na versão 1 do broker • **TOPIC** – Reconhece somente curingas em nível do tópico, conforme usados na versão 2 do broker	Qual versão de sintaxe curinga deve ser usada

Notas:

1. Esta propriedade pode ser usada com a versão 70 do IBM MQ classes for JMS.
2. Para obter informações importantes sobre o uso da propriedade sslFipsRequired, consulte “Limitações do adaptador de recursos do IBM MQ” na página 448.
3. Para obter informações sobre como configurar o adaptador de recursos para que ele possa localizar uma saída, veja “Configurando o IBM MQ classes for JMS para usar saídas de canal” na página 287.
4. A propriedade dynamicallyBalanced não é suportada em conjunto com o suporte a transações XA. Se dinamicallyBalanced for "true", então o MDB deve ser configurado para desativar transações XA.

Propriedades usadas para criar um consumidor de conexão do JMS

Nota: Os parâmetros **destination** e **destinationType** devem ser definidos explicitamente. Todas as outras propriedades no Tabela 65 na página 472 são opcionais.

Tabela 65. Propriedades de um objeto ActivationSpec que são usadas para criar um consumidor de conexão do JMS

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
destino	Sequência	Um nome do destino	O destino a partir do qual receber mensagens. A propriedade **useJNDI** determina como o valor dessa propriedade é interpretado.
destinationLookup	Sequência	• **null** • O nome JNDI para um objeto de Destino	Se essa propriedade for configurada, o ActivationSpec procurará um objeto de Destino do JMS com o nome JNDI especificado no namespace JNDI do servidor de aplicativos e, em seguida, usará as propriedades desse objeto para criar um consumidor de conexão do JMS em preferência às outras propriedades especificadas no ActivationSpec. Para obter informações adicionais, consulte “Propriedades ActivationSpec connectionFactoryLookup e destinationLookup” na página 476.
destinationType	Sequência	• jakarta.jms.Queue (Jakarta Messaging 3.0) • jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	O tipo de destino, uma fila ou um tópico
maxMessages	int	• **1** • Um inteiro positivo	O número máximo de mensagens que podem ser designadas a uma sessão do servidor de uma vez. Se a especificação de ativação estiver entregando mensagens para um MDB em uma transação XA, um valor de 1 será usado independentemente da configuração dessa propriedade.

Tabela 65. Propriedades de um objeto ActivationSpec que são usadas para criar um consumidor de conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
maxPoolDepth	int	• **10** • Um inteiro positivo	O número máximo de sessões do servidor no conjunto de sessões do servidor usado pelo consumidor de conexão
messageSelector	Sequência	• **null** • Uma expressão do seletor de mensagem SQL92	Uma expressão do seletor de mensagem especificando quais mensagens devem ser entregues
nonASFTimeout	int	• **0** • Um inteiro positivo	Um valor positivo indica que a entrega de não ASF é usada. O valor é o tempo, em milissegundos, que uma solicitação get aguarda por mensagens que podem ainda não ter chegado (um get com chamada de espera). O valor padrão, 0, indica que a entrega ASF é usada. Esse parâmetro será válido se: • O aplicativo está em execução no WebSphere Application Server 7.0 ou mais recente • O aplicativo está em execução no WebSphere Liberty usando o nível apropriado do recurso do cliente wmqJms Para obter mais informações, consulte “Liberty e o adaptador de recursos do IBM MQ” na página 450.
nonASFRollbackEnabled	Booleana	• **false** – A mensagem é consumida mesmo se o MDB falhar • true - A falha no MDB faz com que a mensagem seja retrocedida para a fila.	Se a entrega de mensagens estiver dentro de um ponto de sincronização do IBM MQ caso o MDB seja não transacionado. Ignorado se o MDB for transacionado ou se **nonASFTimeout** for configurado para 0.
poolTimeout	int	• **300000** • Um inteiro positivo	O tempo, em milissegundos, que uma sessão de servidor não usada é mantida aberta no conjunto de sessões do servidor antes de ser fechada devido à inatividade

Tabela 65. Propriedades de um objeto ActivationSpec que são usadas para criar um consumidor de conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
readAheadAllowed	int	• **DESTINATION** – Determine se a leitura antecipada será permitida ao consultar a definição da fila ou do tópico. • DISABLED - A leitura antecipada não é permitida. • ENABLED – A leitura antecipada é permitida. • QUEUE – Determine se a leitura antecipada é permitida, fazendo referência à definição de fila. • TOPIC – Determine se a leitura antecipada é permitida, fazendo referência à definição de tópico.	Se o encadeamento de navegação da especificação de ativação tem ou não permissão para usar a leitura antecipada para navegar por várias mensagens do destino em um buffer interno, antes de entregar as sessões do servidor para o consumo destrutivo. **Nota:** A ativação da leitura antecipada pode resultar em um aumento das mensagens do JMSCC0108, em uma redução no desempenho ou em ambos se a taxa de processamento do MDB não acompanhar a taxa de mensagens de navegação do destino.
readAheadClosePolicy	int	• **ALL** – Todas as mensagens no buffer interno de leitura antecipada são entregues para o MDB antes de parar. • CURRENT – Somente a chamada do MDB atual é concluída, possivelmente deixando as mensagens no buffer interno de leitura antecipada, que depois são descartadas.	O que acontece com as mensagens no buffer interno de leitura antecipada quando o MDB é interrompido pelo administrador.
receiveCCSID	int	• **0** - Use JVM `Charset.defaultCharset` • 1208 - UTF-8 • Um identificador de conjunto de caracteres codificados suportados	A propriedade de destino que configura o destino CCSID para a conversão de mensagens do gerenciador de filas. O valor será ignorado, a menos que **`receiveConversion`** seja configurado como `QMGR`.
receiveConversion	Sequência	• **CLIENT_MSG** • QMGR	A propriedade de destino que determina se a conversão de dados será executada pelo gerenciador de filas.

Tabela 65. Propriedades de um objeto ActivationSpec que são usadas para criar um consumidor de conexão do JMS (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
sharedSubscription	Booleana	• **False** – O MDB não deve abrir a assinatura como uma assinatura compartilhada. • True – O MDB deve abrir a assinatura como uma assinatura compartilhada (com as regras que o JMS 2.0 implica, consulte a especificação JMS 2.0 no Java.net).	Controla como um MDB é conduzido a partir de uma assinatura compartilhada. Para obter mais informações sobre como usar esta propriedade, consulte “Exemplos de como definir a propriedade sharedSubscription” na página 478.
startTimeout	int	• **10 000** • Um inteiro positivo	O tempo, em milissegundos, dentro do qual uma entrega de mensagem para um MDB deverá iniciar depois de o trabalho de entrega da mensagem ter sido planejado. Se esse tempo expirar, a mensagem será retrocedida na fila.
subscriptionDurability	Sequência	• **Não durável** – A assinatura não durável é usada para entregar mensagens para uma assinatura MDB para o tópico. • – Durável Uma assinatura durável é usada para entregar mensagens para uma assinatura MDB para o tópico.	Se uma assinatura durável ou não durável é usada para entregar mensagens para uma assinatura MDB para o tópico
subscriptionName	Sequência	• **"" (empty string)** • Um nome de assinatura	O nome da assinatura durável
useJNDI	Booleana	• **false** – A propriedade chamada de destino é interpretada como o nome de uma fila ou um tópico do IBM MQ. • true-A propriedade chamada de destino é interpretada como o nome de um dos seguintes objetos no namespace JNDI do servidor de aplicativos: – jakarta.jms.Queue (Jakarta Messaging 3.0) – jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated Determina como o valor da propriedade chamada de destino é interpretado **Nota:** Essa propriedade foi descontinuada no IBM MQ 9.0. A propriedade destinationLookup deve ser usada em substituição.

Conflitos e dependências de propriedade

Um objeto ActivationSpec pode ter propriedades conflitantes. Por exemplo, é possível especificar propriedades TLS para uma conexão no modo de ligações. Nesse caso, o comportamento é determinado pelo tipo de transporte e o domínio de mensagens, que é ponto a ponto ou publicar/assinar, conforme determinado pela propriedade **destinationType**. Todas as propriedades que não são aplicáveis ao tipo de transporte especificado ou domínio de sistema de mensagens são ignoradas.

Se você definir uma propriedade que requer que outras propriedades sejam definidas, mas não definir estas outras propriedades, o objeto ActivationSpec emitirá uma exceção InvalidPropertyException quando seu método de validação() for chamado durante a implementação de um MDB. A exceção é relatada ao administrador do servidor de aplicativos de uma maneira que depende do servidor de aplicativos. Por exemplo, se você configurar a propriedade subscriptionDurability para Durável, indicando que deseja utilizar assinaturas duráveis, também deverá definir a propriedade **subscriptionName**.

Se as propriedades chamadas **ccdtURL** e **channel** forem definidas, uma exceção InvalidPropertyException será lançada. No entanto, se você definir somente a propriedade **ccdtURL** , deixando a propriedade chamada **channel** com seu valor padrão de SYSTEM.DEF.SVRCONN, nenhuma exceção é lançada e a tabela de definições de canal do cliente identificada pela propriedade **ccdtURL** é usada para iniciar uma conexão JMS .

Propriedades ActivationSpec connectionFactoryLookup e destinationLookup

A especificação do JMS 2.0 introduziu duas novas propriedades do ActivationSpec. As propriedades connectionFactoryLookup e destinationLookup podem ser fornecidas com um nome JNDI de um objeto administrado a ser usado em preferência às outras propriedades ActivationSpec.

Por exemplo, se um connection factory for definido em JNDI e o nome JNDI desse objeto for especificado na propriedade de Consulta connectionFactorypara uma especificação de ativação, todas as propriedades do connection factory definidas em JNDI serão usadas em preferência às propriedades em Tabela 64 na página 461.

Se um destino for definido em JNDI e o nome JNDI for configurado na propriedade destinationLookup de ActivationSpec, os valores deles serão usados em preferência aos valores em Tabela 65 na página 472. Para obter mais informações sobre como essas duas propriedades são usadas, veja “Propriedades ActivationSpec connectionFactoryLookup e destinationLookup” na página 476.

Essas duas propriedades podem ser usadas para especificar os nomes JNDI de objetos ConnectionFactory e Destination que são usados em preferência às propriedades do ActivationSpec, conforme definido em Tabela 64 na página 461 e Tabela 65 na página 472.

É importante observar os pontos a seguir que descrevem como essas propriedades funcionam em detalhes.

connectionFactoryLookup
O ConnectionFactory consultado a partir do JNDI é usado como uma origem das propriedades listadas em Tabela 64 na página 461. O objeto ConnectionFactory não é usado para criar realmente nenhuma conexão JMS, somente as propriedades do objeto são consultadas. Essas propriedades do ConnectionFactory substituem quaisquer propriedades definidas no ActivationSpec. Há uma única exceção para isso. Se o ActivationSpec tiver a propriedade **ClientID** configurada, o valor dessa propriedade substituirá o valor especificado no ConnectionFactory. Isso ocorre porque um cenário comum está usando um único ConnectionFactory com múltiplos ActivationSpecs. Isso simplifica a administração. No entanto, a especificação JMS 2.0 afirma que cada conexão JMS criada por um ConnectionFactory deve ter um **ClientID** exclusivo. Devido a isso, ActivationSpecs precisa ter a capacidade de substituir qualquer valor configurado no ConnectionFactory. Se nenhum **ClientID** for configurado no ActivationSpec, qualquer valor no connection factory será usado.

destinationLookup
Uma propriedade **Destination** e uma **UseJndi** são definidas no ActivationSpec. Se a sinalização **UseJndi** estiver configurada como `true`, o texto especificado na propriedade de destino será considerado um nome JNDI e um objeto de destino com esse nome JNDI será consultado pelo JNDI.

A propriedade destinationLookup se comporta exatamente da mesma maneira. Se ela tiver sido configurada, um objeto de destino com o nome JNDI especificado pela propriedade será consultado no JNDI. Essa propriedade tem precedência sobre a propriedade **useJNDI**.

Deprecated A propriedade useJNDI foi descontinuada em IBM MQ 9.0 , pois a propriedade **destinationLookup** é a especificação JMS 2.0 ou equivalente mais recente de executar a mesma função..

Propriedades ActivationSpec sem equivalentes no IBM MQ classes for JMS

A maioria das propriedades de um objeto ActivationSpec são equivalentes a propriedades de objetos IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , ou a parâmetros de métodos IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging No entanto, três propriedades de ajuste e uma propriedade de usabilidade não têm equivalentes em IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging:

startTimeout
O tempo, em milissegundos, em que o gerenciador de trabalho do servidor de aplicativos aguarda para que os recursos fiquem disponíveis depois que o adaptador de recursos planeja um objeto de trabalho para entregar uma mensagem para um MDB. Se esse tempo decorrer antes da entrega da mensagem ser iniciada, o objeto de trabalho atinge o tempo limite, a mensagem é retrocedida para a fila e o adaptador de recursos pode tentar entregar a mensagem novamente. Um aviso é gravada para rastreio de diagnóstico, se ativado, mas, do contrário, não afeta o processo de entrega de mensagens. Você pode esperar essa condição somente nos momentos em que o servidor de aplicativos estiver passando por uma carga muito alta. Se a condição ocorrer regularmente, considere aumentar o valor desta propriedade para dar o gerenciador de trabalho mais tempo para planejar a entrega de mensagens.

maxPoolDepth
O número máximo de sessões do servidor no conjunto de sessões do servidor usado por um consumidor de conexão. Quando uma sessão do servidor é criada, ela inicia uma conversa com um gerenciador de filas. O consumidor de conexão usa uma sessão do servidor para entregar uma mensagem para um MDB. Uma profundidade de conjunto maior permite que mais mensagens sejam entregues simultaneamente em situações de alto volume, mas usa mais recursos do servidor de aplicativos. Se vários MDBs tiverem que ser implementados, considere tornar a profundidade do conjunto menor para manter o carregamento no servidor de aplicativos em um nível gerenciável. Cada consumidor de conexão usa seu próprio servidor do conjunto de sessão, de modo que essa propriedade não define o número total de sessões de servidor disponíveis para todos os consumidores de conexão.

poolTimeout
O tempo, em milissegundos, que uma sessão de servidor não usada é mantida aberta no conjunto de sessões do servidor antes de ser fechada devido à inatividade. Um aumento temporário na carga de trabalho da mensagem faz com que as sessões do servidor adicionais sejam criadas a fim de distribuir a carga mas, após a carga de trabalho da mensagem retorna ao normal, as sessões adicionais do servidor permanecem no conjunto e não são usadas.

Toda vez que uma sessão do servidor for usada, ela será marcado com um registro de data e hora. Periodicamente, um encadeamento scavenger verifica que cada sessão de servidor foi usada dentro do período especificado por essa propriedade. Se uma sessão do servidor não foi usada, ela é fechada e removida do conjunto de sessões do servidor. Uma sessão do servidor não pode ser fechada imediatamente após o período especificado ter decorrido, esta propriedade representa o período mínimo de inatividade antes da remoção.

useJNDI
Para obter uma descrição desta propriedade, consulte Tabela 65 na página 472.

Implementando um MDB

Para implementar um MDB, primeiro defina as propriedades de um objeto ActivationSpec, especificando as propriedades que o MDB requer. O exemplo a seguir é um conjunto típico de propriedades que podem ser definidas explicitamente:

JM 3.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

O servidor de aplicativos usa as propriedades para criar um objeto ActivationSpec, que é, então, associado a um MDB. As propriedades do objeto ActivationSpec determinam como as mensagens são entregues para o MDB. A implementação do MDB falhará se o MDB requer transações distribuídas, mas o adaptador de recursos não suporta transações distribuídas. Para obter informações sobre como instalar o adaptador de recursos para que as transações distribuídas sejam suportados, consulte “Instalando o adaptador de recursos do IBM MQ” na página 452.

Se mais de um MDB está recebendo mensagens do mesmo destino, uma mensagem enviada no domínio ponto a ponto é recebida então por apenas um MDB, mesmo se outros MDBs são elegíveis para receber a mensagem. Em particular, se dois MDBs estiverem usando seletores de mensagens diferentes, e uma mensagem recebida corresponde a ambos os seletores de mensagens, apenas um dos MDBs recebe a mensagem. O MDB escolhido para receber uma mensagem é indefinido, e não é possível confiar em um MDB específico que recebe a mensagem. As mensagens enviadas no domínio de publicação/assinatura são recebidas por todos os MDBs elegíveis.

Em algumas circunstâncias, uma mensagem entregue a um MDB pode ser retrocedida em uma fila do IBM MQ. Esse retrocesso pode ocorrer, por exemplo, se uma mensagem é entregue em uma unidade de trabalho que é, então, retrocedida. Uma mensagem que é retrocedida é entregue novamente, mas uma mensagem formatada incorretamente pode causar repetidamente a falha de um MDB e, portanto, não pode ser entregue. Essa mensagem é chamada de uma mensagem suspeita. É possível configurar o IBM MQ para que o IBM MQ classes for JMS transfira automaticamente uma mensagem suspeita para outra fila para investigação adicional ou descarte a mensagem.

Para obter detalhes sobre como manipular mensagens suspeitas, consulte “Manipulando mensagens suspeitas no IBM MQ classes for JMS” na página 238.

Conceitos relacionados

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Tarefas relacionadas

Configurando recursos JMS no WebSphere Application Server

Exemplos de como definir a propriedade sharedSubscription

Você pode definir a propriedade sharedSubscription de uma especificação de ativação dentro de um arquivo WebSphere Liberty `server.xml`. Como alternativa, é possível definir a propriedade dentro de um bean acionado por mensagens (MDB) usando anotações.

Exemplo: definindo dentro de um arquivo Liberty `server.xml`

Dentro de um arquivo WebSphere Liberty `server.xml`, você define uma especificação de ativação como mostrado no exemplo a seguir. Esse exemplo cria uma assinatura durável compartilhada em um gerenciador de filas em localhost/port 1490.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

Exemplo: definindo dentro de um MDB

Também é possível definir a propriedade sharedSubscription dentro do MDB usando anotações, conforme mostrado no exemplo a seguir:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

O exemplo a seguir mostra uma parte do código MDB que usa o método de anotações:

JM 3.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
```

```
      // implement business logic here
   }

}
```

Conceitos relacionados

Assinantes e assinaturas

Durabilidade da assinatura

“Assinaturas clonadas e compartilhadas” na página 332

Existem dois métodos para fornecer a vários consumidores acesso à mesma assinatura. Esses dois métodos são através do uso de assinaturas clonadas ou compartilhadas.

Configurando o adaptador de recursos para comunicação de saída

Para configurar a comunicação de saída, defina as propriedades de um objeto ConnectionFactory e um objeto de destino administrado.

Exemplo de como usar a comunicação de saída

Ao usar a comunicação de saída, um aplicativo em execução em um servidor de aplicativos inicia uma conexão com um gerenciador de filas e, em seguida, envia mensagens para suas filas e recebe mensagens de suas filas de maneira síncrona. Por exemplo, o método de servlet a seguir, doGet(), usa comunicação de saída:

JMS 3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");
```

```
// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

Quando o servlet recebe uma solicitação HTTP GET, ele recupera um objeto ConnectionFactory e um objeto Queue do namespace JNDI, e usa os objetos para enviar uma mensagem para uma fila do IBM MQ. O servlet, então, recebe a mensagem que ele enviou.

Recursos necessários para a comunicação de saída

Para configurar a comunicação de saída, defina os recursos do Java EE Connector Architecture (JCA) nas seguintes categorias:

- As propriedades de um objeto ConnectionFactory, que o servidor de aplicativos usa para criar um objeto JMS ConnectionFactory.
- As propriedades de um objeto de destino administrado, que o servidor de aplicativos usa para criar um objeto Fila do JMS ou um objeto Tópico do JMS.

A maneira como você define essas propriedades depende das interfaces de administração fornecidas por seu servidor de aplicativos. Os objetos ConnectionFactory, Queue e Topic criados pelo servidor de aplicativos são ligados a um namespace JNDI a partir de onde eles podem ser recuperados por um aplicativo.

Geralmente, você define um objeto ConnectionFactory para cada gerenciador de filas para os quais os aplicativos podem precisar se conectar. Você define um objeto Queue para cada fila que os aplicativos podem precisar acessar no domínio ponto a ponto. E define um objeto Topic para cada tópico para os quais os aplicativos podem desejar publicar ou assinar. Um objeto ConnectionFactory pode ser independente do domínio. Como alternativa, ele pode ser específico do domínio, um objeto QueueConnectionFactory para o domínio ponto a ponto ou um objeto TopicConnectionFactory para o domínio de publicar/assinar.

Sugestão: Com o JMS 2.0, um connection factory pode ser usado para criar conexões e contextos. Como resultado, é possível ter um conjunto de conexões associado a um connection factory que contém uma combinação de conexões e contextos. É recomendado que um connection factory seja usado somente para criar conexões ou contextos. Isso assegura que o conjunto de conexões para esse connection factory só contenha objetos de um único tipo, o que torna o conjunto mais eficiente.

Propriedades de um objeto ConnectionFactory

O Tabela 66 na página 482 lista as propriedades de um objeto ConnectionFactory. O servidor de aplicativos usa essas propriedades para criar um objeto JMS ConnectionFactory.

Tabela 66. Propriedades de um objeto ConnectionFactory

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
applicationName	Sequência	• O nome da classe de chamada, se estiver disponível, ajustado para não ter mais de 28 caracteres. Se ela não estiver disponível, a sequência `WebSphere MQ Client for Java` será usada.	O nome pelo qual um aplicativo é registrado com o gerenciador de filas. Esse nome do aplicativo é mostrado pelo comando **`DISPLAY CONN MQSC/PCF`** (em que o campo é chamado **`APPLTAG`**) ou na exibição IBM MQ Explorer **Conexões de Aplicativos** (em que o campo é chamado **`App name`**).
Fila brokerCCSub	Sequência	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila da qual um consumidor de conexão recebe mensagens de assinaturas não duráveis.
Fila brokerControl	Sequência	• **SYSTEM.BROKER.CONTROL.QUEUE** • Um nome da fila	O nome da fila de controle do broker.
Fila brokerPub	Sequência	• **SYSTEM.BROKER.DEFAULT.STREAM** • Um nome da fila	O nome da fila para onde as mensagens publicadas são enviadas (a fila de fluxo).
brokerQueueManager	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas no qual o broker está em execução.
Fila brokerSub	Sequência	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila a partir da qual um consumidor de mensagens não duráveis recebe as mensagens. Consulte a propriedade BROKERSUBQ para obter mais informações.
brokerVersion	Sequência	• **unspecified** – Depois que o broker foi migrado da V6 para V7, configure essa propriedade de modo que os cabeçalhos RFH2 não sejam mais usados. Após a migração, essa propriedade não é mais relevante. • **V1** - Para usar um broker de publicação/assinatura do IBM MQ. Este valor é o valor padrão, se TRANSPORT estiver definido como BIND ou CLIENT. • **V2** – Para usar um broker do IBM Integration Bus no modo nativo. Este valor é o valor padrão, se TRANSPORT estiver definido como DIRECT ou DIRECTHTTP.	A versão do broker que está sendo usada.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
ccdtURL	Sequência	• **null** • Um localizador uniforme de recursos (URL)	Uma URL que identifica o nome e o local do arquivo contendo a tabela de definição de canal de cliente (CCDT) e especifica como o arquivo pode ser acessado.
CCSID	Sequência	• **819** • Um identificador de conjunto de caracteres codificados suportados pela máquina virtual Java (JVM)	O identificador do conjunto de caracteres codificado para uma conexão.
channel	Sequência	• **SYSTEM.DEF.SVRCONN** • O nome de um canal de MQI	O nome do canal de MQI a ser usado.
cleanupInterval	int	• **3 600 000** • Um inteiro positivo	O intervalo, em milissegundos, entre as execuções em segundo plano do utilitário de limpeza de publicação/assinatura.
cleanupLevel	Sequência	• **SAFE** • Nenhum • STRONG • FORCE • NONDUR	O nível de limpeza para um armazenamento de assinaturas baseado no broker.
clientID	Sequência	• **null** • Um identificador de cliente	O identificador do cliente para uma conexão.
cloneSupport	Sequência	• **DISABLED** – Apenas uma instância de um assinante de tópico durável pode ser executada de cada vez. • ENABLED - Duas ou mais instâncias do mesmo assinante de tópico durável podem ser executadas simultaneamente, mas cada instância deve ser executada em uma máquina virtual Java (JVM) separada.	Se duas ou mais instâncias do mesmo assinante de tópico durável puderem ser executadas simultaneamente.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
connectionNameList	Sequência	• **localhost(1414)** • Uma sequência composta de itens separados por vírgulas, em que cada item tem o formato: `HOSTNAME(PORT)` em que *HOSTNAME* é um nome DNS ou um endereço IP.	Uma lista de nomes de conexão TCP/IP usada para comunicações de saída. **connectionNameList** substitui as propriedades **hostname** e **port**. Essa propriedade é usada para se reconectar a gerenciadores de filas de várias instâncias. **connectionNameList** é semelhante em forma a **localAddress**, mas não deve ser confundida com ela. **localAddress** especifica as características das comunicações locais, enquanto **connectionNameList** especifica como alcançar um gerenciador de filas remotas.
failIfQuiesce	Booleana	• **true** • false	Indica se as chamadas para certos métodos falharão se o gerenciador de filas estiver em um estado de quiesce.
headerCompression	Sequência	• **NONE** • SYSTEM - a compactação do cabeçalho da mensagem de RLE é executada.	Uma lista de técnicas que podem ser usadas para compactar os dados do cabeçalho em uma conexão.
hostName	Sequência	• **localhost** • Um nome do host • Um endereço IP	O nome do host ou o endereço IP do sistema em que o gerenciador de filas reside. As propriedades **hostname** e **port** são substituídas pela propriedade **connectionNameList** quando especificado.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
localAddress	Sequência	• **null** • Uma sequência no formato: `[ host_name ][(low_port [, high_port ])]` em que *host_name* é um nome do host ou endereço IP, *low_port* e *high_port* são números de porta TCP e colchetes denotam um componente opcional	Para uma conexão com um gerenciador de filas, esta propriedade especifica um ou ambos: • A interface de rede local a ser usada • A porta local ou um intervalo de portas locais a ser usado **localAddress** é semelhante em forma a **connectionNameList**, mas não deve ser confundida com ela. **localAddress** especifica as características das comunicações locais, enquanto **connectionNameList** especifica como alcançar um gerenciador de filas remotas.
messageCompression	Sequência	• **NONE** • Uma lista de um ou mais dos seguintes valores separados por caracteres em branco: RLE ZLIBFAST ZLIBHIGH V 9.4.0 LZ4FAST V 9.4.0 LZ4HIGH	Uma lista de técnicas que podem ser usadas para compactar os dados da mensagem em uma conexão.
messageSelection	Sequência	• **CLIENT** • BROKER	Determina se a seleção de mensagem é feita pelo IBM MQ classes for JMS ou pelo broker. A seleção de mensagens pelo broker não é suportada quando **brokerVersion** tem o valor 1.
senha	Sequência	• **null** • Uma senha	A senha padrão a ser usada ao criar uma conexão com o gerenciador de filas.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
pollingInterval	int	• **5000** • Qualquer número inteiro positivo	Se cada listener de mensagem dentro de uma sessão não tiver mensagens adequadas em sua fila, este valor será o intervalo máximo, em milissegundos, que decorrerá antes que cada listener da mensagem tente novamente obter uma mensagem de sua fila. Se ocorrer com frequência o fato de nenhuma mensagem adequada estar disponível para qualquer um dos listeners da mensagem em uma sessão, considere aumentar o valor desta propriedade. Essa propriedade só será relevante se **TRANSPORT** tiver o valor `BIND` ou `CLIENT`.
port	int	• **1414** • Um número da porta TCP	A porta na qual o gerenciador de filas atende. As propriedades **hostname** e **port** são substituídas pela propriedade **connectionNameList** quando especificado.
providerVersion	sequência	• **unspecified** • Uma sequência em um dos formatos a seguir – V.R.M.F – V.R.M – V.R – V em que V, R, M e F são valores de números inteiros maiores ou iguais a zero.	A versão, liberação, nível de modificação e fix pack do gerenciador de filas ao qual o aplicativo pretende se conectar.
Intervalo de pubAck	int	• **25** • Um inteiro positivo	O número de mensagens publicadas por um publicador antes de o IBM MQ classes for JMS solicitar uma confirmação do broker.
queueManager	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas para conexão.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
receiveExit [3]	Sequência	• **null** • Uma sequência composta por um ou mais itens separados por vírgulas, em que cada item é o nome completo de uma classe que implementa a interface do IBM MQ classes for Java, MQReceiveExit	Identifica um programa de saída de recebimento de canal ou uma sequência de programas de saída de recebimento a ser executada na sucessão.
receiveExitInit	Sequência	• **null** • Uma sequência composta de um ou mais itens de dados do usuário separados por vírgulas	Os dados do usuário que são transmitidos para programas de saída de recebimento do canal quando são chamados.
rescanInterval	int	• **5000** • Qualquer número inteiro positivo	Quando um consumidor de mensagens no domínio ponto a ponto usa um seletor de mensagem para selecionar quais mensagens ele deseja receber, o IBM MQ classes for JMS procura na fila do IBM MQ mensagens adequadas na sequência determinada pelo atributo **MsgDeliverySequence** da fila. Quando o IBM MQ classes for JMS localiza uma mensagem adequada e a entrega ao consumidor, o IBM MQ classes for JMS retoma a procura para a próxima mensagem adequada a partir de sua posição atual na fila. O IBM MQ classes for JMS continua a procura na fila desta maneira até atingir o final da fila ou até que o intervalo de tempo em milissegundos, conforme determinado pelo valor dessa propriedade, tenha expirado. Em cada caso, o IBM MQ classes for JMS retorna ao início da fila para continuar sua procura e um novo intervalo de tempo começa.
securityExit [3]	Sequência	• **null** • O nome completo de uma classe que implementa a interface do IBM MQ classes for Java, MQSecurityExit	Identifica um programa de saída de segurança do canal.
securityExitInit	Sequência	• **null** • Uma sequência de dados do usuário	Os dados do usuário que são transmitidos para um programa de saída de segurança do canal quando ele é chamado.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
sendCheckCount	int	• **0** • Qualquer número inteiro positivo	O número de chamadas de envio a serem permitidas entre a verificação de erros de colocação assíncronos em uma única sessão do JMS não transicionada.
sendExit [3]	Sequência	• **null** • Uma sequência composta por um ou mais itens separados por vírgulas, em que cada item é o nome completo de uma classe que implementa a interface do IBM MQ classes for Java, MQSendExit	Identifica um programa de saída de envio de canal ou uma sequência de programas de saída de envio a serem executadas na sucessão.
sendExitInit	Sequência	• **null** • Uma sequência composta de um ou mais itens de dados do usuário separados por vírgulas	Os dados do usuário que são transmitidos para programas de saída de envio do canal quando são chamados.
shareConvAllowed	Booleana	• NO-Uma conexão do cliente não pode compartilhar seu soquete.. • **YES** -Uma conexão do cliente pode compartilhar seu soquete	Se uma conexão do cliente puder compartilhar seu soquete com outras conexões do JMS de nível superior a partir do mesmo processo no mesmo gerenciador de filas, se as definições do canal corresponderem.
sparseSubscriptions	Booleana	• **false** – As assinaturas recebem mensagens de correspondência frequentes. • true – As assinaturas recebem mensagens de correspondência não frequentes. Esse valor requer que a fila de assinaturas possa ser aberta para procurar.	Controla a política de recuperação de mensagens de um objeto TopicSubscriber.
sslCertStores	Sequência	• **null** • Uma sequência de uma ou mais URLs de LDAP separadas por espaços em branco. Cada URL de LDAP possui o formato: `ldap://host_name [: port ]` em que *host_name* é um nome de host ou endereço IP, *port* é um número de porta TCP e colchetes denotam um componente opcional.	Os servidores Lightweight Directory Access Protocol (LDAP) que retêm as listas de revogação de certificado (CRLs) para uso em uma conexão TLS.
sslCipherSuite	Sequência	• **null** • O nome de um CipherSuite	O CipherSuite a ser usado para uma conexão TLS.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
sslFipsRequired [2]	Booleana	• **falso** • true	Indica se uma conexão TLS deve usar um CipherSuite que seja suportado pelo provedor do IBM Java JSSE FIPS (IBMJSSEFIPS).
sslPeerName	Sequência	• **null** • Um modelo para nomes distintos	Para uma conexão TLS, um modelo que é usado para verificar o nome distinto no certificado digital fornecido pelo gerenciador de filas.
sslResetCount	int	• **0** • Um número inteiro no intervalo de 0 – 999999999	O número total de bytes enviados e recebidos por uma conexão TLS antes de as chaves secretas usadas pelo TLS serem renegociadas.
sslSocketFactory	Sequência	Uma sequência que representa o nome de classe completo de uma classe que fornece uma implementação da interface javax.net.ssl.SSLSocketFactory, incluindo opcionalmente um argumento a ser transmitido para o método do construtor, entre parênteses.	Quaisquer conexões estabelecidas no escopo do objeto de destino administrado usam soquetes obtidos a partir desta implementação da interface SSLSocketFactory.
Intervalo statusRefresh	int	• **60000** • Qualquer número inteiro positivo	O intervalo, em milissegundos, entre atualizações da transação de execução longa, que detecta quando um assinante perde sua conexão com o gerenciador de filas. Essa propriedade só será relevante se **SUBSTORE** tiver o valor QUEUE.
subscriptionStore	Sequência	• **Broker** • MIGRATE • FILA	Determina onde o IBM MQ classes for JMS armazena os dados persistentes sobre assinaturas ativas.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
targetClientMatching	Booleana	• **true** • false	Se uma mensagem de resposta, enviada para a fila identificada pelo campo de cabeçalho JMSReplyTo de uma mensagem de entrada, tiver um cabeçalho MQRFH2, somente se a mensagem de entrada tiver um cabeçalho MQRFH2. Também é possível configurar essa propriedade para uma especificação de ativação. Para obter informações adicionais, consulte "Configurando a propriedade targetClientMatching para uma especificação de ativação" na página 499.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
temporaryModel	Sequência	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • Qualquer sequência	O nome da fila modelo a partir da qual as filas temporárias do JMS são criadas. Use SYSTEM.DEFAULT.MODEL.QUEUE se ambas as instruções a seguir forem verdadeiras: • Seu aplicativo usa uma fila temporária que aceitará mensagens não persistentes. • Apenas um aplicativo criará uma fila temporária no gerenciador de filas para a qual o ConnectionFactory aponta de cada vez. Observe que SYSTEM.DEFAULT.MODEL.QUEUE só pode ser aberto por um aplicativo de cada vez. Use SYSTEM.JMS.TEMPQ.MODEL nas seguintes situações: • Quando o aplicativo usa uma fila temporária que aceitará mensagens persistentes. • Se vários aplicativos podem se conectar ao gerenciador de filas para o qual o ConnectionFactory aponta, e esses aplicativos precisam criar filas temporárias ao mesmo tempo. Defina uma nova fila modelo com o atributo **DEFPSIST** configurado como YES e o atributo **DEFSOPT** configurado como SHARED na seguinte situação: • Quando o aplicativo usa uma fila temporária que aceitará mensagens não persistentes e vários aplicativos se conectarão ao gerenciador de filas para o qual o ConnectionFactory, e esses aplicativos precisam criar filas temporárias ao mesmo tempo. Quando a nova fila modelo for criada, configure a propriedade **temporaryModel** para o nome da nova fila modelo.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
tempQPrefix	Sequência	• **"" (empty string)** • Um prefixo que pode ser usado para formar o nome de uma fila dinâmica do IBM MQ. As regras para formar o prefixo são as mesmas que as regras para formar o conteúdo do campo **DynamicQName** em um descritor de objeto IBM MQ , estrutura MQOD, mas o último caractere não em branco deve ser um asterisco (*). Se o valor da propriedade for a sequência vazia, o IBM MQ classes for JMS usará o valor AMQ.* ao criar uma fila dinâmica.	O prefixo que é usado para formar o nome de uma fila dinâmica do IBM MQ.
tempTopicPrefix	Sequência	Qualquer sequência não nula que consiste apenas de caracteres válidos para uma sequência de tópico do IBM MQ	Ao criar tópicos temporários, o JMS gera uma cadeia de tópicos no formato "TEMP/*TEMPTOPICPREFIX*/*unique_id* " ou, se essa propriedade for deixada com o valor padrão, apenas "TEMP/*unique_id*". A especificação de um **TEMPTOPICPREFIX** não vazio permite que filas modelo específicas sejam definidas para criar as filas gerenciadas para assinantes de tópicos temporários criados nessa conexão.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
transportType	Sequência	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	Indica se uma conexão com um gerenciador de filas usa o modo cliente ou o modo de ligações. Se o valor `BINDINGS_THEN_CLIENT` for especificado, o adaptador de recursos tentará primeiro fazer uma conexão no modo de ligações. Se essa tentativa de conexão falhar, o adaptador de recursos tentará estabelecer uma conexão do modo cliente. z/OS Se uma especificação de ativação que está em execução em um sistema WebSphere Application Server for z/OS tiver sido configurada para usar o modo de transporte `BINDINGS_THEN_CLIENT` e uma conexão estabelecida anteriormente for interrompida, quaisquer tentativas de reconexão pela especificação de ativação primeiro tentará usar o modo de transporte `BINDINGS`. Se a tentativa de conexão de modo de transporte `BINDINGS` for malsucedida, a especificação de ativação tentará subsequentemente uma conexão de modo de transporte `CLIENT`.
nome do usuário	Sequência	• **null** • Um nome de usuário	O nome do usuário padrão a ser usado ao criar uma conexão com um gerenciador de filas.
wildcardFormat	int	• CHAR - Reconhece somente os caracteres curingas, conforme usados na versão 1 do broker • TOPIC – Reconhece somente curingas em nível do tópico, conforme usados na versão 2 do broker	Qual versão de sintaxe curinga deve ser usada.

Notas:

1. Para obter informações importantes sobre o uso da propriedade sslFipsRequired, consulte “Limitações do adaptador de recursos do IBM MQ” na página 448.
2. Para obter informações sobre como configurar o adaptador de recursos para que ele possa localizar uma saída, veja “Configurando o IBM MQ classes for JMS para usar saídas de canal” na página 287.

O exemplo a seguir mostra um conjunto típico de propriedades de um objeto ConnectionFactory:

```
channel:        SYSTEM.DEF.SVRCONN
```

```
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

Propriedades de um objeto de destino administrado

O servidor de aplicativos usa as propriedades de um objeto de destino administrado para criar um objeto Fila do JMS ou um objeto Tópico do JMS.

O Tabela 67 na página 494 lista as propriedades que são comuns para um objeto Fila e um objeto Tópico.

Tabela 67. Propriedades que são comuns para um objeto Fila e um objeto Tópico

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
CCSID	Sequência	• **1208** • Um identificador de conjunto de caracteres codificados suportados pela máquina virtual Java (JVM)	O identificador do conjunto de caracteres codificado para um destino.
codificação	Sequência	• **NATIVE** • Uma sequência de três caracteres: – O primeiro caractere especifica a representação de números inteiros binários: - *N* denota a codificação normal. - *R* denota a codificação reversa. – O segundo caractere especifica a representação de números inteiros decimais compactados: - *N* denota a codificação normal. - *R* denota a codificação reversa. – O terceiro caractere especifica a representação de números de vírgula flutuante: - *N* denota a codificação IEEE padrão. - *R* denota a codificação IEEE reversa. - *3* denota a codificação do zSeries. NATIVE é equivalente à sequência NNN.	A representação de números inteiros binários, números inteiros decimais compactados e números de vírgula flutuante, para o destino.
expiração	Sequência	• **APP** – O tempo de expiração de uma mensagem é determinado pelo produtor da mensagem. • UNLIM - Uma mensagem nunca expira. • 0 – Uma mensagem nunca expira. • Um número inteiro positivo representando o tempo de expiração de uma mensagem em milissegundos.	O tempo de expiração de uma mensagem enviada para o destino.

Tabela 67. Propriedades que são comuns para um objeto Fila e um objeto Tópico (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
failIfQuiesce	Sequência	• **true** • false	Se uma tentativa de acessar o destino falhar se o gerenciador de filas estiver em um estado de quiesce.
messageBodyStyle	Sequência	• **UNSPECIFIED** • JMS • MQ	É possível configurar a propriedade **messageBodyStyle** em filas e tópicos do JMS: UNSPECIFIED(default) • Ao enviar, o IBM MQ classes for JMS gera e inclui um cabeçalho MQRFH2, dependendo do valor de WMQ_TARGET_CLIENT. • Ao receber, o IBM MQ classes for JMS configura as propriedades da mensagem do JMS de acordo com os valores no MQRFH2, se presentes. MQRFH2 não é apresentado como parte do corpo da mensagem do JMS. JMS • Ao enviar, o IBM MQ classes for JMS gera automaticamente um cabeçalho MQRFH2 e o inclui na mensagem do IBM MQ. • Ao receber, o IBM MQ classes for JMS configura as propriedades da mensagem do JMS de acordo com os valores no MQRFH2, se presentes. MQRFH2 não é apresentado como parte do corpo da mensagem do JMS. MQ • Ao enviar, o IBM MQ classes for JMS não gera um MQRFH2. • Ao receber, o IBM MQ classes for JMS apresenta o MQRFH2 como parte do corpo da mensagem do JMS.

Tabela 67. Propriedades que são comuns para um objeto Fila e um objeto Tópico (continuação)

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
persistence	Sequência	• **APP** – A persistência de uma mensagem é determinada pelo produtor de mensagem. • QDEF – A persistência de uma mensagem é determinada pelo atributo **DefPersistence** da fila do IBM MQ. • PERS - Uma mensagem é persistente. • NON - Uma mensagem é não persistente. • HIGH – A persistência de uma mensagem é determinado pelo atributo **NonPersistentMessageClass** da fila do IBM MQ de acordo com a explicação em "Mensagens persistentes do JMS" na página 258.	A persistência de uma mensagem enviada para o destino.
priority	Sequência	• **APP** – A prioridade de uma mensagem é determinada pelo produtor da mensagem. • QDEF – A prioridade de uma mensagem é determinado pelo atributo **DefPriority** da fila do IBM MQ. • Um número inteiro no intervalo de 0, a prioridade mais baixa, e 9 para prioridade mais alta.	A prioridade de uma mensagem enviada para o destino.
putAsyncAllowed	Sequência	• QUEUE – Determine se as colocações assíncronas são permitidas consultando a definição de fila. • TOPIC – Determine se as colocações assíncronas são permitidas consultando a definição de tópico. • DESTINO – Determine se as colocações assíncronas são permitidas consultando a definição de fila ou tópico. • DISABLED – Colocações assíncronas não são permitidas. • ENABLED - Colocações assíncronas são permitidas.	Indica se os produtores de mensagens têm permissão para usar postagens assíncronas para enviar mensagens para este destino.

Tabela 67. Propriedades que são comuns para um objeto Fila e um objeto Tópico (continuação)			
Nome da propriedade	**tipo**	**Valores válidos (valor padrão em negrito)**	**Descrição**
readAheadAllowed	int	• **DESTINATION** – Determine se a leitura antecipada será permitida ao consultar a definição da fila ou do tópico. • DISABLED - A leitura antecipada não é permitida. • ENABLED – A leitura antecipada é permitida. • QUEUE – Determine se a leitura antecipada é permitida, fazendo referência à definição de fila. • TOPIC – Determine se a leitura antecipada é permitida, fazendo referência à definição de tópico.	Se os consumidores de mensagens e os navegadores de filas têm permissão para usar a leitura antecipada para obter as mensagens não persistentes do destino em um buffer interno antes de recebê-las.
receiveCCSID	int	• **0** - Use JVM `Charset.defaultCharset` • 1208 - UTF-8 • Um identificador de conjunto de caracteres codificados suportados	A propriedade de destino que configura o destino CCSID para a conversão de mensagens do gerenciador de filas. O valor será ignorado, a menos que **`receiveConversion`** seja configurado como `QMGR`.
receiveConversion	Sequência	• **CLIENT_MSG** • QMGR	A propriedade de destino que determina se a conversão de dados será executada pelo gerenciador de filas.
targetClient	Sequência	• **JMS** – O destino de uma mensagem é um aplicativo do JMS. • MQ - O destino de uma mensagem é um aplicativo não JMS IBM MQ.	Se o alvo de uma mensagem enviada ao destino é um aplicativo do JMS. Uma mensagem com um destino que é um aplicativo do JMS contém um cabeçalho MQRFH2.

O Tabela 68 na página 497 lista as propriedades que são específicas para um objeto Fila.

Tabela 68. Propriedades específicas de um objeto Fila			
Nome da propriedade	**tipo**	**Valores válidos (valor padrão em negrito)**	**Descrição**
baseQueueManagerName	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas que possui a fila subjacente do IBM MQ.
baseQueueName	Sequência	• **"" (empty string)** • Um nome da fila	O nome da fila subjacente do IBM MQ.

O Tabela 69 na página 498 lista as propriedades que são específicas para um objeto Tópico.

Tabela 69. As propriedades que são específicas para um objeto Tópico

Nome da propriedade	tipo	Valores válidos (valor padrão em negrito)	Descrição
baseTopicName	Sequência	• **"" (empty string)** • Um nome de tópico	O nome do tópico subjacente.
brokerCCDurSubQueue >	Sequência	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila a partir da qual um consumidor de conexão recebe mensagens de assinaturas duráveis.
brokerDurSubQueue	Sequência	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • Um nome da fila	O nome da fila a partir da qual um assinante de tópico durável recebe mensagens. Consulte a propriedade **BROKEDURRSUBQ** na documentação do IBM MQ Explorer para obter mais informações.
Fila brokerPub	Sequência	• **Not set** • Um nome da fila	O nome da fila para onde as mensagens publicadas são enviadas (a fila de fluxo). O valor dessa propriedade substitui o valor da propriedade **brokerPubQueue** do objeto ConnectionFactory. No entanto, se você não configurar o valor dessa propriedade, o valor da propriedade **brokerPubQueue** do objeto ConnectionFactory será usado em seu lugar.
brokerPubQueueManager	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas que possui a fila em que mensagens publicadas sobre o tópico são enviadas.
brokerVersion	Sequência	• **Not set** • 1 • 2	A versão do broker que está sendo usada. O valor dessa propriedade substitui o valor da propriedade **brokerVersion** do objeto ConnectionFactory. No entanto, se você não configurar o valor dessa propriedade, o valor da propriedade **brokerVersion** do objeto ConnectionFactory será usado em seu lugar.

O exemplo a seguir mostra um conjunto de propriedades de um objeto Fila:

```
expiry:              UNLIM
persistence:         QDEF
```

```
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

O exemplo a seguir mostra um conjunto de propriedades de um objeto Tópico:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

Tarefas relacionadas

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Configurando recursos JMS no WebSphere Application Server

Referências relacionadas

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Configurando a propriedade targetClientMatching para uma especificação de ativação

Será possível configurar a propriedade **targetClientMatching** para uma especificação de ativação para que um cabeçalho MQRFH2 seja incluído em mensagens de resposta quando as mensagens de solicitação não contiverem um cabeçalho MQRFH2. Isso significa que quaisquer propriedades de mensagem que um aplicativo definir em uma mensagem de resposta serão incluídas quando a mensagem for enviada.

Sobre esta tarefa

Se um aplicativo bean acionado por mensagens (MDB) consumir mensagens que não contiverem um cabeçalho MQRFH2 por meio de uma especificação de ativação do adaptador de recursos do JCA do IBM MQ e subsequentemente enviar mensagens de resposta para o Destino do JMS criado por meio do campo `JMSReplyTo` da mensagem de solicitação, as mensagens de resposta deverão incluir um cabeçalho MQRFH2, mesmo se as mensagens de solicitação não incluírem, caso contrário, qualquer propriedade de mensagem que o aplicativo tiver definido em uma mensagem de resposta será perdida.

A propriedade **targetClientMatching** define se uma mensagem de resposta enviada para a fila identificada pelo campo de cabeçalho JMSReplyTo de uma mensagem recebida terá um cabeçalho MQRFH2 apenas se a mensagem recebida tiver um cabeçalho MQRFH2. É possível configurar essa propriedade para uma especificação de ativação em ambos, WebSphere Application Server traditional e WebSphere Liberty.

Se você configurar o valor da propriedade **targetClientMatching** como `false`, um cabeçalho MQRFH2 poderá ser incluído em uma mensagem de resposta enviada para um Destino JMS criado por meio do cabeçalho JMSReplyTo de uma mensagem de solicitação recebida que não contém um MQRFH2. Isso é porque a propriedade **targetClient** no Destino JMS está configurada com o valor `0`, o que significa que as mensagens contêm um cabeçalho MQRFH2. A presença do cabeçalho MQRFH2 na mensagem de saída permite o armazenamento de propriedades de mensagem definidas pelo usuário na mensagem quando enviadas para a fila IBM MQ.

Se a propriedade **targetClientMatching** for configurada como `true` e uma mensagem de solicitação não incluir um cabeçalho MQRFH2, um cabeçalho MQRFH2 não será incluído na mensagem de resposta.

Procedimento

- No WebSphere Application Server traditional, use o console de administração para definir a propriedade **targetClientMatching** como uma propriedade customizada na especificação de ativação do IBM MQ:
  a) Na área de janela de navegação, clique em **Recursos -> JMS -> Especificações de ativação**.
  b) Selecione o nome da especificação de ativação que deseja visualizar ou alterar.
  c) Clique em **Propriedades customizadas -> Novo** e, em seguida, insira os detalhes da nova propriedade customizada.

Configure o nome da propriedade como `targetClientMatching`, o tipo como `java.lang.Boolean` e o valor como `false`.

- No WebSphere Liberty, especifique a propriedade **`targetClientMatching`** na definição de uma especificação de ativação dentro do `server.xml`.

  Por exemplo:

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

Conceitos relacionados

“Criando destinos em um aplicativo JMS” na página 225
Em vez de recuperar destinos como objetos administrados a partir de um namespace Java Naming and Directory Interface (JNDI), um aplicativo JMS pode usar uma sessão para criar destinos dinamicamente no tempo de execução. Um aplicativo pode usar um identificador uniforme de recursos (URI) para identificar uma fila do IBM MQ ou um tópico e, como opção, especificar uma ou mais propriedades de um objeto Queue ou Topic.

“Configurando o adaptador de recursos para comunicação de saída” na página 480
Para configurar a comunicação de saída, defina as propriedades de um objeto ConnectionFactory e um objeto de destino administrado.

Pausa do bean acionado por mensagens do IBM MQ no WebSphere Liberty

A propriedade **`maxSequentialDeliveryFailures`** para uma especificação de ativação define o número máximo de falhas de entrega de mensagem sequenciais para uma instância de bean acionada por mensagens (MDB) que o adaptador de recursos tolera antes de pausar o MDB.

Antes de começar

É necessário estar ciente do conjunto de eventos que podem fazer com que um MDB pause no WebSphere Liberty. O adaptador de recursos considera qualquer um dos seguintes como uma falha na entrega de mensagem:

- Uma exceção não verificada sendo emitida por meio do método **`onMessage`** do MDB.
- Uma `JMSException` que ocorre no processamento do adaptador de recursos, antes da entrega da mensagem para o MDB.
- Uma `JMSException` que ocorre no processamento do adaptador de recursos, após a entrega da mensagem para o MDB.
- A transação XA, ou a transação local, usada para consumir a mensagem que está sendo recuperada.
- Nenhum encadeamento disponível no servidor de aplicativos para entregar a mensagem para o MDB.

Sobre esta tarefa

O valor padrão da propriedade **`maxSequentialDeliveryFailures`** é *-1*, o que significa que o MDB nunca é pausado. Qualquer outro valor negativo é tratado da mesma forma que *-1*. Um valor de:

- *0* significa que o MDB pausa no primeiro erro
- *1* significa que o MDB pausa em dois erros sequenciais
- *2* significa que o MDB pausa em três erros sequenciais e assim por diante

É possível configurar essa propriedade para uma especificação de ativação apenas no WebSphere Liberty e quando o nível do Liberty é 18.0.0.4 ou superior.

Atenção: Se você configurar o atributo como um valor não padrão em qualquer ambiente de servidor de aplicativos diferente do Liberty, o valor será ignorado e uma mensagem de aviso será gravada no log.

Além disso, é possível instalar o adaptador de recursos do IBM MQ no WebSphere Liberty como um adaptador de recursos genérico. Esse procedimento desativa todos os recursos de integração do IBM MQ e do WebSphere Application Server e evita que o adaptador de recursos possa detectar que ele está em execução no Liberty. Portanto, a configuração de **maxSequentialDeliveryFailures** como maior ou igual a *0* não é suportada e resulta em uma mensagem de aviso no log.

Procedimento

- No WebSphere Liberty, especifique a propriedade **maxSequentialDeliveryFailures** na definição de uma especificação de ativação dentro do `server.xml`.

  Por exemplo:

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

Conceitos relacionados

“Configurando o adaptador de recursos para comunicação de saída” na página 480
Para configurar a comunicação de saída, defina as propriedades de um objeto ConnectionFactory e um objeto de destino administrado.

Verificando a instalação do adaptador de recursos

O programa de teste de verificação de instalação (IVT) para o adaptador de recursos do IBM MQ é fornecido como um arquivo EAR. Para usar o programa, deve-se implementá-lo e definir alguns objetos como recursos JCA.

Sobre esta tarefa

O programa de teste de verificação de instalação (IVT) é fornecido como um arquivo archive corporativo (EAR) chamado `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) ou `wmq.jmsra.ivt.ear` (JMS 2.0).. Esse arquivo é instalado com o IBM MQ classes for JMS no mesmo diretório que o arquivo RAR do adaptador de recurso IBM MQ , `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) ou `wmq.jmsra.rar` (JMS 2.0). Para obter informações sobre onde esses arquivos estão instalados, consulte “Instalando o adaptador de recursos do IBM MQ” na página 452

Deve-se implementar o programa IVT em seu servidor de aplicativos. O programa IVT inclui um servlet e um MDB que testa que uma mensagem pode ser enviada para e recebida de uma fila do IBM MQ.
É possível usar o programa IVT para verificar se o adaptador de recursos do IBM MQ foi configurado corretamente para suportar transações distribuídas. Caso esteja implementando o adaptador de recursos do IBM MQ em um servidor de aplicativos não IBM, o IBM Service poderá pedir que você demonstre o IVT em funcionamento para validar se seu servidor de aplicativos está configurado corretamente.

Para que seja possível executar o programa IVT, deve-se definir um objeto ConnectionFactory, um objeto da Fila e possivelmente um objeto da Especificação de ativação como recursos JCA e assegurar-se de que seu servidor de aplicativos crie objetos do JMS a partir dessas definições e ligue-as em um namespace JNDI. É possível escolher as propriedades dos objetos para que correspondam às configurações de host e porta do seu próprio QueueManager, mas o conjunto de propriedades a seguir é um exemplo simples:

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1550
queueManager:         QM1
transportType:        CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

O mecanismo usado para definir os objetos ConnectionFactory, Queue e Activation Specification varia dependendo do seu servidor de aplicativos. Por exemplo, para configurar essas propriedades dentro do WebSphere Liberty, inclua as entradas a seguir no arquivo `server.xml` do servidor de aplicativos:

JM 3.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

JMS 2.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

Por padrão, o programa IVT espera que um objeto ConnectionFactory seja ligado no namespace JNDI com o nome jms/ivt/IVTCF e um objeto da Fila seja ligado com o nome jms/ivt/IVTQueue. É possível usar diferentes nomes, mas se fizer isso, deverá inserir os nomes dos objetos na página inicial do programa IVT e modificar o arquivo EAR conforme apropriado.

Depois de ter implementado o programa IVT e o servidor de aplicativos ter criado os objetos do JMS e tê-los ligado ao namespace do JNDI, será possível iniciar o programa IVT concluindo as seguintes etapas.

Procedimento

1. Inicie o programa IVT inserindo uma URL no formato a seguir em seu navegador da web:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   em que *app_server_host* é o endereço IP ou nome do host do sistema no qual seu servidor de aplicativos está em execução, e *port* é o número da porta TCP na qual o servidor de aplicativos está atendendo. Aqui está um exemplo:

   ```
   http://localhost:9080/WMQ_IVT/
   ```

A seguir há um exemplo da página inicial exibida pelo programa IVT.

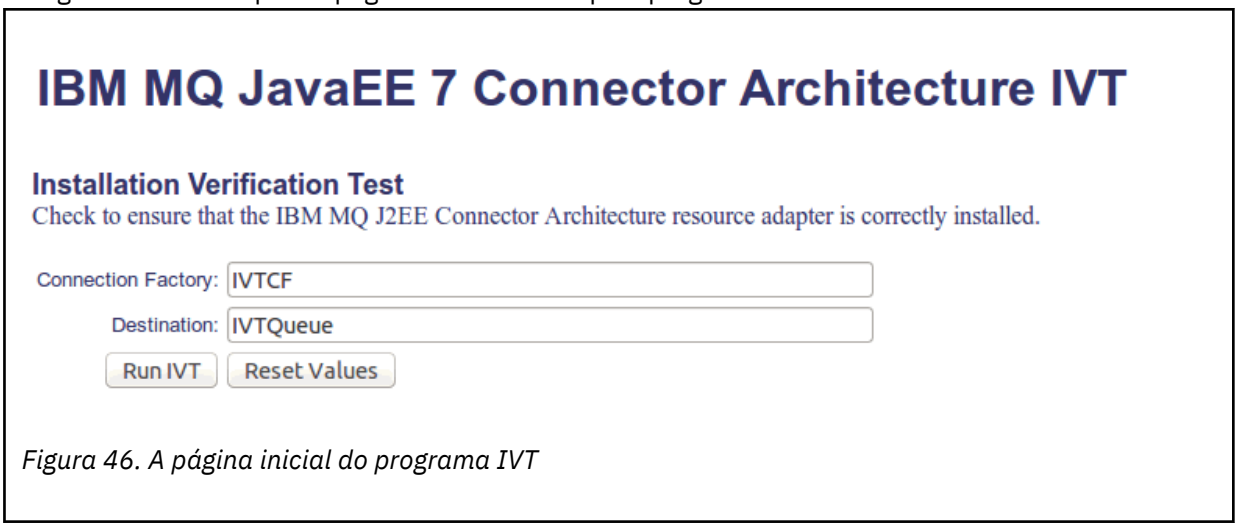


Figura 46. A página inicial do programa IVT

2. Para executar o teste, clique em **Run IVT**.

   Aqui está um exemplo da página exibida se o IVT for bem-sucedido.

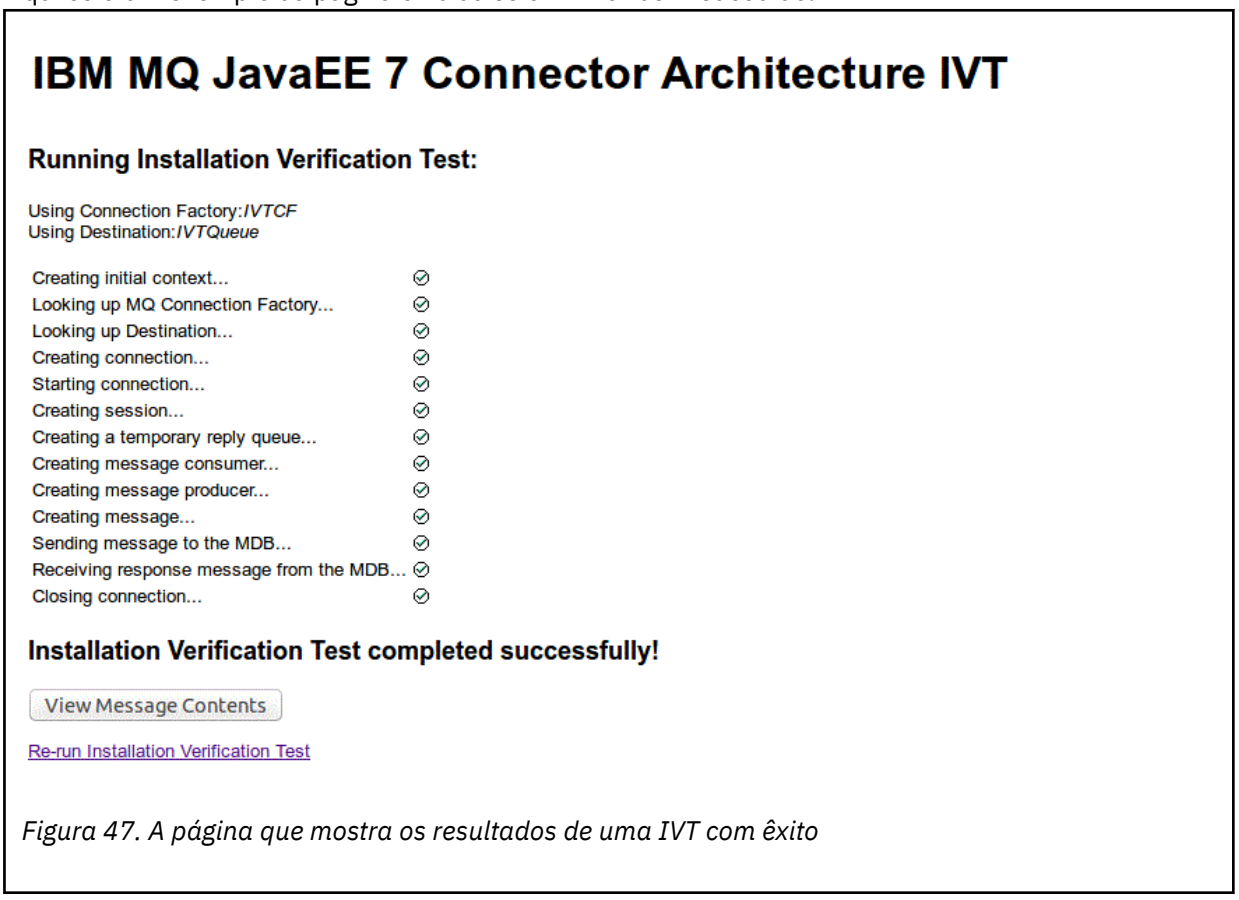


Figura 47. A página que mostra os resultados de uma IVT com êxito

Aqui está um exemplo da página exibida se o IVT falhar. Para obter informações adicionais sobre a causa da falha, clique em **Visualizar rastreio de pilha**.

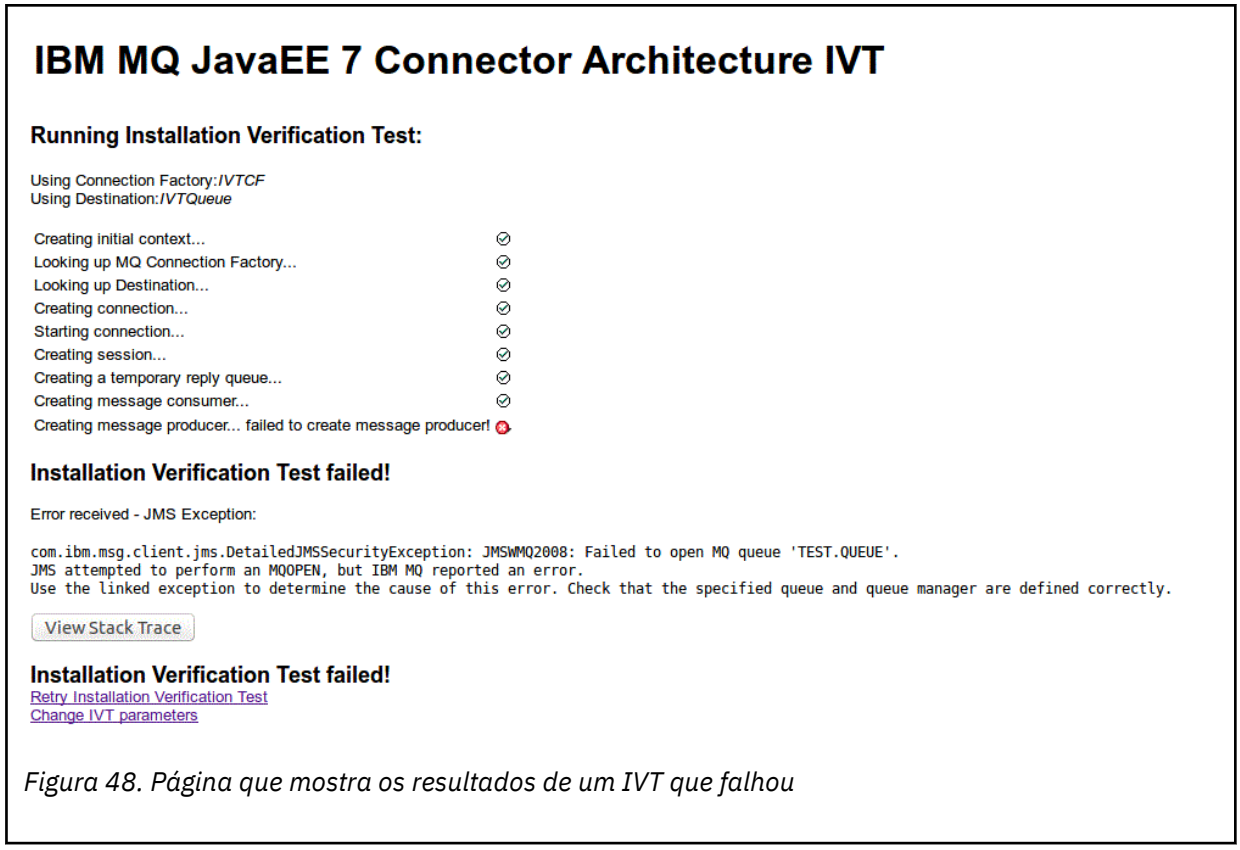


Figura 48. Página que mostra os resultados de um IVT que falhou

Windows Instalando e testando o adaptador de recursos no GlassFish Server

Para instalar o adaptador de recursos do IBM MQ no GlassFish Server em um sistema operacional Windows, deve-se primeiro criar e iniciar um domínio. É possível então implementar e configurar o adaptador de recursos e implementar e executar o aplicativo de teste de verificação de instalação (IVT).

Antes de começar

- Estas instruções são para GlassFish Server versão 4.
- Esta versão do GlassFish Server não suporta Jakarta EE..

Sobre esta tarefa

Esta tarefa presume você tenha um em servidor de aplicativos GlassFish Server em execução e que você esteja familiarizado com as tarefas de administração padrão para isso. Esta tarefa também presume que você tenha uma instalação do IBM MQ em seu sistema local e que você esteja familiarizado com as tarefas de administração padrão.

Nota: Para concluir os passos da tarefa a seguir, deve-se ter uma instalação do IBM MQ em funcionamento com os objetos a seguir configurados:

- Um gerenciador de filas chamado QM, que é iniciado na porta 1414, que usa o canal SYSTEM.DEF.SVRCONN e que se conecta usando o transporte do Cliente.
- Uma fila chamada Q1.

Procedimento

1. Inicie o programa de shell **asadmin** do GlassFish Server.
   a) Abra a linha de comandos Windows e navegue até o diretório *GlassFish*/bin, onde *GlassFish* é o diretório onde a versão 4 do GlassFish Server está instalada.

b) Insira o comando **asadmin** na linha de comandos.

O comando **asadmin** abre um programa shell na linha de comandos que permite criar um novo domínio.

O GlassFish Server versão 4 é iniciado em seu sistema.

2. Criar e, em seguida, inicie um domínio.

a) Use o comando **create-domain**, especificando a porta e o nome de domínio para criar um novo domínio. Digite o seguinte comando na linha de comandos:

```
create-domain --adminport port domain_name
```

em que *port* é o número da porta e *domain_name* é o nome que você deseja que o domínio use.

Nota: O comando **create-domain** tem muitos parâmetros opcionais associados a ele. No entanto, para essa tarefa, é necessário somente o parâmetro `--adminport`. Para obter mais informações, veja a documentação do produto para o GlassFish Server versão 4.

Se a porta especificada estiver em uso, a mensagem a seguir aparecerá:

`A porta para domain_name port está em uso`

Se o nome de domínio especificado estiver em uso, você receberá uma mensagem informando que seu nome especificado já está em uso, bem como uma lista de todos os nomes de domínio que estão atualmente indisponíveis.

b) Quando solicitado a inserir uma senha e um nome de usuário, insira as credenciais para serem usadas para efetuar logon no servidor de aplicativos através de um navegador da web.

Se o comando for concluído com êxito, uma mensagem resumindo a criação do domínio será exibida na linha de comandos, incluindo a mensagem `Command create-domain executed successfully.`

Você criou um domínio com êxito.

c) Inicie seu domínio, inserindo o seguinte comando na linha de comandos:

```
start-domain domain_name
```

em que *domain_name* é o nome de domínio especificado anteriormente.

3. Use um navegador da web para acessar o servidor de aplicativos GlassFish.

a) Na barra de endereço de um navegador da web, insira o seguinte comando:

```
localhost:port
```

em que *port* é a porta que você especificou anteriormente ao criar seu domínio.

O Console GlassFish é exibido.

b) Quando o Console GlassFish foi carregado e você solicitou uma senha e um nome de usuário, insira as credenciais que você especificou na etapa 2b.

4. Faça upload do adaptador de recursos para GlassFish Server 4.

a) Na barra de ferramentas **Tarefas comuns**, selecione o item de menu **Aplicativos** para exibir a página **Aplicativos**.

b) Clique no botão **Implementar** para abrir a página **Implementar aplicativos ou módulos**.

c) Clique no botão **Procurar** , em seguida, navegue para o local do arquivo `wmq.jmsra.rar` . Selecione o arquivo e, em seguida, clique em **OK**

5. Crie um conjunto de conexões.

a) Na barra de ferramentas, em **Recursos**, selecione o item de menu **Conectores**.

b) Em seguida, selecione o item de menu **Conjuntos de conexões do conector** para abrir a página **Conjuntos de conexões do conector**.

c) Clique em **Novo** para abrir a página **Novo conjunto de conexões do conector (Etapa 1 de 2)**.

d) Na página **Novo conjunto de conexões do conector (Etapa 1 de 2)**, insira o nome do conjunto como jms/ivt/IVTCF-Connection-Pool no campo **Nome do conjunto**.

e) No campo **Adaptador de recursos**, selecione `wmq.jmsra`.

f) No campo **definição de conexão** , insira `javax.jms.ConnectionFactory`

g) Selecione **Avançar**, em seguida, selecione **Concluir**.

6. Crie os recursos do conector.

a) Na barra de ferramentas, no menu **Conectores**, selecione a opção **Recurso do conector** para abrir a página **Recursos do conector**.

b) Selecione **Novo** para abrir a página **Novo recurso do conector**.

c) No campo **Nome JNDI**, insira `IVTCF`.

d) No campo **Nome do conjunto**, insira `jms/ivt/IVTCF-Connection-Pool`.

e) Deixe todos os outros campos vazios.

f) Para cada um dos pares de propriedade/valor a seguir, clique em **Incluir propriedade** e insira o nome da propriedade e o valor conforme mostrado no exemplo a seguir:

- name: host; value: localhost
- name: port; value 1414
- name: channel; value: SYSTEM.DEF.SVRCONN
- name: queueManager; value: QM
- name: transportType; value: CLIENT

Nota: Certifique-se de usar os valores corretos para suas próprias definições de configuração, que poderão ser diferentes das mostradas neste exemplo.

g) Na barra de ferramentas, em **Conectores**, selecione o item de menu **Recursos de objetos de administrador** para abrir a página **Recursos de objetos de administrador**.

h) Na página **Recursos de objetos administrativos**, clique em **Novo** para abrir a página **Novo recurso de objetos administrativos**.

i) No campo **Nome JNDI**, insira `IVTQueue`.

j) No campo **Adaptador de recursos**, insira `wmq.jmsra`.

k) No campo **Tipo de recurso**, insira `javax.jms.Queue`.

l) Deixe o campo **Nome da classe** como está.

m) Para cada um dos pares de propriedade/valor a seguir, clique em **Incluir propriedade** e insira o nome da propriedade e o valor conforme mostrado no exemplo a seguir:

- name: name; value: IVTQueue
- name: baseQueueManagerName; value QM
- name: baseQueueName; value: Q1

Nota: Certifique-se de usar os valores corretos para suas próprias definições de configuração, que poderão ser diferentes das mostradas neste exemplo.

n) Clique em **OK**.

o) Marque a caixa de seleção **Ativado** e, em seguida, clique em **Ativar**.

7. Implemente o arquivo EAR `wmq.jmsra.ivt.ear` no GlassFish Server.

a) Clique na opção **Aplicativos** na barra de ferramentas para exibir a página **Aplicativos**.

b) Clique em **Implementar** para incluir o aplicativo IVT.

c) No campo **Local** navegue para, e selecione o `wmq.jmsra.ivt.ear`.

d) No campo **Servidores virtuais**, selecione **servidor** e, em seguida, clique em **OK**.

8. Ative o programa IVT.

a) Clique na opção **Aplicativos** na barra de ferramentas para exibir a página **Aplicativos**.

b) Clique em `wmq.jmsra.ivt` na tabela Aplicativos Implementados.

c) Clique no botão **Ativar** na tabela Módulos e Componentes.

d) Selecione o http: link.

e) Clique em **Executar IVT**.

Você ativou o programa IVT e se você for bem-sucedido, a seguinte saída será exibida:

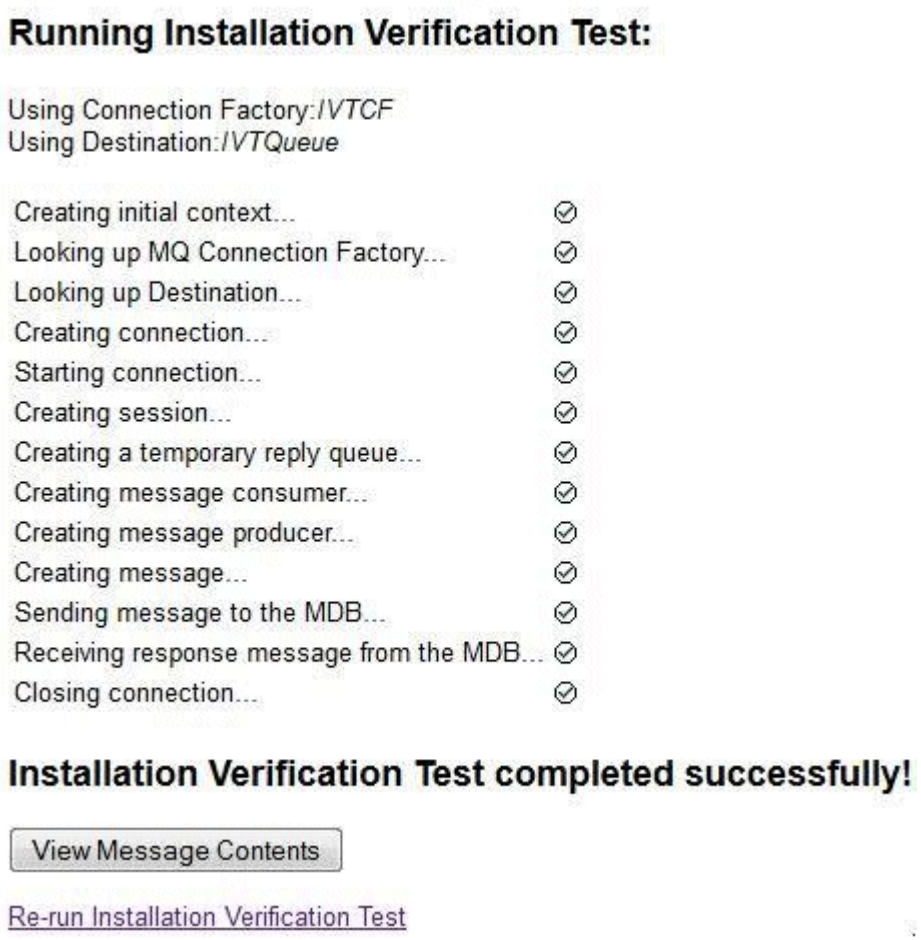


Figura 49. Saída de IVT bem-sucedida

Instalando e testando o adaptador de recursos no Wildfly

Se você está instalando o adaptador de recursos do IBM MQ no Wildfly V10, deve-se primeiro fazer algumas mudanças no arquivo de configuração para incluir uma definição de subsistema para o adaptador de recursos do IBM MQ. É possível, então, implementar o adaptador de recursos e testá-lo instalando e executando o aplicativo de teste de verificação de instalação (IVT).

Antes de começar

- Essas instruções são para o Wildfly V10.
- Esta versão do WildFly não suporta o Jakarta EE

Sobre esta tarefa

Essa tarefa presume que você tenha um servidor de aplicativos WildFly em execução e que esteja familiarizado com as tarefas de administração padrão para isso. Essa tarefa também presume que você tenha uma instalação do IBM MQ e que esteja familiarizado com as tarefas de administração padrão.

Procedimento

1. Crie um gerenciador de filas do IBM MQ chamado ExampleQM e configure-o conforme descrito em “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081.

   Ao configurar o gerenciador de filas, observe os pontos a seguir:

   - O listener deve ser iniciado na porta 1414.
   - O canal a ser usado é chamado SYSTEM.DEF.SVRCONN.
   - A fila usada pelo aplicativo IVT é denominada TEST.QUEUE.

   Também é necessário conceder autoridade DSP e PUT à fila modelo SYSTEM.DEFAULT.MODEL.QUEUE para que esse aplicativo possa criar uma fila de resposta provisória.

2. Edite o arquivo de configuração *WildFly_Home*/standalone/configuration/standalone-full.xml e inclua o subsistema a seguir:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
```

```
    </resource-adapters>
</subsystem>
```

3. Implemente o adaptador de recursos para o servidor copiando o arquivo `wmq.jmsra.rar` no diretório *WildFly_Home*`/standalone/deployments`.
4. Implemente o aplicativo IVT copiando o arquivo `wmq.jmsra.ivt.ear` no diretório *WildFly_Home*`/standalone/deployments`.
5. Inicie o servidor de aplicativos abrindo um prompt de comandos, navegando para o diretório *WildFly_Home*`/bin` e executando o comando:

```
standalone.bat -c standalone-full.xml
```

6. Execute o aplicativo IVT.

   Para obter informações adicionais, consulte "Verificando a instalação do adaptador de recursos" na página 501. Para Wildfly, a URL padrão é `http://localhost:8080/WMQ_IVT/`.

Usando o IBM MQ e o WebSphere Application Server juntos

Através do provedor de mensagens do IBM MQ em aplicativos de mensagens no WebSphere Application Server, Java Message Service (JMS) podem usar o seu sistema IBM MQ como um provedor externo dos recursos de mensagens do JMS.

Sobre esta tarefa

Aplicativos que são gravados em Java e estão em execução em WebSphere Application Server podem usar a especificação Java Message Service (JMS) para executar o sistema de mensagens. O sistema de mensagens neste ambiente pode ser fornecido por um gerenciador de filas do IBM MQ.

Um benefício de usar um gerenciador de filas do IBM MQ é que a conexão de aplicativos JMS pode participar totalmente da funcionalidade de uma rede do IBM MQ, o que permite que os aplicativos troquem mensagens com gerenciadores de filas que estiverem em execução em uma variedade de plataformas.

Os aplicativos podem usar o *transporte de cliente* ou *transporte de ligações* para o objeto connection factory da fila. Para o transporte de ligações, o gerenciador de filas deve existir localmente para o aplicativo que requer uma conexão.

Por padrão, mensagens do JMS mantidas em filas do IBM MQ usam um cabeçalho MQRFH2 para manter algumas das informações do cabeçalho de mensagem do JMS. Muitos aplicativos legados do IBM MQ não podem processar mensagens com esses cabeçalhos e requerem seus próprios cabeçalhos característicos, por exemplo, os aplicativos MQCIH for CICS Bridge ou MQWIH for IBM MQ Workflow. Para obter mais informações sobre essas considerações especiais, consulte as mensagens de Mapeamento JMS em IBM MQ mensagens.

Tarefas relacionadas

Configurando os recursos do JMS no WebSphere Application Server

Configurando o servidor de aplicativos para usar o nível de manutenção mais recente do adaptador de recursos

Usando o WebSphere Application Server com o IBM MQ

O IBM MQ e o IBM MQ for z/OS podem ser usados com ou como uma alternativa para o provedor de sistemas de mensagens padrão que está incluído com o WebSphere Application Server.

O provedor de sistemas de mensagens do IBM MQ é instalado como parte do WebSphere Application Server. Isso inclui uma versão do adaptador de recursos do IBM MQ e a funcionalidade do IBM MQ Extended Transactional Client, que permite que o gerenciador de filas participe de transações XA gerenciadas pelo servidor de aplicativos. Usando o adaptador de recursos, os beans acionados por mensagens podem ser configurados para usar as especificações de ativação ou as portas listener.

Para que o servidor de aplicativos seja suportado, o programa de teste de verificação de instalação do adaptador de recursos do IBM MQ precisa ser implementado no servidor de aplicativos e executado com sucesso. Após o programa de teste de verificação de instalação do adaptador de recursos do IBM MQ ter sido executado com sucesso, o adaptador de recursos do IBM MQ poderá se conectar a qualquer gerenciador de filas suportado do IBM MQ.

Conexões do JMS do WebSphere Application Server ao IBM MQ

Antes de considerar os níveis do IBM MQ que podem ser usados com o WebSphere Application Server, é importante entender como os aplicativos Java Message Service (JMS) em execução dentro do servidor de aplicativos podem se conectar aos gerenciadores de filas do IBM MQ.

Os aplicativos do JMS que precisam acessar os recursos de um gerenciador de filas do IBM MQ podem fazer isso usando um dos tipos de transporte a seguir:

BINDINGS
Esse transporte pode ser utilizado quando o servidor de aplicativos e o gerenciador de filas são instalados na mesma máquina e imagem do sistema operacional. Ao utilizar o modo BINDINGS, toda a comunicação entre os dois produtos é feita utilizando a Comunicação Interprocessual (IPC).

O provedor de sistemas de mensagens do IBM MQ não inclui as bibliotecas nativas necessárias para conexão com um gerenciador de filas do IBM MQ no modo BINDINGS. Para utilizar uma conexão do modo BINDINGS, o IBM MQ deve ser instalado na mesma máquina que o servidor de aplicativos e o caminho da biblioteca nativa do adaptador de recursos deve ser configurado para apontar para o diretório IBM MQ no qual essas bibliotecas estão localizadas. Para obter mais informações, consulte a documentação do produto WebSphere Application Server:

- Para o WebSphere Application Server traditional, consulte Configurando o provedor de sistemas de mensagens do IBM MQ com bibliotecas nativas.
- Para WebSphere Liberty, consulte Implementando aplicativos JMS no Liberty para usar o provedor de sistemas de mensagens IBM MQ.

z/OS No z/OS, se você deseja conectar um connection factory do WebSphere Application Server a um gerenciador de filas do IBM MQ no modo de ligações, as bibliotecas IBM MQ corretas devem ser especificadas na concatenação STEPLIB do WebSphere Application Server. Para obter mais informações, consulte Bibliotecas do IBM MQ e o WebSphere Application Server for z/OS STEPLIB na documentação do produto WebSphere Application Server.

CLIENTE
O transporte cliente usa TCP/IP para se comunicar entre o WebSphere Application Server e o IBM MQ. Além de ser usado quando o servidor de aplicativos e o gerenciador de filas estão localizados em máquinas diferentes, o modo CLIENT também pode ser usado quando os dois produtos são instalados na mesma máquina e imagem do sistema operacional.

Os aplicativos JMS também podem especificar um tipo de transporte de BINDINGS_THEN_CLIENT. Quando esse tipo de transporte for usado, o aplicativo tentará inicialmente se conectar ao gerenciador de filas usando o modo BINDINGS; se ele não puder fazer isso, ele tentará o transporte CLIENT.

Como localizar qual versão do adaptador de recursos do IBM MQ está instalada dentro do WebSphere Application Server

Para obter informações sobre qual versão do adaptador de recursos do IBM MQ está instalada dentro do WebSphere Application Server, consulte a nota técnica Qual versão do WebSphere MQ Resource Adapter (RA) é fornecida com o WebSphere Application Server?.

É possível usar os comandos Jython e JACL a seguir para determinar o nível do adaptador de recurso que o WebSphere Application Server está atualmente utilizando:

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
```

```
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

Nota: É necessário clicar em **Retornar** duas vezes depois de inserir esse comando para executá-lo.

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

Atualizando o adaptador de recursos

As atualizações no adaptador de recursos do IBM MQ que está instalado com o servidor de aplicativos são incluídas nos Fix Packs do WebSphere Application Server. Atualizando o adaptador de recursos do IBM MQ usando **Atualizar adaptador de recursos ...** no WebSphere Application Server Administrative Console não é recomendado, pois fazer isso significará que as atualizações fornecidas no WebSphere Application Server Fix Packs não terão efeito

Variável MQ_INSTALL_ROOT

A partir do WebSphere Application Server 7.0, o MQ_INSTALL_ROOT é usado apenas para localizar bibliotecas nativas e é substituído por qualquer caminho de biblioteca nativa configurado no adaptador de recursos.

Conectando-se do WebSphere Application Server ao IBM MQ

Atenção:

1. Qualquer versão suportada do WebSphere Application Server pode usar o adaptador de recursos do IBM MQ que é empacotado com ele para se conectar a qualquer versão suportada do IBM MQ.
2. Caso o modo de ligações seja usado, determinadas bibliotecas no WebSphere Application Server precisam corresponder à versão do gerenciador de filas ao qual ele está se conectando:
   - O WebSphere Application Server deve ser configurado para carregar as bibliotecas nativas fornecidas com o IBM MQ 9.4. Consulte"Configurando as bibliotecas do Java Native Interface (JNI)" na página 97 para obter mais informações.
   - z/OS No z/OS, deve-se especificar as bibliotecas IBM MQ corretas na concatenação WebSphere Application Server STEPLIB.

     Consulte as bibliotecas IBM MQ e o WebSphere Application Server para z/OS STEPLIB para obter detalhes das bibliotecas IBM MQ necessárias.

     Se você tiver bibliotecas para uma versão do IBM MQ em LINKLIST (LINKLST), será possível conectar-se a uma versão diferente do IBM MQ substituindo as bibliotecas pela STEPLIB.
3. A versão do adaptador de recursos do IBM MQ é independente das versões de bibliotecas nativas (compartilhadas) fornecidas pela instalação do gerenciador de filas.

   Por exemplo, o WebSphere Application Server 8.5 com um Adaptador de recursos do IBM MQ 8.0 ainda pode gerenciar uma conexão de ligações com um gerenciador de filas do IBM MQ 9.0 usando as bibliotecas nativas do IBM MQ 9.0.

Para obter informações adicionais, consulte "Instrução de suporte do adaptador de recursos do IBM MQ" na página 446.

Os tipos de transporte BINDINGS e CLIENT podem ser usados para se conectar ao IBM MQ de qualquer versão do WebSphere Application Server. Para o tipo de transporte BINDINGS, as restrições a seguir se aplicam:

- IBM MQ deve ser instalado na mesma máquina que o servidor de aplicativos.
- O WebSphere Application Server deve ser configurado para carregar as bibliotecas nativas fornecidas com o IBM MQ.
- z/OS No z/OS, se você deseja conectar um connection factory do WebSphere Application Server a um gerenciador de filas do IBM MQ no modo de ligações, as bibliotecas IBM MQ corretas devem ser especificadas na concatenação STEPLIB do WebSphere Application Server.

A tabela a seguir mostra as versões do WebSphere Application Server em que cada versão do adaptador de recursos IBM MQ é suportada para execução.

Tabela 70. Mapeando WebSphere Application Server versões para IBM MQ versões do adaptador de recursos.

Versão do adaptador de recursos do IBM MQ	**Em qual versão do WebSphere Application Server esta versão do adaptador de recursos pode ser executada?**
IBM MQ 9.0 e mais recente	O adaptador de recursos pode ser executado em: • Qualquer versão compatível do Java EE 7 do WebSphere Liberty. • WebSphere Application Server traditional 9.0
IBM MQ 8.0	O adaptador de recursos pode ser executado em qualquer versão compatível com o Java EE 7 do WebSphere Liberty O adaptador de recursos do IBM MQ 8.0 não é suportado para execução no WebSphere Application Server traditional. O adaptador de recursos já instalado no WebSphere Application Server traditional deve ser usado para se conectar aos gerenciadores de filas do IBM MQ 8.0.

Conceitos relacionados

"Instrução de suporte do adaptador de recursos do IBM MQ" na página 446

O adaptador de recursos do IBM MQ que deve ser usado para comunicação entre um aplicativo e um gerenciador de filas depende se você está usando a API Jakarta Messaging 3.0 ou a API JMS 2.0 .

Informações relacionadas

Requisitos do Sistema para IBM MQ

Determinando o número de conexões TCP/IP criadas do WebSphere Application Server para o IBM MQ

Usando o recurso de compartilhamento de conversas, várias conversas podem compartilhar instâncias do canal MQI; isso também é conhecido como uma conexão TCP/IP.

Sobre esta tarefa

Os aplicativos em execução no WebSphere Application Server 7 e no WebSphere Application Server 8, que usam o modo normal do provedor de sistemas de mensagens do IBM MQ usarão esse recurso automaticamente. Isso significa que vários aplicativos em execução na mesma instância do servidor de aplicativos, que se conectam ao mesmo gerenciador de filas do IBM MQ, são capazes de compartilhar a mesma instância do canal.

O número de conversas que podem ser compartilhadas em uma única instância de canal é determinado pela propriedade de canal do IBM MQ, **SHARECNV**. O valor padrão dessa propriedade para os canais de conexão do servidor é 10.

Ao olhar o número de conversas criadas pelo WebSphere Application Server 7 e WebSphere Application Server 8, é possível determinar o número de instâncias do canal criadas.

Para obter mais informações sobre o modo do provedor de sistemas de mensagens do IBM MQ, consulte Modo normal PROVIDERVERSION.

Conceitos relacionados

Usando compartilhando conversas
Em um ambiente no qual conversas de compartilhamento são permitidas, as conversas podem compartilhar uma instância do canal MQI.

"Compartilhando uma conexão TCP/IP em IBM MQ classes for JMS" na página 322
Várias instâncias de um canal MQI podem ser feitas para compartilhar uma única conexão TCP/IP.

Connection factories do JMS

Os aplicativos em execução dentro do WebSphere Application Server, que usam um connection factory do provedor de sistemas de mensagens do IBM MQ para criar conexões e sessões, têm conversas ativas para cada conexão JMS criada por meio do connection factory e para cada sessão do JMS criada por meio de uma conexão JMS.

Uma conversa para cada conexão JMS que foi criada por meio do connection factory

Cada connection factory do JMS possui um conjunto de conexões associado, dividido em duas seções, o conjunto livre e o conjunto ativo. Ambos os conjuntos estão inicialmente vazios.

Quando um aplicativo cria uma conexão JMS por meio de um connection factory, o WebSphere Application Server verifica se há uma conexão JMS no conjunto livre. Se houver, ela será movida para o conjunto ativo e fornecida para o aplicativo. Caso contrário, uma nova conexão JMS será criada, colocada no conjunto ativo e retornada para o aplicativo. O número máximo de conexões que podem ter sido criadas a partir de um connection factory é especificado pela propriedade do conjunto de conexões do connection factory **Maximum connections** O valor padrão para essa propriedade é 10.

Depois que um aplicativo tiver sido concluído com uma conexão JMS e a tiver encerrado, a conexão será movida do conjunto ativo para o conjunto livre, no qual ficará disponível para reutilização. A propriedade **Unused timeout** do conjunto de conexões define quanto tempo uma conexão JMS pode permanecer no conjunto livre antes de ser desconectada. O valor padrão para essa propriedade é 1.800 segundos (30 minutos).

Quando uma conexão JMS é criada pela primeira vez, uma conversa entre o WebSphere Application Server e o IBM MQ é iniciada. A conversa permanece ativa até que a conexão seja encerrada quando o valor da propriedade **Unused timeout** para o conjunto livre for excedido.

Uma conversa para cada sessão JMS que tenha sido criada por meio de uma conexão JMS

Cada conexão JMS criada por meio de um connection factory do provedor de sistemas de mensagens do IBM MQ possui um conjunto de sessões associado do JMS. Os conjuntos de sessões funcionam da mesma maneira que os conjuntos de conexões. O número máximo de JMS Sessões que podem ser criadas a partir de uma única conexão JMS é determinado pela propriedade do conjunto de sessões do connection factory **Maximum connections** O valor padrão desta propriedade é 10.

Uma conversa inicia quando uma sessão JMS é criada pela primeira vez. A conversa permanece ativa até que a sessão JMS seja encerrada porque ela permaneceu no conjunto livre por mais tempo do que o valor da propriedade **Unused timeout** para o conjunto de sessões.

Calculando um valor para a propriedade SHARECNV

É possível calcular o número máximo de conversas de um único connection factory para o IBM MQ usando a fórmula a seguir:

```
Maximum number of conversations =
```

```
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

O número de instâncias do canal que serão criadas para permitir que esse número de conversas ocorra pode ser trabalhado usando o cálculo a seguir:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Qualquer restante desse cálculo pode ser arredondado para cima.

Para um connection factory simples que está usando o valor padrão para as propriedades do conjunto de conexões **Maximum connections** e do conjunto de sessão **Maximum connections** , o número máximo de conversações que podem existir entre WebSphere Application Server e IBM MQ para esse connection factory é:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Por exemplo:

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

Se esse connection factory estiver se conectando ao IBM MQ usando um canal que tenha a propriedade **SHARECNV** configurada como 10, o número máximo de instâncias do canal que serão criadas para esse connection factory será:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

Por exemplo:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

Especificações de ativação

Os aplicativos bean acionados por mensagens, que são configurados para usar uma especificação de ativação, possuem conversas ativas para a especificação de ativação para monitorar um destino JMS e para cada sessão do servidor usada para executar uma instância de bean acionado por mensagens para processar mensagens.

As conversas a seguir estão ativas para aplicativos bean acionados por mensagens que são configurados para usar uma especificação de ativação:

- Uma conversa para a especificação de ativação monitorar um destino JMS para mensagens adequadas. Essa conversa é iniciada assim que a especificação de ativação é iniciada e permanece ativa até que a especificação de ativação seja interrompida.
- Uma conversa para cada sessão do servidor usada para executar uma instância de bean acionado por mensagens para processar mensagens.

A propriedade avançada da especificação de ativação **Maximum server sessions** especifica o número máximo de sessões do servidor que podem estar ativas a qualquer momento para uma determinada especificação de ativação Essa propriedade tem o valor padrão 10. As sessões do servidor são criadas conforme são necessárias e serão encerradas se tiverem ficado inativas pelo período de tempo especificado pela propriedade avançada de especificação de ativação **Server session pool timeout**. O valor padrão para essa propriedade é de 300.000 milissegundos (5 minutos).

As conversas iniciam quando uma sessão do servidor é criada e são interrompidas quando a especificação de ativação é interrompida ou quando uma sessão do servidor atinge o tempo limite.

Isso significa que o número máximo de conversas de uma única especificação de ativação para o IBM MQ pode ser calculado usando a fórmula a seguir:

```
Maximum number of conversations = Maximum server sessions + 1
```

O número de instâncias de canal criadas para permitir que esse número de conversas ocorra pode ser localizado usando o cálculo a seguir:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Qualquer restante desse cálculo pode ser arredondado para cima.

Para uma especificação de ativação simples, que usa o valor padrão para a propriedade **Maximum server sessions** , o número máximo de conversações que podem existir entre WebSphere Application Server e IBM MQ para essa especificação de ativação é calculado como:

```
Maximum number of conversations = Maximum server sessions + 1
```

Por exemplo:

```
    = 10 + 1
    = 11
```

Se essa especificação de ativação estiver se conectando ao IBM MQ usando um canal que tem a propriedade **SHARECNV** configurada como 10, o número de instâncias de canal criadas será calculado como:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Por exemplo:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Portas do listener em execução no modo do Application Server Facilities (ASF)

As portas do listener em execução no modo ASF usadas pelos aplicativos bean acionados por mensagens criam conversas para cada sessão do servidor. Uma monitora um destino para mensagens adequadas e a outra executa uma instância de bean acionado por mensagens para processar mensagens. O número de conversas para cada porta do listener pode ser calculado por meio de um número máximo de sessões.

Por padrão, as portas do listener serão executadas no modo ASF como parte da especificação 1.1 que define o mecanismo que os servidores de aplicativos devem usar para detectar mensagens e entregá-las aos beans acionados por mensagens para processamento. Aplicativos bean acionados por mensagens configurados para usar portas do listener nesse modo de operação padrão criam conversas:

Uma conversa para a porta do listener para monitorar um destino para mensagens adequadas
As portas do listener são configuradas para usar um connection factory do JMS. Quando uma porta do listener é iniciada, é feita uma solicitação por uma conexão JMS por meio do conjunto livre de connection factories. A conexão será retornada para o conjunto livre quando a porta do listener for interrompida. Para obter mais informações sobre como o conjunto de conexões é usado e como isso afeta o número de conversas para o IBM MQ, consulte “Connection factories do JMS” na página 513.

Uma conversa para cada sessão do servidor usada para executar uma instância de bean acionado por mensagens para processar mensagens

A propriedade da porta listener **Maximum sessions** especifica o número máximo de sessões do servidor que podem estar ativas a qualquer momento para uma determinada porta listener. Essa propriedade tem o valor padrão 10. As sessões do servidor são criadas conforme são necessárias e usam sessões JMS tiradas do conjunto de sessões associado à conexão JMS que a porta do listener está usando.

Se uma sessão do servidor tiver ficado inativa pelo período de tempo especificado pela propriedade customizada **SERVER.SESSION.POOL.UNUSED.TIMEOUT** do Serviço de listener de mensagens, a sessão será encerrada e a sessão JMS usada será retornada para o conjunto livre do conjunto de sessões. A sessão JMS permanecerá no conjunto livre do conjunto de sessões até que seja necessária ou será encerrada porque está inativa no conjunto livre por mais tempo que o valor da propriedade **Unused timeout** do conjunto de sessões.

Para obter mais informações sobre como o conjunto de sessões é usado e como as conversas entre o WebSphere Application Server e o IBM MQ são gerenciadas, consulte “Connection factories do JMS” na página 513.

Para obter mais informações sobre a propriedade customizada **SERVER.SESSION.POOL.UNUSED.TIMEOUT** do Serviço de listener de mensagens, consulte Monitorando conjuntos de sessões do servidor para portas do listener na documentação do produto WebSphere Application Server.

Calculando o número máximo de conversas de uma única porta do listener para o IBM MQ

É possível calcular o número máximo de conversas de uma única porta do listener para o IBM MQ usando a fórmula a seguir:

```
Maximum number of conversations = Maximum sessions + 1
```

O número de instâncias do canal que serão criadas para permitir que esse número de conversas ocorra pode ser trabalhado usando o cálculo a seguir:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Qualquer restante desse cálculo pode ser arredondado para cima.

Para uma porta listener simples que está usando o valor padrão para a propriedade **Maximum sessions** , o número máximo de conversas que podem existir entre WebSphere Application Server e IBM MQ para essa porta listener é calculado como:

```
Maximum number of conversations = Maximum sessions + 1
```

Por exemplo:

```
    = 10 + 1
    = 11
```

Se essa porta do listener estiver se conectando ao IBM MQ usando um canal que tenha a propriedade **SHARECNV** configurada como 10, o número de instâncias do canal que serão criadas será calculado como:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Por exemplo:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Portas do listener em execução no modo não Application Server Facilities (não ASF)

As portas do listener em execução no modo não ASF podem ser configuradas para monitorar o destino da fila e o destino do tópico usando Sessões do servidor. As sessões do servidor podem ter várias conversas, cujo número máximo pode ser calculado em cada caso.

As portas do listener podem ser configuradas para execução no modo não ASF que muda a maneira como as portas do listener monitoram os destinos JMS. Os aplicativos bean acionados por mensagens, usando portas do listener no modo de operação não ASF, criam uma conversa para cada sessão do servidor usada para executar uma instância de bean acionado por mensagens para processar mensagens. A propriedade **maximum sessions** da porta do listener especifica o número máximo de sessões do servidor que podem estar ativas em um determinado momento para uma determinada porta do listener. O valor padrão para essa propriedade é 10.

Ao executar no modo não ASF, uma porta do listener que monitora um destino de fila criará automaticamente o número de Sessões do servidor especificadas pela propriedade de porta do listener **Máximo de sessões**. Todas essas Sessões do servidor usam as Sessões JMS tiradas do conjunto de sessões associado à Conexão JMS que a porta do listener está usando e monitoram continuamente um Destino JMS para mensagens adequadas.

Se a porta do listener estiver configurada para monitorar um destino de tópico, o valor de **Máximo de sessões** será ignorado e uma única sessão do Servidor será usada.

As Sessões do servidor usadas por uma porta do listener em execução no modo não ASF permanecem ativas até que a porta do listener seja interrompida, ponto em que as Sessões JMS que foram usadas são retornadas para o Conjunto livre do conjunto de sessões da Conexão JMS que a porta do listener estava usando.

Para obter mais informações sobre como o conjunto de sessões é usado e como as conversas entre o WebSphere Application Server e o IBM MQ são gerenciadas, consulte “Connection factories do JMS” na página 513.

Para obter mais informações sobre o modo de operação ASF e não ASF com o WebSphere Application Servere como configurar Portas do listener para usar o modo não ASF, consulte Processamento de mensagens no modo ASF e no modo não ASF.

Calculando o número máximo de conversas ao monitorar um destino de fila

O número máximo de conversas de uma única porta do listener, em execução no modo não ASF e monitorando um destino de fila para o IBM MQ pode ser calculado usando a seguinte fórmula:

```
Maximum number of conversations = Maximum sessions
```

O número de instâncias do canal que serão criadas para permitir que esse número de conversas ocorra pode ser localizado usando o cálculo a seguir:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Qualquer restante desse cálculo pode ser arredondado para cima.

Para uma porta de listener simples em execução no modo não ASF que está usando o valor padrão para a propriedade **Máximo de sessões** e monitorando um destino de fila, o número máximo de conversas que podem existir entre o WebSphere Application Server e o IBM MQ para essa porta do listener é:

```
Maximum number of conversations = Maximum sessions
```

Por exemplo:

```
= 10
```

Se essa porta listener estiver se conectando ao IBM MQ usando um canal que tenha a propriedade **SHARECNV** configurada como 10, então, o número de instâncias do canal que são criadas será calculado como:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Por exemplo:

```
= 10 / 10
= 1
```

Calculando o número máximo de conversas ao monitorar um destino de tópico

Para uma porta do listener em execução no modo não ASF e configurada para monitorar um destino de tópico, o número de conversas da porta do listener para o IBM MQ é:

```
Maximum number of conversations = 1
```

O número de instâncias do canal que serão criadas para permitir que esse número de conversas ocorra pode ser localizado usando o cálculo a seguir:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Qualquer restante desse cálculo pode ser arredondado para cima.

Para uma porta de listener simples em execução no modo não ASF que está usando o valor padrão para a propriedade **Máximo de sessões** e monitorando um destino de tópico, o número máximo de conversas que podem existir entre o WebSphere Application Server e o IBM MQ para essa porta do listener é:

```
Maximum number of conversations = Maximum sessions
```

Por exemplo:

```
= 10
```

Se essa porta listener estiver se conectando ao IBM MQ usando um canal que tenha a propriedade **SHARECNV** configurada como 10, então, o número de instâncias do canal que são criadas será calculado como:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Por exemplo:

```
= 10 / 10
= 1
```

Configurando aliases de autenticação para proteger a conexão do WebSphere Application Server para o IBM MQ

Os aliases de autenticação são mapeados para uma combinação de nome do usuário e senha que pode ser usada para proteger a conexão do WebSphere Application Server para o IBM MQ. É possível configurar um connection factory com um alias de autenticação.

Usando Aliases de Autenticação com Aplicativos Corporativos

Quando um aplicativo corporativo em execução no WebSphere Application Server tenta criar uma conexão JMS para o IBM MQ, o aplicativo consulta uma definição de connection factory do provedor de sistemas de mensagens do IBM MQ por meio do repositório Java Naming Directory Interface (JNDI) do servidor de aplicativos.

Quando a definição de connection factory do provedor de sistemas de mensagens do IBM MQ está localizada no repositório JNDI do servidor de aplicativos, um dos métodos a seguir é chamado:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Se o connection factory tiver sido configurado com um alias de autenticação J2C definido, o nome do usuário e a senha no alias de autenticação poderão fluir para o IBM MQ quando o connection factory for usado para criar uma conexão.

Connection Factories e Aliases de Autenticação

Os connection factories do provedor de sistemas de mensagens do IBM MQ contêm informações sobre como conectar-se a gerenciadores de filas do IBM MQ. Os aplicativos corporativos em execução no WebSphere Application Server podem usar os connection factories para criar conexões JMS para o IBM MQ.

O WebSphere Application Server armazena definições de connection factories em um repositório que pode ser acessado usando o JNDI. Quando um connection factory é criado, o connection factory recebe um nome JNDI para identificá-lo exclusivamente no escopo do servidor de aplicativos (no escopo da Célula, do Nó ou do Servidor) no qual ele foi definido.

Por exemplo, uma connection factory do provedor de sistemas de mensagens IBM MQ definida no escopo da célula WebSphere Application Server contém informações sobre como se conectar ao gerenciador de filas (`myQM`) usando o protocolo de transporte BINDINGS. Essa connection factory é dada o JNDI name `jms/myCF` para identificá-la de forma exclusiva.

Os connection factories também podem ser configurados para usar um alias de autenticação. Os aliases de autenticação são mapeados para uma combinação de nome do usuário e senha. Dependendo de como o connection factory é usado, o nome do usuário e a senha no alias de autenticação podem, ou não, fluir para o IBM MQ quando a conexão JMS é criada.

Importante: Antes do IBM MQ 8.0, o Gerenciador de Autoridade de Objeto (OAM) padrão do IBM MQ executava uma verificação de autorização somente para assegurar que o nome do usuário passado para o IBM MQ, quando uma conexão era feita, tinha a autoridade para acessar o gerenciador de filas.

Nenhuma verificação era feita para validar a senha especificada. Para executar uma verificação de autenticação e validar que o identificador de usuário e a senha correspondiam, era necessário gravar uma saída de segurança de canal do IBM MQ. Detalhes sobre como fazer isso podem ser encontrados em “Programas de saída de segurança do canal” na página 981.

No IBM MQ 8.0, o gerenciador de filas verifica a senha além do nome do usuário.

Usando o connection factory

Os tópicos a seguir contêm informações sobre como usar o connection factory usando consultas diretas e indiretas:

- “Usando o connection factory por meio de uma consulta direta” na página 523
- “Usando o connection factory por meio de uma consulta indireta” na página 524

Usando o transporte CLIENT

Os connection factories que são configurados para usar o transporte CLIENT devem especificar qual canal de conexão do servidor (SVRCONN) do IBM MQ eles usarão para se conectar ao gerenciador de filas.

Se a propriedade de identificador de usuário do agente de canal do IBM MQ (MCAUSER) permanecer em branco para o canal que o connection factory foi configurado para usar, o connection factory poderá ser usado com uma consulta direta ou indireta.

Se a propriedade MCAUSER for configurada para um identificador de usuário, esse identificador de usuário será passado para o IBM MQ quando o connection factory for usado para criar uma conexão com o IBM MQ, independentemente se o aplicativo corporativo estiver usando uma consulta direta ou indireta.

Tabelas de resumo

As tabelas a seguir resumem quais identificadores de usuário fluem para o IBM MQ quando o transporte BINDINGS e o transporte CLIENT, respectivamente, são usados:

Tabela 71. modo BINDINGS

Configuração	**Chamadas do aplicativo `ConnectionFactory.createConnection()`**	**Chamadas do aplicativo `ConnectionFactory.createConnection(String username, String password)`**
O descritor de implementação do aplicativo não contém uma Referência de Recurso para o connection factory	O identificador do usuário para o processo do servidor de aplicativos é transmitido para o IBM MQ.	O identificador do usuário e a senha que foram passados no método `ConnectionFactory.createConnection(String username, String password)` são transmitidos para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de Recurso para o connection factory e a propriedade **`res-auth`** é configurada para "Application"	O identificador do usuário para o processo do servidor de aplicativos é transmitido para o IBM MQ.	O identificador do usuário e a senha que foram passados no método `ConnectionFactory.createConnection(String username, String password)` são transmitidos para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de Recurso para o connection factory e a propriedade **`res-auth`** é configurada para "Container"	O identificador do usuário e a senha especificados no alias de autenticação para a connection factory são transmitidos para o IBM MQ.	O identificador do usuário e a senha especificados no alias de autenticação para a connection factory são transmitidos para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de Recurso para o connection factory que tem a propriedade **`res-auth`** configurada como "Container" e o aplicativo foi configurado com um alias de autenticação	O identificador do usuário e a senha especificados no alias de autenticação que o aplicativo foi configurado para usar são transmitidos para o IBM MQ.	O identificador do usuário e a senha especificados no alias de autenticação que o aplicativo foi configurado para usar são transmitidos para o IBM MQ.

Tabela 72. Modo CLIENT

Configuração	**Chamadas do aplicativo `ConnectionFactory.createConnection()`**	**Chamadas do aplicativo `ConnectionFactory.createConnection(String username, String password)`**
O descritor de implementação do aplicativo não contém uma Referência de recurso para o connection factory, que está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER não configurada	O identificador do usuário para o processo do servidor de aplicativos é transmitido para o IBM MQ.	O identificador do usuário e a senha que foram passados no método `ConnectionFactory.createConnection(String username, String password)` são transmitidos para o IBM MQ.
O descritor de implementação do aplicativo não contém uma Referência de recurso para o connection factory, que está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER configurada para um identificador de usuários	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ que a connection factory está configurada para usar é transmitido para o IBM MQ.	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ que a connection factory está configurada para usar é transmitido para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de recurso para o connection factory que tem a propriedade **`res-auth`** configurada como *Application* e o connection factory está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER não configurada	O identificador do usuário para o processo do servidor de aplicativos é transmitido para o IBM MQ.	O identificador do usuário e a senha que foram passados no método `ConnectionFactory.createConnection(String username, String password)` são transmitidos para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de recurso para o connection factory que tem a propriedade **`res-auth`** configurada como *Application* e o connection factory está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER configurada para um identificador de usuários	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ o qual a connection factory está configurada para usar é transmitido para o IBM MQ.	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ o qual a connection factory está configurada para usar é transmitido para o IBM MQ.

Tabela 72. Modo CLIENT (continuação)

Configuração	Chamadas do aplicativo `ConnectionFactory.createConnection()`	Chamadas do aplicativo `ConnectionFactory.createConnection(String username, String password)`
O descritor de implementação do aplicativo contém uma Referência de recurso para o connection factory que tem a propriedade **`res-auth`** configurada como *Container* e o connection factory está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER não configurada	O identificador do usuário e a senha especificados no alias de autenticação para a connection factory são transmitidos para o IBM MQ.	O identificador do usuário e a senha especificados no alias de autenticação para a connection factory são transmitidos para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de recurso para o connection factory que tem a propriedade **`res-auth`** configurada como *Container* e o connection factory está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER configurada para um identificador de usuários	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ o qual a connection factory está configurada para usar é transmitido para o IBM MQ.	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ o qual a connection factory está configurada para usar é transmitido para o IBM MQ.
O descritor de implementação do aplicativo contém uma Referência de recurso para o connection factory que tem a propriedade **`res-auth`** configurada como *Container*, o aplicativo foi configurado com um alias de autenticação e o connection factory está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER não configurada	O identificador do usuário e a senha especificados no alias de autenticação que o aplicativo foi configurado para usar são transmitidos para o IBM MQ.	O identificador do usuário e a senha especificados no alias de autenticação que o aplicativo foi configurado para usar são transmitidos para o IBM MQ.

Tabela 72. Modo CLIENT (continuação)		
Configuração	**Chamadas do aplicativo `ConnectionFactory.createConnection()`**	**Chamadas do aplicativo `ConnectionFactory.createConnection(String username, String password)`**
O descritor de implementação do aplicativo contém uma Referência de recurso para o connection factory que tem a propriedade **`res-auth`** configurada como *Container*, o aplicativo foi configurado com um alias de autenticação e o connection factory está configurado para usar um canal do IBM MQ que tem a propriedade MCAUSER configurada para um identificador de usuários	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ o qual a connection factory está configurada para usar é transmitido para o IBM MQ.	O identificador do usuário especificado pela propriedade MCAUSER no canal do IBM MQ o qual a connection factory está configurada para usar é transmitido para o IBM MQ.

Usando o connection factory por meio de uma consulta direta

Após um connection factory do provedor de sistemas de mensagens do IBM MQ ter sido definido, um aplicativo corporativo pode consultar a definição de connection factory e usá-la para criar uma conexão JMS para um gerenciador de filas do IBM MQ. Isso pode ser feito por meio de uma consulta direta.

Para usar uma consulta direta, um aplicativo corporativo se conecta ao repositório JNDI do servidor de aplicativos, fazendo a chamada de método a seguir:

```
InitialContext ctx = new InitialContext();
```

Após ser conectado ao repositório JNDI, o aplicativo corporativo identifica a definição de connection factory usando o nome JNDI do connection factory, conforme a seguir:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

Notas:

- Seu desenvolvedor de aplicativos precisa saber o nome JNDI do connection factory necessário quando o aplicativo corporativo está sendo desenvolvido. Como o nome JNDI é codificado permanentemente no aplicativo, se o nome JNDI mudar, será necessário gravar novamente e reimplementar o aplicativo.
- Quando uma definição de connection factory é usada dessa maneira, o nome do usuário e a senha especificados no alias de autenticação (que o connection factory foi configurado para usar) não fluem para o IBM MQ. Isso é para evitar que aplicativos desautorizados identifiquem o connection factory e possam usá-lo para se conectar a sistemas IBM MQ seguros.

O nome do usuário e a senha que fluem para o IBM MQ dependem do método que é usado para criar uma conexão JMS do connection factory.

Se um aplicativo cria uma conexão JMS usando o método:

```
ConnectionFactory.createConnection()
```

a identidade do usuário padrão é passada para o IBM MQ. Esses são o nome do usuário e a senha que iniciaram o servidor de aplicativos no qual o aplicativo corporativo está em execução.

Como alternativa, um aplicativo pode criar uma conexão JMS chamando o método:

```
ConnectionFactory.createConnection(String username, String password)
```

Se um aplicativo executou uma consulta direta de um connection factory e, em seguida, chamou esse método, o nome do usuário e a senha que foram passados para o método `createConnection()` fluem para o IBM MQ.

Importante: Antes do IBM MQ 8.0, o IBM MQ processava uma verificação de autorização somente para assegurar que o nome do usuário que tivesse fluído para baixo tinha a autoridade para acessar o gerenciador de filas.

Nenhuma verificação era feita na senha. Para executar uma verificação de autenticação e validar que o nome do usuário e a senha eram válidos, uma saída de segurança de canal do IBM MQ precisava ser gravada. Detalhes sobre como fazer isso podem ser encontrados em "Programas de saída de segurança do canal" na página 981.

No IBM MQ 8.0, o gerenciador de filas verifica a senha além do nome do usuário.

Usando o connection factory por meio de uma consulta indireta

Quando você estiver gravando um aplicativo corporativo, se o nome do JNDI do connection factory for desconhecido ou se o aplicativo tiver que ser instalado em diferentes servidores de aplicativos usando um connection factory diferente, com um nome JNDI diferente (dependendo de em qual servidor de aplicativos ele estiver instalado), o connection factory poderá ser consultado usando uma referência de recurso. Isso poderá ser feito por meio de uma consulta indireta.

exemplo

Em vez de consultar diretamente o connection factory usando `jms/myCF`, um aplicativo corporativo contém uma referência de recurso que tem o nome JNDI local de: `jms/myResourceReferenceCF`.

Para usar esse nome JNDI, o aplicativo conecta-se ao repositório JNDI do servidor de aplicativos, da mesma maneira como se o aplicativo estivesse executando uma consulta direta:

```
InitialContext ctx = new InitialContext();
```

Em vez de identificar `jms/myCF` diretamente, o aplicativo agora identifica o nome JNDI da referência de recurso:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

Você precisa do prefixo `java:comp/env` para o nome JNDI local, para indicar ao servidor de aplicativos que o aplicativo corporativo está executando uma consulta indireta.

Quando o aplicativo é implementado, o usuário mapeia o nome JNDI da referência de recurso `jms/myResourceReferenceCF` para o nome JNDI do connection factory que o aplicativo já criou: `jms/myCF`.

Quando o aplicativo é executado, ele consulta um connection factory de JMS usando o nome JNDI local, que o servidor de aplicativos mapeia para: `jms/myCF`. Esse connection factory é, então, usado pelo aplicativo para criar uma conexão para o IBM MQ.

Aliases de Autenticação e Consultas Indiretas

Uma referência de recurso também permite que propriedades adicionais sejam definidas, que alteram o comportamento do connection factory fornecido. Uma das propriedades de uma referência de recursos é **`res-auth`**. O valor desta propriedade especifica se o aplicativo corporativo deve utilizar o alias de autenticação do connection factory ao qual o recurso de referência se mapeia ao criar uma conexão ao IBM MQ (se um alias de autenticação foi definido) ou se o aplicativo está especificando seu próprio nome de usuário e senha.

O valor padrão dessa propriedade é *Application*. Isso significa que o nome do usuário e a senha que fluem para o gerenciador de filas, quando uma conexão JMS é criada, são determinados pelo próprio aplicativo. O alias de autenticação do connection factory para o qual a referência de recurso é mapeada não é usado.

Os aplicativos podem criar conexões JMS usando um dos métodos a seguir:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Se um aplicativo usa `ConnectionFactory.createConnection()` e **res-auth** está configurado como *Application*, a identidade do usuário padrão flui para o IBM MQ. Esses são o nome do usuário e a senha que iniciaram o servidor de aplicativos no qual o aplicativo corporativo está em execução.

Se um aplicativo usar `ConnectionFactory.createConnection(String username, String password)`, e **res-auth** for configurado como *Aplicativo*, o nome do usuário e a senha transmitidos para o método serão enviados para IBM MQ

Para usar o alias de autenticação definido no connection factory para o qual a referência de recurso é mapeada ao criar uma conexão, você precisa configurar a propriedade **res-auth** para o valor *Container*. Quando um aplicativo cria uma conexão JMS, os detalhes do alias de autenticação são usados, mesmo se a chamada `createConnection` especifica um nome do usuário e senha.

Substituindo o alias de autenticação ao usar uma consulta indireta

Se um aplicativo usa uma referência de recurso que tem a propriedade **res-auth** configurada como *Container*, é possível substituir o alias de autenticação que é usado quando as conexões JMS são criadas.

Para substituir o alias de autenticação, a referência de recurso precisa incluir uma propriedade extra chamada **authDataAlias**, que é mapeada para um alias de autenticação existente que já tenha sido criado no ambiente do servidor de aplicativos no qual o aplicativo será implementado. É possível especificar essa propriedade em quaisquer referências de recurso que são criadas usando o conjunto de ferramentas do Rational fornecido pela IBM.

Usando esse método, é possível usar um alias de autenticação diferente ao usar um connection factory do JMS que tenha sido consultado indiretamente. Se o alias de autenticação especificado não existir, um novo poderá ser especificado após a instalação do aplicativo corporativo. Para obter mais informações, veja *Referências de Recurso* na documentação do produto WebSphere Application Server.

Informações relacionadas para WebSphere Application Server 8.5.5

Referências de Recurso

Informações relacionadas para WebSphere Application Server 8.0

Referências de Recurso

Informações relacionadas para WebSphere Application Server 7.0

Referências de Recurso

Balanceamento de carga de trabalho para beans acionados por mensagens ao usar clusters do WebSphere Application Server

Ao usar os aplicativos de bean acionado por mensagens implementados em um cluster do WebSphere Application Server 7.0 e WebSphere Application Server 8.0 e configurados para execução no modo normal do provedor de sistemas de mensagens do IBM MQ, um dos membros de cluster processa a maioria das mensagens. É possível equilibrar a carga de trabalho de membros de cluster para distribuir o processamento de mensagens em mais de um membro de cluster.

IBM MQ inclui um recurso chamado **Asynchronous consume**, que permite que os aplicativos consumam mensagens de forma assíncrona de uma fila usando APIs chamadas **MQCB** e **MQCTL**.

Os aplicativos de bean acionado por mensagens dentro do WebSphere Application Server 7.0 e WebSphere Application Server 8.0 que usam o modo normal do provedor de sistemas de mensagens do IBM MQ usarão automaticamente esse recurso. Quando os aplicativos forem inicializados, eles configurarão um consumidor assíncrono no destino JMS que eles foram configurados para monitorar

chamando **MQCB**. A API **MQCTL** será, então, chamada para indicar que o aplicativo está pronto para receber mensagens do destino JMS.

Quando os aplicativos de bean acionado por mensagens tiverem sido implementados em um cluster do WebSphere Application Server, cada membro de cluster configurará um consumidor assíncrono para o destino JMS que o bean acionado por mensagens está monitorando para mensagens. O gerenciador de filas do IBM MQ que hospeda o destino JMS serão, então, responsável por notificar o membro de cluster quando houver uma mensagem adequada no destino JMS para ele processar.

Quando o WebSphere Application Server estiver se conectando a um gerenciador de filas do IBM MQ , as mensagens que chegam em um destino do JMS serão distribuídas mais uniformemente para todos os consumidores assíncronos que foram registrados nesse destino do JMS Para aplicativos de bean acionado por mensagens implementados dentro de um cluster WebSphere Application Server 7.0 e WebSphere Application Server 8.0, isso significa que as mensagens serão distribuídas mais uniformemente entre os membros de cluster.

Tarefas relacionadas

Configurando a propriedade JMS **PROVIDERVERSION**

Usando o pacote IBM MQ Headers

O pacote IBM MQ Headers fornece um conjunto de interfaces e classes auxiliares que podem ser usados para manipular os cabeçalhos do IBM MQ de uma mensagem. Geralmente, você usa o pacote IBM MQ Headers porque deseja executar serviços administrativos usando o servidor de comandos (usando mensagens de Formato de Comando Programável (PCF)).

Sobre esta tarefa

O pacote IBM MQ Headers está localizado nos pacotes `com.ibm.mq.headers` e `com.ibm.mq.headers.pcf`. É possível usar esse recurso para as duas APIs alternativas que o IBM MQ fornece para uso em aplicativos Java:

- IBM MQ classes for Java (também referido como IBM MQ Base Java).
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, também referido como IBM MQ JMS).

Os aplicativos IBM MQ Base Java geralmente manipulam objetos MQMessage e as classes de suportes Headers podem interagir diretamente com esses objetos, desde que entendam nativamente as interfaces do IBM MQ Base Java.

No IBM MQ JMS, a carga útil para uma mensagem é normalmente uma sequência ou um objeto de matriz de bytes, que pode ser manipulado com os fluxos DataInput e DataOutput. O pacote IBM MQ Headers pode ser usado para interagir com esses fluxos de dados e é adequado para manipular qualquer mensagem MQ enviada e recebida pelos aplicativos IBM MQ JMS.

Portanto, embora o pacote IBM MQ Headers contenha referências ao pacote IBM MQ Base Java, ele também é destinado a ser usado em aplicativos IBM MQ JMS e é adequado para ser usado em ambientes Java Platform, Enterprise Edition (Java EE).

Uma maneira típica de usar o pacote IBM MQ Headers é manipular mensagens de administração no Formato de Comando Programável (PCF), por exemplo, por qualquer uma das seguintes razões:

- Para acessar detalhes sobre um recurso do IBM MQ.
- Para monitorar a profundidade de uma fila.
- Para inibir o acesso a uma fila.

Ao usar as mensagens PCF com a API do IBM MQ JMS, esse tipo de administração de recursos centrados em aplicativo pode ser executada em aplicativos Java EE sem ter que recorrer ao uso da API do IBM MQ Base Java.

Procedimento

- Para usar o pacote IBM MQ Headers para manipular cabeçalhos de mensagem para o IBM MQ classes for Java, consulte "Usando com o IBM MQ classes for Java" na página 527.
- Para usar o pacote IBM MQ Headers para manipular cabeçalhos de mensagem para o IBM MQ classes for JMS, consulte "Usando com o IBM MQ classes for JMS" na página 527.

Usando com o IBM MQ classes for Java

Os aplicativos IBM MQ classes for Java geralmente manipulam os objetos MQMessage e as classes de suporte do Headers podem interagir diretamente com esses objetos, pois elas entendem nativamente as interfaces do IBM MQ classes for Java.

Sobre esta tarefa

O IBM MQ fornece alguns aplicativos de amostra que demonstram como usar o pacote IBM MQ Headers com a API do IBM MQ Base Java (IBM MQ classes for Java).

As amostras mostram duas coisas:

- Como criar uma mensagem PCF para executar uma ação administrativa e analisar a mensagem de resposta.
- Como enviar essa mensagem PCF usando o IBM MQ classes for Java.

Dependendo da plataforma usada, essas amostras são instaladas sob o diretório `pcf` no diretório `samples` ou `tools` da instalação do IBM MQ (consulte "Diretórios de instalação para o IBM MQ classes for Java" na página 361).

Procedimento

1. Crie uma mensagem PCF para executar uma ação administrativa e analise a mensagem de resposta.
2. Envie essa mensagem PCF usando o IBM MQ classes for Java.

Conceitos relacionados

"Manipulando cabeçalhos de mensagem do IBM MQ com o IBM MQ classes for Java" na página 389
Classes Java são fornecidas representando diferentes tipos de cabeçalho de mensagem. Duas classes auxiliares também são fornecidas.

"Manipulando mensagens PCF com o IBM MQ classes for Java" na página 394
Classes Java são fornecidas para criar e analisar mensagens estruturadas por PCF e para facilitar o envio de solicitações PCF e a coleta de respostas PCF.

Usando com o IBM MQ classes for JMS

Para usar o IBM MQ Headers com o IBM MQ classes for JMS, você executa as mesmas etapas essenciais que para o IBM MQ classes for Java. A mensagem PCF pode ser criada e a resposta analisada exatamente da mesma maneira usando o pacote IBM MQ Headers e o mesmo código de amostra que para o IBM MQ classes for Java.

Sobre esta tarefa

Para enviar uma mensagem PCF usando a API do IBM MQ, a carga útil da mensagem deve ser gravada em uma mensagem do JMS Bytes e enviada usando as APIs padrão do JMS. A única consideração é que a mensagem não deve conter um JMS RFH2 ou qualquer outro cabeçalho com valores específicos no MQMD.

Para enviar uma mensagem PCF, conclua as etapas a seguir. A maneira na qual a mensagem PCF é criada e as informações são extraídas da mensagem de resposta é a mesma que para o IBM MQ classes for Java (consulte "Usando com o IBM MQ classes for Java" na página 527).

Procedimento

1. Crie um JMS Destino de Fila que representa o SYSTEM.ADMIN.COMMAND.QUEUE.

   Os aplicativos IBM MQ JMS enviam as mensagens PCF para SYSTEM.ADMIN.COMMAND.QUEUE, e precisa de acesso a um objeto de Destino JMS que representa essa fila O destino deve ter as seguintes propriedades do conjunto:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Se você está usando o WebSphere Application Server, deve-se definir essas propriedades como propriedades customizadas no Destino.

   Para criar o destino programaticamente a partir de um aplicativo, use o código a seguir:

   ```
   Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
   ((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
       WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQQueue) q1).setMQMDWriteEnabled(true);
   ```

2. Converta uma mensagem PCF em uma mensagem do JMS Bytes que contém os valores MQMD corretos.

   Uma mensagem do JMS Bytes precisa ser criada e a mensagem PCF gravada nele. Uma fila de resposta precisa ser criada, mas ela não precisa ter nenhuma configuração específica.

   O fragmento de código de amostra a seguir mostra como criar uma mensagem do JMS Bytes e escrever um objeto com.ibm.mq.headers,pcf.PCFMessage nela. O objeto PCFMessage (pcfCmd) foi construído anteriormente usando o pacote IBM MQ Headers. (Observe que o pacote para carregar o PCFMessage é com.ibm.mq.headers.pcf.PCFMessage).

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();

   // Create the wrapping streams to put the bytes into the message payload
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   DataOutput dataOutput = new DataOutputStream(baos);

   // Set the JMSReplyTo so the answer comes back
   msg.setJMSReplyTo(new MQQueue("adminResp"));

   // write the pcf into the stream
   pcfCmd.write(dataOutput);
   baos.flush();
   msg.writeBytes(baos.toByteArray());

   // we have taken control of the MD, so need to set all
   // flags in the MD that we require - main one is the format
   msg.setJMSPriority(4);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
     CMQC.MQPER_NOT_PERSISTENT);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
       CMQC.MQRO_PASS_CORREL_ID);
   msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

   // and send the message
   sender.send(msg);
   ```

3. Envie a mensagem e receba a resposta usando as APIs padrão do JMS.
4. Converta a mensagem de resposta em uma mensagem PCF para processamento.

   Para recuperar a mensagem de resposta e processá-la como uma mensagem PCF, use o código a seguir:

   ```
    // Get the message back
   BytesMessage msg = (BytesMessage) consumer.receive();

   // get the size of the bytes message & read into an array
   int bodySize = (int) msg.getBodyLength();
   byte[] data = new byte[bodySize];
   msg.readBytes(data);
   ```

```
// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

Conceitos relacionados

“Mensagens JMS” na página 146
As mensagens do JMS são compostas de um cabeçalho, propriedades e um corpo. O JMS define cinco tipos de corpo da mensagem.

IBM i Configurando o IBM MQ no IBM i com Java e JMS

Esta coleção de tópicos fornece uma visão geral de como configurar e testar o IBM MQ com o Java e JMS no IBM i usando os comandos de CL ou o ambiente qshell.

Nota:

- A partir de IBM MQ 8.0, `ldap.jar`, `jndi.jar` e `jta.jar` fazem parte do JDK.
- JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando IBM MQ classes para JMS/Jakarta Messaging.

Usando os comandos de CL

O CLASSPATH configurado é para teste com MQ base Java, JMS com JNDI e JMS sem JNDI.

Se você não utilizar um arquivo `.profile` sob o seu diretório `/home/Userprofile`, será necessário configurar as variáveis de ambiente de nível do sistema abaixo. É possível verificar se estão configuradas usando o comando **WRKENVVAR**.

1. Para visualizar as variáveis de ambiente para o sistema inteiro emita o comando: **WRKENVVAR LEVEL(*SYS)**
2. Para visualizar as variáveis de ambiente específicas para sua tarefa emita o comando: **WRKENVVAR LEVEL(*JOB)**
3. Se o CLASSPATH não estiver configurado, emita o seguinte comando:

JM 3.0
```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS 2.0
```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. Se QIBM_MULTI_THREADED não estiver configurado, emita o seguinte comando:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. Se QIBM_USE_DESCRIPTOR_STDIO não estiver configurado, emita o comando a seguir:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Se QSH_REDIRECTION_TEXTDATA não estiver configurado, emita o seguinte comando:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

Usando o ambiente qshell

Se você usar o ambiente QSHELL, você pode configurar um `.profile` em seu diretório `/home/Userprofile`. Para obter mais referência de informações sobre a documentação do Qshell Interpreter (qsh).

Especifique o seguinte no `.profile`. Observe que a instrução CLASSPATH deve estar em uma única linha ou separados em linhas diferentes usando o caractere \, conforme mostrado.

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

Assegure-se de que a biblioteca QMQMJAVA esteja na lista de bibliotecas emitindo o comando **DSPLIBL**.

Se a biblioteca QMQMJAVA não estiver na lista, a inclua usando o seguinte comando: **ADDLIBLE LIB(QMQMJAVA)**

IBM i Testando o IBM MQ no IBM i com Java

Como você testa o IBM MQ com o Java usando o programa de amostra MQIVP.

Testando o IBM MQ base Java

Execute o seguinte procedimento:

1. Verifique se o gerenciador de filas foi iniciado e se o estado do gerenciador de filas é ACTIVE emitindo o comando a seguir:

```
WRKMQM MQMNAME(QMGRNAME)
```

2. Verifique se o canal de conexão do servidor JAVA.CHANNEL foi criado emitindo o comando a seguir:

```
WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

   a. Se o JAVA.CHANNEL não existe, emita o comando a seguir:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

3. Verifique se o listener do gerenciador de filas está em execução para a porta 1414 ou qual porta você está usando, emitindo o comando **WRKMQMLSR**.

   a. Se nenhum listener foi iniciada para o gerenciador de filas, emita o comando a seguir:

```
STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
```

Executando o programa de teste de amostra MQIVP

1. Inicie o qshell, a partir da linha de comandos emitindo o comando STRQSH
2. Verifique se o CLASSPATH correto está configurado emitindo o comando **export** e, em seguida, emita o comando **cd** da seguinte forma:

```
cd /qibm/proddata/mqm/java/samples/wmqjava/samples
```

3. Execute o programa **java** emitindo o comando a seguir:

```
java MQIVP
```

É possível pressionar a tecla ENTER quando solicitado:

- Tipo de conexão
- endereço IP
- Nome do gerenciador de filas

para usar os valores padrão. Verifica as ligações do produto, que podem estar localizadas na biblioteca QMQMJAVA.

Você recebe saída semelhante ao exemplo a seguir. Observe que a instrução de copyright depende da versão do produto que você está usando.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

Testando a conexão do cliente IBM MQ Java

Deve-se especificar:

- Tipo de conexão
- endereço IP
- Port
- Canal de conexão do servidor
- Gerenciador de filas

Você recebe saída semelhante ao exemplo a seguir. Observe que a instrução de copyright depende da versão do produto que você está usando.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i Testando o IBM MQ no IBM i com JMS

Como você testa IBM MQ com o JMS com e sem JNDI

Testando JMS sem JNDI usando a amostra IVTRun

Execute o seguinte procedimento:

1. Verifique se o gerenciador de filas foi iniciado e se o estado do gerenciador de filas é ACTIVE emitindo o comando a seguir:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Inicie o qshell, a partir da linha de comandos, emitindo o comando **STRQSH**.
3. Use o comando **cd** para mudar de diretório da seguinte forma:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

4. Execute o arquivo de script:

   ```
   IVTRun -nojndi [-m qmgrname]
   ```

Você recebe saída semelhante ao exemplo a seguir. Observe que as declarações de copyright dependem das versões dos produtos que você está usando:

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
```

```
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

Testando modo do cliente IBM MQ JMS sem JNDI

Execute o seguinte procedimento:

1. Verifique se o gerenciador de filas foi iniciado e se o estado do gerenciador de filas é ACTIVE emitindo o comando a seguir:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Verifique se o canal de conexão do servidor foi criado emitindo o comando a seguir:

   ```
   WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
   MQMNAME(QMGRNAME)
   ```

3. Verifique se o listener está iniciado para a porta correta emitindo o comando **WRKMQMLSR**
4. Inicie o qshell, a partir da linha de comandos, emitindo o comando **STRQSH**.
5. Verifique se o CLASSPATH está correto emitindo o comando **export**.
6. Use o comando **cd** para mudar de diretório da seguinte forma:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

7. Execute o arquivo de script:

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

Você recebe saída semelhante ao exemplo a seguir. Observe que as declarações de copyright dependem das versões dos produtos que você está usando.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Testando IBM MQ JMS com JNDI

Verifique se o gerenciador de filas foi iniciado e se o estado do gerenciador de filas é ACTIVE emitindo o comando a seguir:

```
WRKMQM MQMNAME(QMGRNAME)
```

Usando o script de teste de amostra IVTRun

Execute o seguinte procedimento:

1. Faça as alterações apropriadas ao arquivo `JMSAdmin.config`. Para editar esse arquivo, use o comando **EDTF** (Editar arquivo) a partir de uma linha de comandos do IBM i

```
EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
```

.

a. Para usar LDAP for Weblogic, remova o comentário de:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
```

b. Para usar o LDAP for WebSphere Application Server, remova o comentário de:

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
```

c. Para testar o sistema de arquivos, remova o comentário de:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

d. Assegure que tenha selecionado o PROVIDER_URL correto, removendo o comentário da linha apropriada.

e. Comente todas as outras linhas usando o símbolo *#*.

f. Quando tiver concluído todas as mudanças, pressione **F2=Save** e **F3=Exit**.

2. Inicie o qshell, a partir da linha de comandos, emitindo o comando **STRQSH**.
3. Verifique se o CLASSPATH está correto emitindo o comando **export**.
4. Use o comando **cd** para mudar de diretório da seguinte forma:

```
cd /qibm/proddata/mqm/java/bin
```

5. Inicie o script **IVTSetup** para criar os objetos administrados (*MQQueueConnectionFactory* e *MQQueue*), emitindo o comando **IVTSetup**.
6. Execute o script IVTRun emitindo o comando a seguir:

```
IVTRun -url providerURL [-icf initCtxFact]
```

Você recebe saída semelhante ao exemplo a seguir. Observe que as declarações de copyright dependem das versões dos produtos que você está usando.

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
```

```
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Desenvolvimento de aplicativos Java usando um repositório Maven

Ao desenvolver um aplicativo Java para o IBM MQ usando um repositório Maven para instalar dependências automaticamente, não é necessário instalar nada explicitamente antes de usar as interfaces do IBM MQ.

Armazenador central Maven

O Maven é uma ferramenta para construir aplicativos e também fornece um repositório para armazenar artefatos que seu aplicativo pode desejar acessar.

O repositório (ou Armazenador Central) Maven possui uma estrutura que permite que arquivos como arquivos JAR tenham versões distintas que são, então, facilmente descobertas com um conhecido mecanismo de nomenclatura. As ferramentas de construção podem usar esses nomes para extrair dinamicamente as dependências para o seu aplicativo. Na definição de seu aplicativo, que, ao usar Maven como uma ferramenta de construção, é chamado de arquivo POM, você nomeará as dependências e o processo de construção saberá o que fazer a partir daí.

Arquivos do cliente IBM MQ

As cópias das interfaces do cliente IBM MQ Java estão disponíveis no Armazenador Central sob o `com.ibm.mq GroupId`. É possível localizar o arquivo `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) e o arquivo `com.ibm.mq.allclient.jar` (JMS 2.0). Esses arquivos são geralmente usados para programas independentes. Também é possível localizar o arquivo `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) e o arquivo `wmq.jmsra.rar` (JMS 2.0), que é para uso em servidores de aplicativos Java EE . O `jakarta.client.jar` e o `allclient.jar` contêm IBM MQ classes for JMS e o IBM MQ classes for Java.

Importante: O uso do formato do Apache Maven Assembly Plugin, *jar-with-dependencies*, para construir um aplicativo que inclui o arquivo JAR realocável do IBM MQ não é suportado.

Em um arquivo `pom.xml` processado pelo comando maven, você inclui dependências para estes arquivos JAR, como mostrado nos exemplos a seguir:

- JM 3.0 Para mostrar a relação entre o seu código do aplicativo e `com.ibm.mq.jakarta.client.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 Para mostrar a relação entre o seu código do aplicativo e `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- JM 3.0 Para usar o adaptador de recursos do Jakarta EE:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 Para usar o adaptador de recurso JMS 2.0 Java EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Para obter um exemplo de um projeto simples no Eclipse para executar um projeto do JMS , consulte o IBM Developer artigo Desenvolvendo Java aplicativos para o MQ que ficou mais fácil com o Maven

Desenvolvendo aplicativos C++

O IBM MQ fornece classes C++ equivalentes a objetos do IBM MQ e algumas classes adicionais equivalentes aos tipos de dados de matriz. Ele fornece inúmeros recursos não disponíveis por meio do MQI.

IBM WebSphere MQ 7.0, aprimoramentos das interfaces de programação do IBM MQ não são aplicados às classes C++.

O IBM MQ C++ fornece os recursos a seguir:

- Inicialização automática das estruturas de dados do IBM MQ.
- Conexão do gerenciador de filas e abertura de fila just-in-time.
- Fechamento de fila e desconexão do gerenciador de filas implícitos.
- Transmissão e recebimento do cabeçalho de mensagens não entregues.
- Transmissão e recebimento de cabeçalho de ponte IMS.
- Transmissão e recebimento de cabeçalho de mensagem de referência.
- Recebimento de mensagem do acionador.
- Transmissão e recebimento de cabeçalho de CICS bridge.
- Recebimento e transmissão de cabeçalho do trabalho.
- Definição de canal do cliente.

Os diagramas de classe Booch a seguir mostram que todas as classes são amplamente paralelas às entidades do IBM MQ na MQI processual (por exemplo usando C) que têm identificadores ou estruturas

de dados. Todas as classes são herdadas da classe ImqError (consulte Classe C++ ImqError), o que permite que uma condição de erro seja associada a cada objeto.

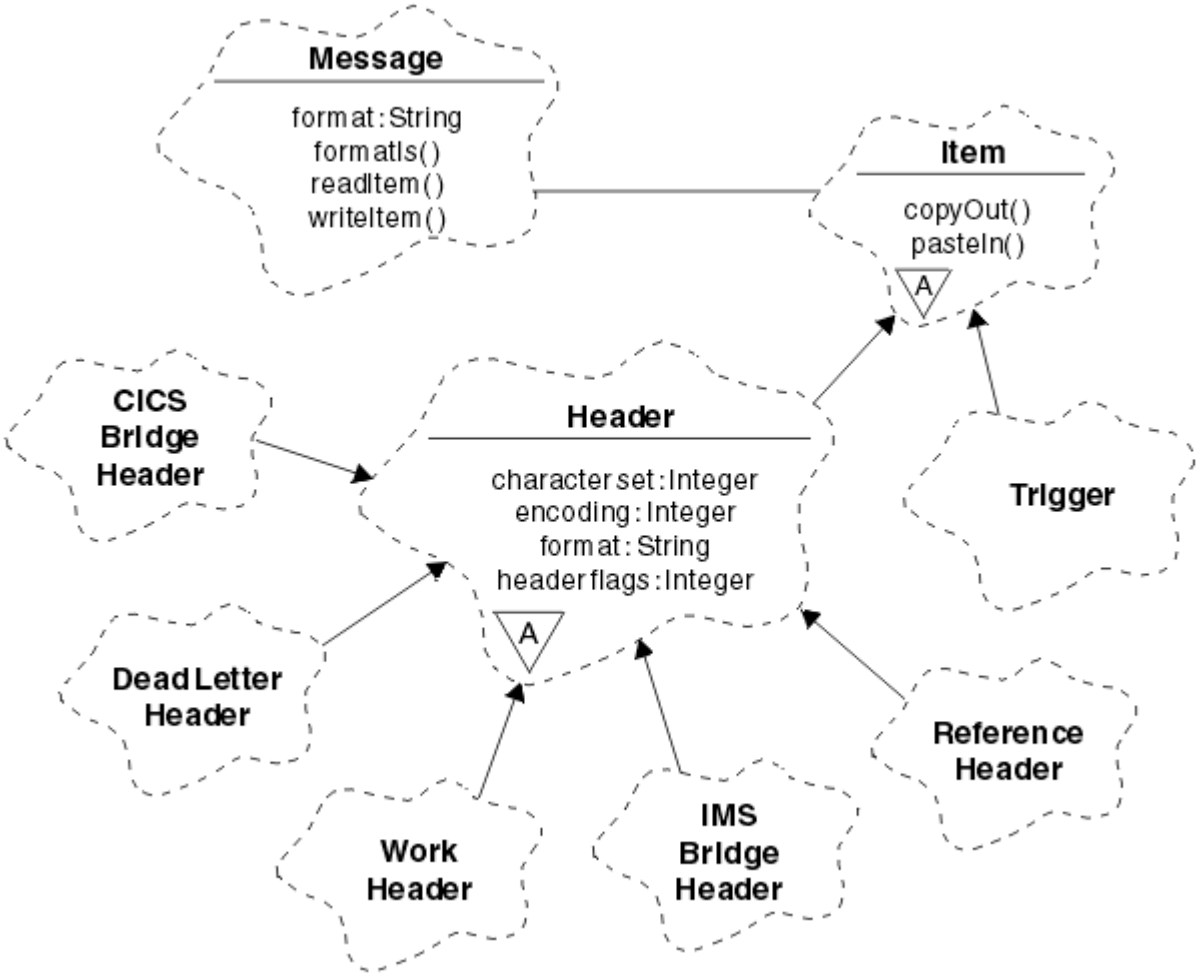


Figura 50. Classes IBM MQ C++ (manipulação de item)

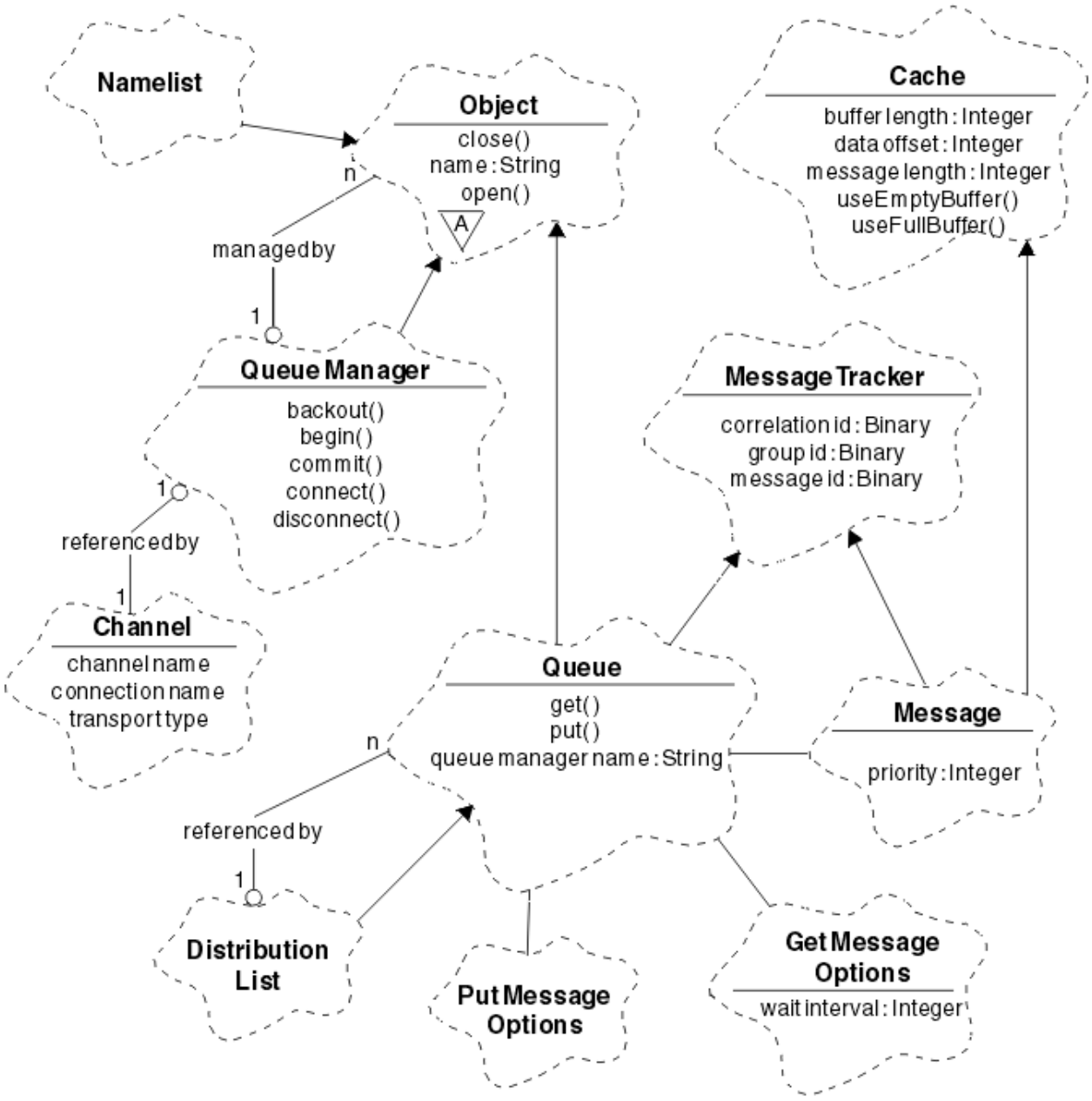


Figura 51. Classes IBM MQ C++ (gerenciamento de fila)

Para interpretar os diagramas de classe Booch corretamente, esteja ciente das convenções a seguir:

- Os métodos e os atributos que vale a pena serem observados são mostrados abaixo do nome da *classe*.
- Um pequeno triângulo em uma nuvem denota uma *classe abstrata*.
- A*herança* é denotada por uma seta para a classe-pai.
- Uma linha não decorada entre as nuvens denota um *relacionamento cooperativo* entre as classes.
- Uma linha decorada com um número denota um *relacionamento referencial* entre duas classes. O número indica o número de objetos que podem participar de um determinado relacionamento a qualquer momento.

As classes e os tipos de dados a seguir são usados nas assinaturas de método C++ das classes de gerenciamento de fila (consulte Figura 51 na página 539) e as classes de manipulação de item (consulte Figura 50 na página 538):

- A classe ImqBinary (consulte Classe C++ ImqBinary), que contém matrizes de bytes, como MQBYTE24.

- O tipo de dados ImqBoolean, que é definido como **typedef unsigned char ImqBoolean**.
- A classe ImqString (consulte Classe C++ ImqString), que contém matrizes de caracteres, como MQCHAR64.

As entidades com estruturas de dados são incluídas nas classes de objeto apropriadas. Campos de estrutura de dados individuais (consulte Referência cruzada de C++ e MQI) são acessados com métodos.

Entidades com identificadores estão abaixo da hierarquia da classe ImqObject (consulte Classe C++ ImqObject) e fornecem interfaces contidas na MQI. Objetos dessas classes exibem o comportamento inteligente que pode reduzir o número de chamadas de método necessárias relativas à MQI processual. Por exemplo, é possível estabelecer e descartar as conexões do gerenciador de filas conforme necessário ou abrir uma fila com as opções a apropriadas e, em seguida, fechá-la.

A classe ImqMessage (consulte Classe C++ ImqMessage) contém a estrutura de dados MQMD e também age como um ponto de contenção de dados do usuário e *itens* (consulte “Lendo mensagens em C++” na página 549), fornecendo recursos de buffer em cache. É possível fornecer buffers de comprimento fixo para dados do usuário e usar o buffer muitas vezes. A quantia de dados presente no buffer pode variar de um uso para o próximo. Como alternativa, o sistema pode fornecer e gerenciar um buffer de comprimento flexível. O tamanho do buffer (a quantia disponível para recebimento de mensagens) e a quantia realmente usada (o número de bytes para transmissão ou o número de bytes realmente recebidos) se tornam considerações importantes.

Conceitos relacionados

Visão geral técnica

“Programas de amostra C++” na página 540
Quatro programas de amostra são fornecidos, para demonstrar a obtenção e a colocação de mensagens.

“contraprestações sobre linguagem C++” na página 544
Esta coleção de tópicos detalha os aspectos do uso da linguagem C++ e convenções que deve-se considerar ao gravar programas de aplicativos que usam um Message Queue Interface (MQI).

“Preparando dados da mensagem em C++” na página 548
Os dados da mensagem são preparados em um buffer, que pode ser fornecido pelo sistema ou aplicativo. Há vantagens para qualquer de um dos métodos. Exemplos de como usar um buffer são fornecidos.

“Desenvolvendo aplicativos para o IBM MQ” na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

Referências relacionadas

“Construindo programas C++ IBM MQ” na página 555
A URL de compiladores suportados está listada, juntamente com os comandos a serem usados para compilar, vincular e executar programas C++ e amostras em plataformas IBM MQ.

Referência cruzada de C++ e MQI

Classes IBM MQ C++

Programas de amostra C++

Quatro programas de amostra são fornecidos, para demonstrar a obtenção e a colocação de mensagens.

Os programas de amostra são:

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

Os programas de amostra estão localizados nos diretórios mostrados em Tabela 73 na página 541.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Tabela 73. Local de programas de amostra		
Meio ambiente	**Diretório que contém a origem**	**Diretório que contém o construído programas**
AIX AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia .
V 9.4.0 AIX AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (consulte a nota “1” na página 541)
IBM i IBM i	`/QIBM/ProdData/mqm/samp/`	(consulte a nota “2” na página 541)
Linux Linux	*MQ_INSTALLATION_PATH*/samp	Nenhum
Windows Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn (consulte a nota “3” na página 541)
z/OS z/OS	thlqual.SCSQCPPS	

Notas:

1. V 9.4.0 AIX Programas construídos usando o compilador XLC 17 estão localizados na pasta "`ca`", enquanto programas construídos usando o compilador XLC 16 estão localizados na pasta "`ia`".
2. IBM i Os programas construídos usando o compilador ILE C++ para o IBM i estão na biblioteca QMQM. Os arquivos de origem estão em `/QIBM/ProdData/mqm/samp`
3. Windows Os programas construídos usando o Microsoft Visual Studio Visual Studio estão localizados em *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn`. Para obter informações adicionais sobre esses compiladores, consulte “Construindo programas C++ no Windows” na página 561.

Programa de amostra HELLO WORLD (imqwrld.cpp)

Este programa de amostra C++ mostra como colocar e obter um datagrama regular (estrutura C) usando a classe ImqMessage.

Este programa mostra como colocar e obter um datagrama regular (estrutura C) usando a classe ImqMessage. Esta amostra usa algumas chamadas de método, aproveitando chamadas de método implícitas como **abrir**, **fechar** e **desconectar**.

Em todas as plataformas, exceto z/OS

Se você estiver usando uma conexão do servidor com IBM MQ, siga um dos procedimentos a seguir:

- Para usar a fila padrão existente, SYSTEM.DEFAULT.LOCAL.QUEUE, execute o programa **`imqwrlds`** sem passar nenhum parâmetro
- Para usar uma fila temporária dinamicamente designada, execute **`imqwrlds`** transmitindo o nome da fila de modelo padrão, SYSTEM.DEFAULT.MODEL.QUEUE.

Se você estiver usando uma conexão do cliente com IBM MQ, siga um dos procedimentos a seguir:

- Configure a variável de ambiente MQSERVER (consulte *MQSERVER* para obter mais informações) e execute **imqwrldc**ou
- Execute **imqwrldc** transmitindo-os como parâmetros **queue-name**, **queue-manager-name** e **channel-definition**, em que um **channel-definition** típico deve ser SYSTEM.DEF.SVRCONN/TCP/ *hostname* (1414)

Em z/OS

z/OS

Construa e execute uma tarefa em lote, usando a amostra de JCL **imqwrldr**.

Consulte z/OS Batch, RRS Batch e CICS para obter mais informações.

Código de amostra

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
```

```
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

Programas de amostra SPUT (imqsput.cpp) e SGET (imqsget.cpp)

Esses programas C++ colocam mensagens e recuperam as mensagens de uma fila nomeada.

Essas amostras mostram o uso das seguintes classes:

- ImqError (consulte Classe C++ ImqError)
- ImqMessage (consulte Classe C++ ImqMessage)

- ImqObject (consulte Classe C++ ImqObject)
- ImqQueue (consulte Classe C++ ImqQueue)
- ImqQueueManager (consulte Classe C++ ImqQueueManager)

Siga as instruções apropriadas para executar os programas.

Em todas as plataformas, exceto z/OS

1. Execute **imqsputs** *queue-name*.
2. Linhas de tipo de texto no console. Essas linhas são colocados como mensagens na fila especificada.
3. Insira uma linha nula para terminar a entrada.
4. Execute **imqsgets** *queue-name* para recuperar todas as linhas e exibi-las no console.

z/OS Consulte o "Building C++ programs on z/OS Batch, RRS Batch and CICS" na página 563 para obter informações adicionais.

Em z/OS

z/OS

1. Construa e execute uma tarefa em lote usando o JCL **imqsputr** de amostra. As mensagens são lidas a partir do conjunto de dados SYSIN.
2. Construa e execute uma tarefa em lote usando o JCL **imqsgetr** de amostra. As mensagens são recuperadas da fila e enviadas para o conjunto de dados SYSPRINT.

Programa de amostra DPUT (imqdput.cpp)

Este programa de amostra C++ coloca mensagens em uma lista de distribuição que consiste em duas filas.

DPUT mostra o uso da classe ImqDistributionList (consulte Classe C++ ImqDistributionList). Esta amostra não é suportada no z/OS.

1. Execute **imqdputs** *queue-name-1 queue-name-2* para colocar mensagens nas duas filas nomeadas.
2. Execute **imqsgets** *queue-name-1* e **imqsgets** *queue-name-2* para recuperar as mensagens dessas filas.

contraprestações sobre linguagem C++

Esta coleção de tópicos detalha os aspectos do uso da linguagem C++ e convenções que deve-se considerar ao gravar programas de aplicativos que usam um Message Queue Interface (MQI).

Arquivos de cabeçalho C++

Os arquivos de cabeçalho são fornecidos como parte da definição do MQI, para ajudá-lo a gravar programas de aplicativo IBM MQ na linguagem C++.

Esses arquivos de cabeçalho são resumidos na tabela a seguir.

Tabela 74. arquivos de cabeçalho C/C++

Nome do arquivo	**Índice**
IMQI.HPP	Classes C++ MQI (inclui CMQC.H e IMQTYPE.H)
IMQTYPE.H	Define o tipo de dados do **ImqBoolean**
CMQC.H	estruturas de dados MQI e constantes manifest

Para melhorar a portabilidade dos aplicativos, codifique o nome do arquivo de cabeçalho em minúsculas na diretriz do pré-processador **#include**:

```
#include <imqi.hpp> // C++ classes
```

Métodos C++ e atributos

Nomes de métodos são compostos por letras maiúsculas e minúsculas. Várias considerações se aplicam aos parâmetros e valores de retorno. Atributos são acessados usando os métodos get e set conforme apropriado.

Parâmetros dos métodos que são *const* são apenas para entrada. Os parâmetros com assinaturas incluindo um ponteiro (*) ou uma referência (&) são passados por referência. Valores de retorno que não incluem um ponteiro ou uma referência são transmitidos por valor; no caso de objetos retornados, estes serão novas entidades que se tornarão responsabilidade do responsável pela chamada.

Algumas assinaturas de método incluem itens que tomam um padrão se não forem especificados. Tais itens estão sempre no final de assinaturas e são indicados por um sinal de igual (=); o valor após o sinal de igual indica o valor padrão que se aplicará se o item for omitido.

Todos os nomes de métodos nessas classes são compostos por letras maiúsculas e minúsculas, começando por minúsculas. Cada palavra, exceto a primeira dentro de um nome de método, começa com uma letra maiúscula. Abreviações não são usadas a menos que seu significado seja amplamente compreendido. Abreviaturas usadas incluem *ID* (para identidade) e *sync* (para sincronização).

Os atributos do objeto são acessados usando métodos get e set. Um método set começa com a palavra *set*; um método get não possui nenhum prefixo. Se um atributo for *read-only*, não haverá método set.

Atributos são inicializados para estados válidos durante a construção do objeto e o estado de um objeto é sempre consistente.

Tipos de dados em C++

Todos os tipos de dados são definidos pela instrução C **typedef**.

O tipo **ImqBoolean** é definido como **unsigned char** em IMQTYPE.H e pode ter os valores TRUE e FALSE. É possível usar os objetos de classe **ImqBinary** no lugar de matrizes **MQBYTE**, e objetos de classe **ImqString** no lugar de **char ***. Muitos métodos retornam objetos em vez de ponteiros **char** ou **MQBYTE** para facilitar o gerenciamento de armazenamento. Todos os valores de retorno se tornam responsabilidade do responsável pela chamada, e, no caso de um objeto retornado, o armazenamento pode ser disposto a usar a exclusão.

Manipulação de sequências binárias em C++

Sequências de dados binários são declaradas como objetos da classe **ImqBinary**. Objetos dessa classe podem ser copiados, comparados e configurados usando os operadores C familiares. Código de exemplo é fornecido.

A amostra de código a seguir mostra operações em uma sequência binária:

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
...
```

Manipulação de sequências de caracteres em C++

Os dados de caracteres são geralmente retornados nos objetos de classe **ImqString** que podem ser convertidos em **char *** usando um operador de conversão. A classe ImqString contém métodos para auxiliar no processamento de sequências de caracteres.

Quando os dados de caractere são aceitos ou retornados usando métodos MQI C++, os dados de caracteres são sempre terminados em nulos e podem ter qualquer comprimento. No entanto, determinados limites são impostos pelo IBM MQ que podem resultar em informações que estão sendo truncadas. Para facilitar o gerenciamento de armazenamento, os dados de caracteres são geralmente retornados em objetos de classe **ImqString**. Esses objetos podem ser convertidos em **char ***, usando o operador de conversão fornecido e usado para fins de *leitura apenas* em muitas situações em que um **char *** é necessário.

Nota: A conversão **char *** resulta de um objeto de classe **ImqString** pode ser nula.

Embora as funções C possam ser usadas no **char ***, há métodos especiais da classe **ImqString** que são preferenciais; **operator length** ( ) é equivalente a **strlen** e **storage** ( ) indica a memória alocada para os dados de caractere.

Estado inicial de objetos em C++

Todos os objetos possuem um estado inicial consistente refletido por seus atributos. Os valores iniciais são definidos nas descrições de classe.

Usando C a partir de C++

Ao usar as funções C de um programa C++, inclua cabeçalhos apropriados.

O exemplo a seguir mostra `string.h` incluído em um programa C++:

```
extern "C" {
#include <string.h>
}
```

Convenções de notação do C++

Este exemplo mostra como chamar métodos e declarar parâmetros.

This code samplEssa amostra de código usa os métodos e os parâmetros **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

Declare e use os parâmetros da seguinte forma:

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

...
}
```

Operações implícitas em C++

Várias operações podem ocorrer implicitamente, *no tempo exato*, para atender às condições de pré-requisito para a execução bem-sucedida de um método. Essas operações implícitas são conectar, abrir, reabrir, fechar e desconectar. É possível controlar o comportamento implícito de conectar e abrir usando atributos de classe.

Connect

Um objeto ImqQueueManager é conectado automaticamente para qualquer método que resulta em qualquer chamada para o MQI (consulte Referência cruzada C++ e MQI).

Abrir

Um objeto ImqObject é aberto automaticamente para qualquer método que resulta em uma chamada MQGET, MQINQ, MQPUT ou MQSET. Use o método **openFor** para especificar um ou mais valores de **opção aberta** relevante.

Reabrir

Um ImqObject é reaberto automaticamente para qualquer método que resulta em uma chamada MQGET, MQINQ, MQPUT ou MQSET, em que o objeto já está aberto, mas as **opções abertas** existentes não são adequadas para permitir que a chamada MQI seja bem-sucedida. O objeto é temporariamente fechado usando um valor de **opções fechadas** temporárias de MQCO_NONE. Use o método **openFor** para incluir um relevante**opções abertas**.

Reabrir pode causar problemas em circunstâncias específicas:

- Uma fila dinâmica temporária é destruída quando ela é fechada e nunca pode ser reaberta.
- Uma fila aberta para entrada exclusiva (seja explicitamente ou por padrão) pode ser acessada por outros na janela de oportunidade durante o fechamento e a reabertura.
- A posição do cursor de pesquisa é perdida quando uma fila está fechada. Esta situação não impede o fechamento e a reabertura, mas impede o uso subsequente do cursor até que MQGMO_BROWSE_FIRST seja usado novamente.
- O contexto da última mensagem recuperada é perdido quando uma fila está fechada.

Se qualquer uma destas circunstâncias ocorrer ou puder ser prevista, evite reaberturas configurando explicitamente **opções de abertura** adequadas antes que um objeto seja aberto (seja explícita ou implicitamente).

Configurar **opções abertas** explicitamente para situações de manipulação de filas complexas resulta em melhor desempenho e evita os problemas associados ao uso da reabertura.

Fechar

Um ImqObject é fechado automaticamente em qualquer ponto em que o estado do objeto não é mais viável, por exemplo, se uma referência de conexão ImqObject for interrompida ou se um objeto ImqObject for destruído.

Desconectar

Um ImqQueueManager é desconectado automaticamente em qualquer ponto em que a conexão não é mais viável, por exemplo, se uma referência de conexão ImqObject for severa ou se um objeto ImqQueueManager for destruído.

Sequências binárias e de caracteres em C++

A classe ImqString contém o formato de dados *char ** tradicional. A classe ImqBinary contém a matriz de bytes binários. Alguns métodos que configuram os dados de caracteres podem truncar os dados.

Os métodos que configuram os dados de caracteres (**char ***) sempre têm uma cópia dos dados, mas alguns métodos podem truncar a cópia, porque determinados limites são impostos pelo IBM MQ.

A classe ImqString (consulte a classe ImqString C++) contém o tradicional **char *** e fornece suporte para:

- Comparação
- Concatenação
- Copiando
- Conversão de número inteiro para texto e de texto para número inteiro
- Extração de token (palavra)
- Conversão de maiúscula

A classe ImqBinary (consulte a classe ImqBinary C++) contém as matrizes de bytes binários de tamanho arbitrário. Em particular, ela é usada para conter os seguintes atributos:

- **token de contabilidade** (MQBYTE32)
- **identificação da conexão** (MQBYTE128)
- **ID de correlação** (MQBYTE24)
- **token de recurso** (MQBYTE8)
- **ID de grupo** (MQBYTE24)
- **ID de instância** (MQBYTE24)
- **ID de mensagem** (MQBYTE24)
- **token de mensagem** (MQBYTE16)
- **ID de instância de transação** (MQBYTE16)

Em que estes atributos pertencem a objetos das seguintes classes:

- ImqCICSBridgeHeader (consulte a classe C++ ImqCICSBridgeHeader)
- ImqGetMessageOptions (consulte a classe C++ ImqGetMessageOptions)
- ImqIMSBridgeHeader (consulte a classe C++ ImqIMSBridgeHeader)
- ImqMessageTracker (consulte a classe C++ ImqMessageTracker)
- ImqQueueManager (consulte Classe C++ ImqQueueManager)
- ImqReferenceHeader (consulte a classe C++ ImqReferenceHeader)
- ImqWorkHeader (consulte a classe C++ ImqWorkHeader)

A classe ImqBinary também fornece suporte para comparação e cópia.

Funções não suportadas em C++

As classes e métodos IBM MQ C++ são independentes da plataforma IBM MQ. Eles podem, portanto, oferecer algumas funções que não sejam suportadas em certas plataformas.

Ao tentar usar uma função em uma plataforma na qual ela não é suportada, a função será detectada pelo IBM MQ, mas não pelas ligações de linguagem C++. O IBM MQ relata o erro para seu programa, como qualquer outro erro MQI.

Sistema de mensagens em C++

Esta coleção de tópicos detalha como preparar, ler e gravar mensagens em C++.

Preparando dados da mensagem em C++

Os dados da mensagem são preparados em um buffer, que pode ser fornecido pelo sistema ou aplicativo. Há vantagens para qualquer de um dos métodos. Exemplos de como usar um buffer são fornecidos.

Ao enviar uma mensagem, os dados da mensagem são preparados primeiro em um buffer gerenciado por um objeto ImqCache (consulte Classe ImqCache C++). Um buffer está associado (por herança) a cada objeto ImqMessage (consulte Classe ImqMessage C++): pode ser fornecido pelo aplicativo (usando o método **useEmptyBuffer** ou **useFullBuffer**) ou automaticamente pelo sistema. A vantagem do aplicativo que fornece o buffer de mensagem é que nenhuma cópia de dados é necessária em muitos casos porque o aplicativo pode usar áreas de dados preparadas diretamente. A desvantagem é que o buffer fornecido é de um comprimento fixo.

O buffer pode ser reutilizado e o número de bytes transmitidos pode ser ativado de cada vez, usando o método **setMessageLength** antes da transmissão.

Quando fornecido automaticamente pelo sistema, o número de bytes disponíveis é gerenciado pelo sistema e os dados podem ser copiados para o buffer de mensagem usando, por exemplo, o método **wriite** ImqCache ou o método **writeItem** ImqMessage. O buffer de mensagem aumenta de acordo com a necessidade. Conforme o buffer aumentar, não há perda de dados gravados anteriormente. Uma mensagem grande ou multipartes pode ser gravada em partes sequenciais.

Os exemplos a seguir mostram envios da mensagem simplificada.

1. Use os dados preparados em um buffer fornecido pelo usuário

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. Use os dados preparados em um buffer fornecido pelo usuário, em que o tamanho do buffer excede o tamanho de dados

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. Copiar dados para um buffer fornecido pelo usuário

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. Copiar dados para um buffer fornecido pelo sistema

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. Copiar dados para um buffer fornecido pelo sistema usando os objetos (objetos configuram o formato da mensagem, bem como o conteúdo)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

Lendo mensagens em C++

Um buffer pode ser fornecido pelo aplicativo ou pelo sistema. Os dados podem ser acessados diretamente do buffer ou lidos sequencialmente. Há uma classe equivalente a cada tipo de mensagem. Código de amostra é fornecido.

Ao receber dados, o aplicativo ou o sistema pode fornecer um buffer de mensagem adequado. O mesmo buffer pode ser usado para diversas transmissões e diversos recebimentos para um objeto ImqMessage

específico. Se o buffer de mensagem for fornecido automaticamente, ele aumenta para acomodar o comprimento de dados recebido. No entanto, um buffer de mensagem fornecido pelo aplicativo pode não ser grande o suficiente para conter os dados recebidos. Em seguida, truncamento ou falha pode ocorrer, dependendo das opções usadas para o recebimento de mensagens.

Dados de entrada podem ser acessados diretamente do buffer de mensagem, nesse caso, o comprimento dos dados indica a quantia total de dados recebidos. Como alternativa, dados recebidos podem ser lidos sequencialmente a partir do buffer de mensagem. Neste caso, o ponteiro de dados endereça o próximo byte de dados recebidos e o ponteiro de dados e o comprimento de dados são atualizados cada vez que os dados são lidos.

Itens são partes de uma mensagem, todos na área do usuário do buffer de mensagem, que precisam ser processados sequencialmente e separadamente. Além de dados do usuário regulares, um item pode ser um cabeçalho de mensagens não entregues ou uma mensagem do acionador. Os itens são sempre associados aos formatos de mensagem; os formatos de mensagem **nem** sempre são associados a itens.

Há uma classe de objeto para cada item que corresponde a um formato de mensagem reconhecível do IBM MQ. Há uma para um cabeçalho de mensagens não entregues e uma para uma mensagem do acionador. Não há classe de objeto para dados do usuário. Ou seja, quando os formatos reconhecíveis estiverem esgotados, o processamento do restante será deixado para o programa de aplicativo. Classes de dados do usuário podem ser escritas especializando a classe ImqItem.

O exemplo a seguir mostra o recebimento de uma mensagem que leva em consideração diversos itens em potencial que podem preceder os dados do usuário, em uma situação imaginária. Os dados do usuário não de item são definidos como tudo o que ocorre após itens que podem ser identificados. Um buffer automático (o padrão) é usado para conter uma quantia arbitrária de dados da mensagem.

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
```

```
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

Neste exemplo, FMT_USERCLASS é uma constante que representa o nome do formato de 8 caracteres associada a um objeto de classe `UserClass` e definida pelo aplicativo.

`UserClass` é derivada da classe ImqItem (consulte Classe C++ ImqItem) e implementa os métodos virtuais **copyOut** e **pasteIn** dessa classe.

Os dois próximos exemplos mostram o código da classe ImqDeadLetterHeader (consulte Classe C++ ImqDeadLetterHeader). O primeiro exemplo mostra código contido customizado para *gravar* mensagem.

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
```

```
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }

  return bSuccess ;
}
```

O segundo exemplo mostra código contido customizado para *ler* mensagem.

```
// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

Com um buffer automático, o armazenamento em buffer é *volátil*. Ou seja, os dados do buffer podem ser mantidos em um local físico diferente após cada chamada do método **get**. Portanto, toda vez que o buffer de dados é referido, use os métodos **bufferPointer** ou **dataPointer** para acessar dados da mensagem.

Você pode desejar que um programa separe uma área fixa para receber dados da mensagem. Nesse caso, chame o método **useEmptyBuffer** antes de usar o método **get**.

Usar uma área fixa, não automática, limita as mensagens a um tamanho máximo, portanto, é importante considerar a opção MQGMO_ACCEPT_TRUNCATED_MSG do objeto ImqGetMessageOptions. Se essa opção não for especificada (o padrão), o código de razão MQRC_TRUNCATED_MSG_FAILED pode ser esperado. Se essa opção for especificada, o código de razão MQRC_TRUNCATED_MSG_ACCEPTED pode ser esperado, dependendo do design do aplicativo.

O próximo exemplo mostra como uma área fixa de armazenamento pode ser usada para receber mensagens:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
```

```
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

Nesse fragmento de código, o buffer pode sempre ser direcionado diretamente, com *pszBuffer*, em vez de usar o método **bufferPointer**. No entanto, é melhor usar o método **dataPointer** para acesso de propósito geral. O aplicativo (não o objeto de classe ImqCache) deve descartar um buffer definido pelo usuário (nonautomatic).

Atenção: especificar um ponteiro nulo e comprimento zero com **useEmptyBuffer** não denomina um buffer de comprimento fixo com comprimento zero como pode ser esperado. Essa combinação é interpretada como uma solicitação para ignorar qualquer buffer anterior definido pelo usuário e, em vez disso, reverter para o uso de um buffer automático.

Gravando uma mensagem na fila de mensagens não entregues em C++

Código do programa de exemplo para gravar uma mensagem na fila de mensagens não entregues.

Um caso típico de uma mensagem multipartes é um que contém um cabeçalho de mensagens não entregues. Os dados de uma mensagem que não podem ser processados são anexados ao cabeçalho da fila de devoluções.

```
ImqQueueManager mgr ;        // The queue manager.
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

Gravando uma mensagem na ponte IMS em C++

Código do programa de exemplo para gravar uma mensagem na ponte IMS.

As mensagens enviadas à ponte IBM MQ - IMS podem usar um cabeçalho especial. O cabeçalho da ponte IMS tem como prefixo dados da mensagem regular.

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
```

```
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

Gravando uma mensagem no CICS bridge em C++

Código do programa de exemplo para gravar uma mensagem no CICS bridge.

As mensagens enviadas para o IBM MQ for z/OS usando o CICS bridge requerem um cabeçalho especial. O cabeçalho do CICS bridge tem como prefixo dados da mensagem regular.

```
ImqQueueManager mgr ;         // The queue manager.
ImqQueue queueIn ;            // Incoming message queue.
ImqQueue queueBridge ;        // CICS bridge message queue.
ImqMessage msg ;              // Incoming and outgoing message.
ImqCicsBridgeHeader header ;  // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

Gravando uma mensagem com um cabeçalho de trabalho em C++

Código do programa de exemplo para gravar uma mensagem destinada para uma fila gerenciada pelo z/OS Workload Manager.

As mensagens enviadas para IBM MQ for z/OS, que são destinadas a uma fila gerenciada pelo z/OS Workload Manager, requerem um cabeçalho especial. O cabeçalho do trabalho tem como prefixo dados da mensagem regular.

```
ImqQueueManager mgr ;             // The queue manager.
```

```
ImqQueue queueIn ;                // Incoming message queue.
ImqQueue queueWLM ;               // WLM managed queue.
ImqMessage msg ;                  // Incoming and outgoing message.
ImqWorkHeader header ;            // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

Construindo programas C++ IBM MQ

A URL de compiladores suportados está listada, juntamente com os comandos a serem usados para compilar, vincular e executar programas C++ e amostras em plataformas IBM MQ.

Para obter uma lista dos compiladores para cada plataforma e versão suportada do IBM MQ, veja Requisitos do sistema para IBM MQ.

O comando que você precisa compilar e vincular a seu programa IBM MQ C++ depende de sua instalação e requisitos. Os exemplos a seguir mostram comandos típicos de compilação e vinculação para alguns dos compiladores usando a instalação padrão do IBM MQ em várias plataformas.

AIX Construindo programas C++ no AIX

Desenvolva programas IBM MQ C++ no AIX usando o compilador XL C Enterprise Edition.

V 9.4.0 Para obter mais informações sobre o mapeamento diferente de opções do compilador entre os compiladores XLC 16 e XLC 17, consulte Mapeamento de opções

Deprecated V 9.4.0 O suporte para o XL C/C++ para AIX 16 compilador em AIX foi descontinuado de IBM MQ 9.4.0.

Client

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo não encadeado de 32 bits

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

Aplicativo encadeado de 32 bits

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Aplicativo não encadeado de 64 bits

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Aplicativo encadeado de 64 bits

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.4.0 **Aplicativo não encadeado de 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.4.0 **Aplicação encadeada de 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.4.0 **Aplicativo sem encadeamento de 64 bits (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.4.0 **Aplicativo encadeado de 64 bits (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

Servidor

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo não encadeado de 32 bits

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

Aplicativo encadeado de 32 bits

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Aplicativo não encadeado de 64 bits

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Aplicativo encadeado de 64 bits

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.4.0 **Aplicativo não encadeado de 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.4.0 **Aplicação encadeada de 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.4.0 Aplicativo sem encadeamento de 64 bits (XLC 17)

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.4.0 Aplicativo encadeado de 64 bits (XLC 17)

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i Construindo programas C++ no IBM i

Construção dos programas C++ do IBM MQ no IBM i usando o compilador ILE C++.

IBM ILE C++ for IBM i é um compilador nativo para programas C++. As instruções a seguir descrevem como usar este compilador para criar aplicativos C++ IBM MQ usando o *Hello World!* Programa de amostra do IBM MQ como um exemplo.

1. Instale o compilador ILE C++ for IBM i, conforme indicado no *Leia-me primeiro!* manual que acompanha o produto.
2. Assegure-se de que a biblioteca QCXXN esteja em sua lista de bibliotecas.
3. Crie o programa de amostra HELLO WORLD:
   a. Crie um módulo:

   ```
   CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
   SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
   INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
   TERASPACE(*YES)
   ```

   A origem para os programas de amostra C++ pode ser localizada em `/QIBM/ProdData/mqm/samp` e os arquivos de inclusão em `/QIBM/ProdData/mqm/inc`.

   Como alternativa, a origem pode ser localizada na biblioteca `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)`.

   b. Ligue isso com os programas de serviço fornecidos pelo IBM MQ para produzir um objeto de programa:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
   ```

   Para construir um aplicativo encadeado, use os programas de serviço de reentrante:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
   ```

   c. Execute o programa de amostra HELLO WORLD usando SYSTEM.DEFAULT.LOCAL.QUEUE:

   ```
   CALL PGM(MYLIB/IMQWRLD)
   ```

Linux Construindo programas C++ no Linux

Construa programas C++ do IBM MQ no Linux usando o compilador GNU g++.

System p

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: System p

Aplicativo não encadeado de 32 bits

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Aplicativo encadeado de 32 bits

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

Aplicativo não encadeado de 64 bits

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicativo encadeado de 64 bits

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Servidor: System p

Aplicativo não encadeado de 32 bits

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Aplicativo encadeado de 32 bits

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

Aplicativo não encadeado de 64 bits

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Aplicativo encadeado de 64 bits

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: IBM Z

Aplicativo não encadeado de 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Aplicativo encadeado de 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

Aplicativo não encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicativo encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Servidor: IBM Z

Aplicativo não encadeado de 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Aplicativo encadeado de 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Aplicativo não encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Aplicativo encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 bits)

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: x86-64 (32 bits)

Aplicativo não encadeado de 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

Aplicativo encadeado de 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Aplicativo não encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

Aplicativo encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Servidor: x86-64 (32 bits)

Aplicativo não encadeado de 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

Aplicativo encadeado de 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Aplicativo não encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Aplicativo encadeado de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows Construindo programas C++ no Windows

Construa programas C++ do IBM MQ no Windows usando o compilador C++ do Microsoft Visual Studio.

Atenção: As bibliotecas fornecidas pelo IBM MQ são bibliotecas dinâmicas e não são bibliotecas estáticas. IBM MQ fornece algo conhecido como "`import libraries`" que você pode usar durante o tempo de compilação apenas. Para tempo de execução, você deve usar as bibliotecas dinâmicas.

A partir de IBM MQ 8.0.0 Fix Pack 4, IBM MQ envia clientes redistribuíveis, contendo bibliotecas necessárias para executar aplicativos IBM MQ . Essas bibliotecas podem ser empacotadas e redistribuídas com aplicativos clientes. Para obter mais informações, consulte Clientes redistribuíveis no Windows.

Arquivos de biblioteca (.lib) e arquivos dll para uso com aplicativos de 32 bits são instalados no *MQ_INSTALLATION_PATH*`/Tools/Lib`. Arquivos para uso com aplicativos de 64 bits são instalados em *MQ_INSTALLATION_PATH*`/Tools/Lib64`. O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Client

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

Servidor

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Instalando o Tempo de Execução Universal C

Quando você está usando o Windows 8.1 ou o Windows Server 2012 R2, deve-se instalar a atualização universal C runtime update (Universal CRT) por meio da Microsoft. Esse tempo de execução é incluído como parte do Windows 10 e do Windows Server 2016.

A atualização Universal CRT é a atualização da Microsoft KB3118401. É possível verificar se você tem essa atualização procurando por um arquivo chamado `ucrtbase.dll` no diretório `C:\Windows\System32`. Caso contrário, é possível fazer download da atualização da página da Microsoft: https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401.

A tentativa de executar um programa IBM MQ ou um programa que você compila sozinho usando o Microsoft Visual Studio 2017 sem o tempo de execução instalado, resulta em erros, como o seguinte:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Fornecendo tempos de execução para programas Microsoft Visual Studio 2012

Se você compilou um programa IBM MQ usando o Microsoft Visual Studio 2012, esteja ciente de que o instalador do IBM MQ não instala os tempos de execução C/C++ do Microsoft Visual Studio 2012. Se a versão anterior do IBM MQ foi instalada no mesmo computador, os tempos de execução do Microsoft Visual Studio 2012 estarão disponíveis nessa instalação.

No entanto, se você está usando um programa que foi construído usando o Microsoft Visual Studio 2012 e nenhuma versão anterior do IBM MQ foi instalada, deve-se executar uma das seguintes ações:

- Faça download e instale o **`Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)`** de Microsoft.
- Recompilar seu programa com o Microsoft Visual Studio 2017 ou outro nível do Microsoft Visual Studio para o qual os tempos de execução são instalados.

Bibliotecas de clientes C++ construídas usando o compilador do Microsoft Visual Studio 2015

O IBM MQ fornece bibliotecas do cliente C++ que são construídas com o compilador C++ do Microsoft Visual Studio 2015 e o compilador C++ do Microsoft Visual Studio 2017.

Ambas as versões de 32 bits e de 64 bits das bibliotecas C++ do IBM MQ são fornecidas. As bibliotecas de 32 bits estão instaladas sob a pasta `bin\vs2015` e as bibliotecas de 64 bits estão instaladas sob as pastas `bin64\vs2015`.

Por padrão, o IBM MQ está configurado para usar as bibliotecas do Microsoft Visual Studio 2017. Para usar as bibliotecas do Microsoft Visual Studio 2015, deve-se configurar a variável de ambiente `MQ_PREFIX_VS_LIBRARIES` como `MQ_PREFIX_VS_LIBRARIES=vs2015` antes de instalar o IBM MQ ou antes de usar o comando **setmqenv** ou **setmqinst**.

Usando bibliotecas C++ do IBM MQ denominadas de forma diferente

O IBM MQ fornece algumas bibliotecas adicionais do cliente C++ que são nomeadas de forma diferente. Essas bibliotecas são construídas com os compiladores C++ do Microsoft Visual Studio 2015 e do Microsoft Visual Studio 2017. Essas bibliotecas são fornecidas além das bibliotecas C++ existentes que também são construídas com o compilador C++ do Microsoft Visual Studio 2017. Como essas bibliotecas C++ adicionais do IBM MQ possuem nomes diferentes, é possível executar aplicativos C++ do IBM MQ que são construídos usando o IBM MQ C++ e compilados com o Microsoft Visual Studio 2017 e versões anteriores do produto no mesmo computador.

As bibliotecas adicionais do Microsoft Visual Studio 2017 possuem os nomes a seguir:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

As bibliotecas adicionais do Microsoft Visual Studio 2015 possuem os nomes a seguir:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

Ambas as versões de 32 bits e de 64 bits dessas bibliotecas são fornecidas. As bibliotecas de 32 bits estão instaladas sob a pasta `bin` e as bibliotecas de 64 bits estão instaladas sob a pasta `bin64`. As bibliotecas de importação correspondentes estão instaladas sob os diretórios `Tools\lib` e `Tools\lib64`.

Se o aplicativo usar arquivos `imq*vs2015.lib`, você deverá compilá-lo usando o compilador Microsoft Visual Studio 2015. Para executar aplicativos C++ do IBM MQ que são compilados com o Microsoft Visual Studio 2015 ou aplicativos que são compilados com uma versão anterior do produto no mesmo computador, a variável de ambiente PATH deve ser prefixada conforme mostrado nos exemplos a seguir:

- Para aplicativos de 32 bits:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- Para aplicativos de 64 bits:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

Conceitos relacionados

Windows: mudanças do IBM MQ 8.0

z/OS Building C++ programs on z/OS Batch, RRS Batch and CICS

Build IBM MQ C++ programs on z/OS for the Batch, RRS batch or CICS environments and run the sample programs.

You can write C++ programs for three of the environments that IBM MQ for z/OS supports:

- Batch
- RRS batch
- CICS

Compile, prelink and link

Create an z/OS application by compiling, pre-linking, and link-editing your C++ source code.

IBM MQ C++ for z/OS is implemented as z/OS DLLs for the IBM C++ for z/OS language. Using DLLs, you concatenate the supplied definition sidedecks with the compiler output at pre-link time. This allows the linker to check your calls to the IBM MQ C++ member functions.

Note: There are three sets of sidedecks for each of the three environments.

To build an IBM MQ for z/OS C++ application, create and run JCL. Use the following procedure:

1. If your application runs under CICS, use the CICS-supplied procedure to translate CICS commands in your program.

   In addition, for CICS applications you need to:

   a. Add the SCSQLOAD library to the DFHRPL concatenation.

   b. Define the CSQCAT1 CEDA group using the member IMQ4B100 in the SCSQPROC library.

   c. Install CSQCAT1.
2. Compile the program to produce object code. The JCL for your compilation must include statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Be sure to specify the `/cxx` compiler option.

   Note: The name **thlqual** is the high level qualifier of the IBM MQ installation library on z/OS.
3. Pre-link the object code created in step “2” on page 563, including the following definition sidedecks, which are supplied in **thlqual**.SCSQDEFS:

   a. imqs23dm and imqb23dm for batch

   b. imqs23dr and imqb23dr for RRS batch

   c. imqs23dc and imqb23dc for CICS

   These are the corresponding DLLs.

   a. imqs23im and imqb23im for batch

   b. imqs23ir and imqb23ir for RRS batch

   c. imqs23ic and imqb23ic for CICS
4. Link-edit the object code created in step “3” on page 563, to produce a load module, and store it in your application load library.

To run batch or RRS batch programs, include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB or JOBLIB data set concatenation.

To run a CICS program, first get your system administrator to define it to CICS as an IBM MQ program and transaction. You can then run it in the usual way.

Run the sample programs

The programs are described in “Programas de amostra C++” on page 540.

The sample applications are supplied in source form only. The files are:

Table 75. z/OS sample program files

Sample	Source program (in library thlqual.SCSQCPPS)	JCL (in library thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

To run the samples, compile and link-edit them as with any C++ program (see “Building C++ programs on z/OS Batch, RRS Batch and CICS” on page 563). Use the supplied JCL to construct and run a batch job. You must initially customize the JCL, by following the commentary included with it.

z/OS Building C++ programs on z/OS UNIX System Services

Build IBM MQ C++ programs on z/OS UNIX System Services (z/OS UNIX).

To build an application under the z/OS UNIX shell, you must give the compiler access to the IBM MQ include files (located in `thlqual.SCSQC370` and `hlqual.SCSQHPPS`), and link against two of the DLL sidedecks (located in `thlqual.SCSQDEFS`). At runtime, the application needs access to the IBM MQ data sets `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`, and one of the language specific data sets, such as `thlqual.SCSQANLE` [6].

Compiling

1. Copy the sample into the file system using the TSO **oput** command, or use FTP. The rest of this example assumes that you have copied the sample into a directory called `/u/fred/sample`, and named it `imqwrld.cpp`.
2. Log into the z/OS UNIX shell, and change to the directory where you placed the sample.
3. Set up the C++ compiler so that it can accept the DLL sidedeck and .cpp files as input:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Compile and link the sample program. The following command links the program with the batch sidedecks; the RRS batch sidedecks can be used instead. The \ character is used to split the command over more than one line. Do not enter this character; enter the command as a single line:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

For more information on the TSO **oput** command, refer to the z/OS UNIX Command Reference.

You can also use the make utility to simplify building C++ programs. Here is a sample makefile to build the HELLO WORLD C++ sample program. It separates the compiling and linking stages. Set up the environment as in step “3” on page 564 before running make.

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

[6] You can link with any of the sidedecks listed in "Pre-link the object code to run your z/OS UNIX in any of the three environments, “Building C++ programs on z/OS Batch, RRS Batch and CICS” on page 563

```
imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Refer to z/OS UNIX System Services Programming Tools for more information on using make.

Running

1. Log into the z/OS UNIX shell, and change to the directory where you built the sample.
2. Set up the STEPLIB environment variable to include the IBM MQ data sets:

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. Run the sample:

```
/u/fred/sample:> ./imqwrld
```

Desenvolvendo aplicativos do .NET

IBM MQ classes for .NET permitir que .NET aplicativos se conectem ao IBM MQ como um IBM MQ MQI client ou se conectem diretamente a um servidor IBM MQ .

Antes de começar

Deprecated V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, em IBM MQ classes for .NET, os métodos WriteObject(), ReadObject(), CreateObjectMessage () e as classes ObjectMessage e XmsObjectMessageImpl usados para serialização e desserialização de dados foram descontinuados.

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do IBM MQ .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

Sobre esta tarefa

O IBM MQ classes for .NET é um conjunto de classes que permite que aplicativos .NET interajam com o IBM MQ. Elas representam os vários componentes do IBM MQ que seu aplicativo usa, como gerenciadores de filas, filas, canais e mensagens. Para obter mais informações sobre essas classes, consulte As classes e interfaces do IBM MQ .NET

V 9.4.0 IBM MQ 9.4.0 fornece uma biblioteca do cliente IBM MQ .NET construída com relação a .NET 6 como a estrutura de destino. Para obter mais informações, consulte “Instalando IBM MQ classes for .NET” na página 566.

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for .NET. Para obter mais informações, consulte “Instalando IBM MQ classes for .NET” na página 566.

Se você tiver aplicativos que usam o Microsoft .NET Framework e deseja aproveitar os recursos do IBM MQ, deve usar o IBM MQ classes for .NET Framework Para obter mais informações, consulte “Instalando IBM MQ classes for .NET Framework” na página 572.

Para obter mais informações sobre as diferenças entre IBM MQ classes for .NET Framework e IBM MQ classes for .NET, consulte “Instalando IBM MQ classes for .NET” na página 566.

Os aplicativos gerenciados do IBM MQ .NET podem balancear automaticamente as conexões entre os gerenciadores de filas em cluster. As bibliotecas IBM MQ classes for .NET e IBM MQ classes

for .NET Framework são suportadas. Para obter mais informações, consulte Sobre clusters uniformes e Balanceamento automático de aplicativo.

A interface IBM MQ .NET orientada por objeto é diferente da interface MQI no sentido de que usa métodos de objetos, em vez de usar os verbos MQI. A interface de programação do aplicativo IBM MQ processual é construída em torno de verbos como os da lista a seguir:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Todos esses verbos aceitam, como um parâmetro, um manipulador para o objeto do IBM MQ no qual devem operar. Uma vez que o .NET é orientado a objetos, a interface de programação do .NET muda isso. Seu programa consiste em um conjunto de objetos do IBM MQ, que é acionado ao chamar métodos nestes objetos. É possível gravar programas em qualquer idioma suportado por .NET.

Quando você usa a interface processual, desconecta-se de um gerenciador de filas usando a chamada MQDISC(*Hconn*, CompCode, Reason), em que *Hconn* é um identificador para o gerenciador de filas. Na interface do .NET, o gerenciador de filas é representado por um objeto de classe MQQueueManager. Desconecte-se do gerenciador de filas chamando o método disconnect() nessa classe.

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

Conceitos relacionados

Visão geral técnica

“Desenvolvendo aplicativos para o IBM MQ” na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

Tarefas relacionadas

Entrando em Contato com o IBM Support

Resolução de problemas do IBM MQ .NET

“Desenvolvendo aplicativos Microsoft Windows Communication Foundation com o IBM MQ” na página 1279
O canal customizado do Microsoft Windows Communication Foundation (WCF) para IBM MQ envia e recebe mensagens entre clientes e serviços WCF.

“Desenvolvendo aplicativos do XMS .NET” na página 625
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) fornece uma interface de programação de aplicativos (API) chamada XMS que possui o mesmo conjunto de interfaces que a API Java Message Service (JMS). O IBM MQ Message Service Client (XMS) for .NET contém uma implementação totalmente gerenciada do XMS, que pode ser usada por qualquer idioma compatível com o .NET.

Windows Linux

Instalando IBM MQ classes for .NET

IBM MQ classes for .NET, incluindo amostras, são instalados com IBM MQ em Windows e Linux

Pré-requisitos e Instalação

V 9.4.0 IBM MQ 9.4.0 fornece uma biblioteca do cliente do IBM MQ .NET construída em .NET 6 como a estrutura de destino. No IBM MQ 9.4.0, Microsoft .NET 6.0 é a versão mínima necessária para executar aplicativos usando bibliotecas do IBM MQ que são construídas usando o .NET 6 como a estrutura de destino. A biblioteca do cliente do IBM MQ .NET construída usando .NET 6

como a estrutura de destino está disponível em *MQ_INSTALLATION_PATH*/bin em Windows e em *MQ_INSTALLATION_PATH*/lib64 em Linux

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for .NET. Se você estiver usando um aplicativo .NET 6 , será possível executar esse aplicativo sem que qualquer recompilação seja necessária, fazendo uma pequena edição no arquivo `runtimeconfig` para configurar o `targetframeworkversion` como `"net8.0"`

Deprecated V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, em IBM MQ classes for .NET, os métodos WriteObject(), ReadObject(), CreateObjectMessage () e as classes ObjectMessage e XmsObjectMessageImpl usados para serialização e desserialização de dados foram descontinuados.

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do IBM MQ .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

A versão mais recente do IBM MQ classes for .NET é instalada por padrão como parte da instalação padrão do IBM MQ no recurso *Java and .NET Messaging and Web Services* .

Windows Para obter mais informações sobre pré-requisitos e instalação no Windows:

- Consulte Requisitos para IBM MQ classes for .NET para o software obrigatório executar IBM MQ classes for .NET.
- Consulte Instalando o IBM MQ servidor no Windows ou Instalando um cliente IBM MQ em sistemas Windows para instruções de instalação.

Linux Para obter mais informações sobre pré-requisitos e instalação no Linux:

- Consulte Requisitos para IBM MQ classes for .NET para o software obrigatório executar IBM MQ classes for .NET.
- Para obter as instruções de instalação do rpm, consulte Instalando um cliente IBM MQ em sistemas Linux.
- Para Ubuntu Linux, usando pacotes Debian, veja Instalando um cliente IBM MQ em sistemas Linux.

A biblioteca IBM MQ classes for .NET Standard, `amqmdnetstd.dll`, está disponível para download a partir do repositório NuGet. Para obter mais informações, consulte “Fazendo download de IBM MQ classes for .NET a partir do repositório NuGet” na página 571.

Biblioteca do `amqmdnetstd.dll`

V 9.4.0 V 9.4.0 A partir do IBM MQ 9.4.0, a biblioteca `amqmdnetstd.dll` construída usando .NET 6 como a estrutura de destino está disponível nos locais a seguir:

- Windows Em Windows: *MQ_INSTALLATION_PATH*\bin. Os aplicativos de amostra estão instalados em *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base
- Linux Em Linux: *MQ_INSTALLATION_PATH*\lib64. As amostras do .NET estão no *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base

Atenção: V 9.4.0 Removed V 9.4.0 De IBM MQ 9.4.0, IBM MQ .NET bibliotecas do cliente construídas usando .NET Standard 2.0 como a estrutura de destino são removidas. Essas bibliotecas foram descontinuadas em IBM MQ 9.3.1,

Stabilized LTS A biblioteca `amqmdnet.dll` para .NET Framework ainda é fornecida, mas esta biblioteca está estabilizada; ou seja, nenhum novo recurso será introduzido nela. Para qualquer um dos recursos mais recentes deve-se migrar para a biblioteca `amqmdnetstd.dll`. No entanto, é possível continuar usando a biblioteca `amqmdnet.dll` em liberações IBM MQ 9.1 ou posteriores Long Term Support ou Continuous Delivery .

V 9.4.0 V 9.4.0 A seguir estão dois cenários que podem ser encontrados após a remoção das bibliotecas `netstandard2.0` :

- Se você estiver usando um aplicativo IBM MQ classes for .NET Framework construído usando as bibliotecas `netstandard2.0` , como `amqmdnetstd.dll`, será necessário reconstruir seu aplicativo com as bibliotecas Microsoft.NET Framework 4.7.2 , como `amqmdnet.dll`, para que seu aplicativo seja executado com êxito. Se você não reconstruir seu aplicativo, poderá obter um System.IO.Unexceptionable não excepcional:

  ```
  Exceção capturada: System.IO.FileLoadException: Não foi possível carregar o arquivo ou conjunto
  'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' ou uma de suas
  dependências. A definição de manifest do conjunto localizado não corresponde à referência de
  conjunto (Exceção de HRESULT: 0x80131040)
  Nome do arquivo: 'amqmdnetstd, Version=9.3.5.0, Culture=neutral,
  PublicKeyToken=23d6cb914eeaac0e'
     em SimplePut.SimplePut.PutMessages()
     em SimplePut.SimplePut.Principal (String [] args) em C:\SampleCode\Program.cs:line 132
  ```

- Se você estiver usando um aplicativo .NET 6 que é construído usando bibliotecas `netstandard2.0` , será necessário apenas substituir essas bibliotecas pelas mesmas bibliotecas .NET 6 na pasta `bin` do diretório de tempo de execução do aplicativo. Nenhuma reconstrução é necessária.

  Nota: A biblioteca .NET 6 de substituição deve sempre ser do mesmo nível ou superior que a biblioteca `netstandard2.0` substituída

Comando dspmqver

É possível usar o comando **dspmqver** para exibir as informações de versão e de construção para o componente .NET Core.

Comparação de recurso entre IBM MQ classes for .NET Framework e IBM MQ classes for .NET

A tabela a seguir lista os recursos para IBM MQ classes for .NET Framework comparados com os recursos para IBM MQ classes for .NET

Tabela 76. Diferenças entre IBM MQ classes for .NET Framework e IBM MQ classes for .NET .

Recurso	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
Nomes de classes (APIs)	Todas as classes permanecem as mesmas em cada rede.	Todas as classes permanecem as mesmas em cada rede.
Sistema Operacional	Windows	Windows Contêineres Docker Linux macOS
Arquivo `app.config` (arquivo de configuração para ativar o Rastreio no cliente redistribuível)	O arquivo `app.config` é usado para ativar o rastreio para o pacote redistribuível e o cliente IBM MQ .NET independente. Consulte Rastreando um cliente IBM MQ classes for .NET Framework usando um arquivo de configuração de aplicativo para obter mais informações sobre as variáveis usadas para rastreio, incluindo **MQTRACEPATH** e **MQTRACELEVEL**.	`app.config` não será suportada. Use as variáveis de ambiente

Tabela 76. Diferenças entre IBM MQ classes for .NET Framework e IBM MQ classes for .NET . (continuação)

Recurso	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
Trace	Para uma instalação do cliente completa do IBM MQ, é possível usar o comando **strmqtrc** para ativar o rastreamento para IBM MQ classes for .NET Framework Para clientes redistribuíveis, o arquivo `app.config` também é usado para ativar o rastreio.. Para obter mais informações, consulte Rastreio IBM MQ .NET aplicativos.. V 9.4.0 Em IBM MQ 9.4.0, é possível ativar e desativar o rastreio usando o arquivo `mqclient.ini` e configurando as propriedades apropriadas da sub-rotina Trace. Também é possível ativar e desativar o rastreio dinamicamente com o arquivo `mqclient.ini` .. Para obter mais informações, consulte Rastreio IBM MQ .NET de aplicativos com mqclient.ini	A variável de ambiente **MQDOTNET_TRACE_ON** é usada para ativar o rastreio para clientes redistribuíveis. Valores iguais e menores que 0 não ativam o rastreio. Um valor de 1 ativa o rastreio de nível padrão. Um valor maior que 1 ativa o rastreio detalhado. A configuração dessa variável de ambiente para qualquer outro valor como sequência não ativa o rastreio. Consulte Rastreio IBM MQ .NET de aplicativos usando variáveis de ambiente A variável de ambiente **MQDOTNET_TRACE_ON** verifica se o diretório de rastreio IBM MQ está ou não disponível. Se o diretório de rastreio estiver disponível, o arquivo de rastreio será gerado no diretório de rastreio. Entretanto, se IBM MQ não estiver instalado, o arquivo de rastreio será copiado para o diretório atualmente em funcionamento. Outras variáveis de ambiente, incluindo **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**, e assim por diante, que são usadas para IBM MQ classes for .NET Framework, podem ser usadas e trabalhar da mesma maneira V 9.4.0 Em IBM MQ 9.4.0, é possível ativar e desativar o rastreio usando o arquivo `mqclient.ini` e configurando as propriedades apropriadas da sub-rotina Trace. Também é possível ativar e desativar o rastreio dinamicamente com o arquivo `mqclient.ini` .. Para obter mais informações, consulte Rastreio IBM MQ .NET de aplicativos com mqclient.ini
Modos de transporte	Gerenciado, Não gerenciado e Ligações	Gerenciado

Tabela 76. Diferenças entre IBM MQ classes for .NET Framework e IBM MQ classes for .NET . (continuação)

Recurso	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
TLS	O keystore Windows é usado para armazenar os certificados.	Windows No Windows, o keystore deve ser usado para armazenar os certificados. Os valores permitidos são *USER ou *SYSTEM. Com base na entrada, o cliente do IBM MQ .NET consulta o keystore do Windows do usuário atual ou todo o sistema. Linux No Linux, recomenda-se usar a classe X509Store para instalar certificados e o .NET Core instala certificados no local a seguir: "`.dotnet/corefx/cryptography/x509stores`".
CCDT	Suportado	Suportado, e as configurações do caminho CCDT são iguais às para as classes do .NET Framework.
Reconexão automática do cliente	Suportado	Suportado
Transações distribuídas	Suportado	Não Suportado
Instalação de bibliotecas de vínculo dinâmico (dlls) no cache de montagem global (GAC)	As Dlls são instaladas no GAC como parte da instalação do IBM MQ.	As Dlls não são instaladas no GAC como parte da instalação do IBM MQ.

Nota: Windows Identificadores de segurança do Windows (SIDs):

A autenticação de nível de domínio não é suportada para IBM MQ classes for .NET (.NET Standard e .NET 6 bibliotecas). O ID do usuário com login efetuado é usado para autenticação.

Desenvolvendo aplicativos IBM MQ .NET Core no macOS

macOS

Os aplicativos IBM MQ .NET Core podem ser desenvolvidos no macOS.

As bibliotecas IBM MQ .NET não são empacotadas com o kit de ferramentas macOS, por isso, você deve copiá-las de um cliente Windows ou Linux IBM MQ para macOS. Em seguida, é possível usar essas bibliotecas para desenvolver os aplicativos IBM MQ .NET Core no macOS.

Uma vez desenvolvidos, esses aplicativos podem ser executados, suportados em ambientes Windows ou Linux.

Conceitos relacionados

“Instalando IBM MQ classes for .NET Framework” na página 572
IBM MQ classes for .NET Framework, incluindo amostras, são instalados com IBM MQ. Há um pré-requisito do Microsoft.NET Framework no Windows.

“Instalando IBM MQ classes for XMS .NET” na página 630
O IBM MQ classes for XMS .NET, incluindo amostras, é instalado com IBM MQ em Windows e Linux

Windows Linux Fazendo download de IBM MQ classes for .NET a partir do repositório NuGet

O IBM MQ classes for .NET está disponível para download a partir do repositório do NuGet , para que eles possam ser facilmente consumidos pelos Desenvolvedores do .NET

Sobre esta tarefa

NuGet é o gerenciador de pacotes para plataformas de desenvolvimento da Microsoft, incluindo o .NET. As ferramentas do cliente do NuGet fornecem a capacidade de produzir e consumir pacotes. Um pacote NuGet é um único arquivo compactado com a extensão `.nupkg` que contém código compilado (DLLs), outros arquivos relacionados a aquele código e um manifesto descritivo que inclui informações como o número da versão do pacote.

Você pode fazer o download do pacote `IBMMQDotnetClient` NuGet, que contém a biblioteca `amqmdnetstd.dll`, da Galeria NuGet, que é o repositório de pacotes central usado por todos os autores e consumidores de pacotes.

Nota: V 9.4.0 V 9.4.0 No IBM MQ 9.4.0, o pacote NuGet contém bibliotecas construídas usando .NET 6 como a estrutura de destino.

Removed A biblioteca do cliente do IBM MQ .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for .NET. Se você estiver usando um aplicativo .NET 6 , será possível executar esse aplicativo sem que qualquer recompilação seja necessária, fazendo uma pequena edição no arquivo `runtimeconfig` para configurar o `targetframeworkversion` como `"net8.0"`

Existem três maneiras de baixar o pacote `IBMMQDotnetClient`:

- Usando o Microsoft Visual Studio. O NuGet é distribuído como uma extensão do Microsoft Visual Studio. No Microsoft Visual Studio 2012, o NuGet é pré-instalado por padrão.
- Na linha de comandos, usando o NuGet Package Manager ou a CLI do .NET.
- Usando um navegador da web.

Quanto ao pacote redistribuível, você ativa o rastreio usando a variável de ambiente **MQDOTNET_TRACE_ON**.

Procedimento

- Para baixar o pacote `IBMMQDotnetClient` usando a IU do Gerente de Pacote em Microsoft Visual Studio, conclua as seguintes etapas:
  a) Clique com o botão direito no projeto do .NET e, em seguida, clique em **Gerenciar pacotes do Nuget**.
  b) Clique na guia **Procurar** e procure por `"IBMMQDotnetClient"`.
  c) Selecione o pacote e clique em **Instalar**.

  Durante a instalação, o Package Manager fornece informações de progresso na forma de instruções do console.
- Para baixar o pacote `IBMMQDotnetClient` a partir da linha de comando, escolha uma das seguintes opções:
  - usando o NuGet Package Manager, insira o comando a seguir:

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    Durante a instalação, o Package Manager fornece informações de progresso na forma de instruções do console. É possível redirecionar a saída para um arquivo de log.

- usando a CLI do .NET, insira o comando a seguir:

  ```
  dotnet add package IBMMQDotnetClient --version 9.1.4
  ```

- Usando um navegador da web, baixe o pacote `IBMMQDotnetClient` de https://www.nuget.org/packages/IBMMQDotnetClient.

Conceitos relacionados

Informações sobre licença do IBM MQ Client for .NET

Tarefas relacionadas

"Fazendo download do IBM MQ classes for XMS .NET do repositório do NuGet" na página 633
Os IBM MQ classes for XMS .NET estão disponíveis para download no repositório do NuGet para que possam ser facilmente consumidos por desenvolvedores do .NET.

Windows Instalando IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework, incluindo amostras, são instalados com IBM MQ. Há um pré-requisito do Microsoft.NET Framework no Windows.

A versão mais recente do IBM MQ classes for .NET Framework é instalada, por padrão, como parte da instalação padrão do IBM MQ no recurso *Java e .NET Messaging e Web Services*. Para obter instruções de instalação, consulte Instalando o servidor IBM MQ no Windows ou Instalando um cliente IBM MQ em sistemas Windows.

Em IBM MQ 9.3.0, para executar IBM MQ classes for .NET Framework deve-se instalar Microsoft.NET Framework V4.7.2 ou mais recente.

Os aplicativos existentes que são compilados com Microsoft.NET Framework V3.5 podem ser executados sem recompilação ao incluir a sinalização a seguir no arquivo `app.config` do aplicativo:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

Nota: Se o Microsoft .NET Framework V4.7.2 ou superior não for instalado antes da instalação do IBM MQ, a instalação do produto IBM MQ continuará sem erro, mas o IBM MQ classes for .NET não estará disponível Se o .NET Framework for instalado após instalar o IBM MQ, então os conjuntos IBM MQ.NET devem ser registrados executando o script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd`, em que *WMQInstallDir* é o diretório em que o IBM MQ está instalado. Este script instala as montagens necessárias no Global Assembly Cache (GAC). Um conjunto de arquivos do `amqi*.log` que registram as ações que são tomadas é criado no diretório %TEMP%. Não será necessário executar novamente o script `amqiRegisterdotNet.cmd` se .NET for atualizado para V4.7.2 ou superior a partir de uma versão anterior, por exemplo, de .NET V3.5.

Em um ambiente de instalação múltipla, se você tiver instalado anteriormente o IBM MQ classes for .NET como um pacote de suporte, não será possível instalar o IBM MQ, a menos que o pacote de suporte seja desinstalado primeiro. O recurso IBM MQ classes for .NET instalado com o IBM MQ contém as mesmas funções que o pacote de suporte.

Aplicativos de amostra, incluindo arquivos de origem, também são fornecidos; consulte "Aplicativos de amostra para o .NET" na página 573.

Para obter informações sobre o uso do canal customizado IBM MQ para o Microsoft WCF com .NET, consulte "Desenvolvendo aplicativos Microsoft Windows Communication Foundation com o IBM MQ" na página 1279

Conceitos relacionados

"Instalando IBM MQ classes for .NET" na página 566
IBM MQ classes for .NET, incluindo amostras, são instalados com IBM MQ em Windows e Linux

Tarefas relacionadas

Rastreando aplicativos .NET do IBM MQ

Opções para conectar o IBM MQ classes for .NET a um gerenciador de filas

Há três modos de conectar o IBM MQ classes for .NET a um gerenciador de filas. Considere qual tipo de conexão melhor se adequa aos seus requisitos.

Conexão de Ligações do Cliente

Para usar o IBM MQ classes for .NET como um IBM MQ MQI client, é possível instalá-lo, com o IBM MQ MQI client, na máquina do servidor do IBM MQ ou em uma máquina separada. Uma conexão de ligações do cliente pode usar transações XA ou não XA

Conexão de Ligações do Servidor

Quando usado no modo de ligações do servidor, o IBM MQ classes for .NET usa a API do gerenciador de filas, em vez de se comunicar através de uma rede. Isso proporciona melhor desempenho para aplicativos IBM MQ do que usar conexões de rede.

Para usar a conexão de ligações, deve-se instalar o IBM MQ classes for .NET no IBM MQ do servidor.

Conexão do Cliente Gerenciada

Uma conexão feita neste modo conecta-se como um cliente IBM MQ a um servidor IBM MQ em execução na máquina local ou remota.

O IBM MQ classes for .NET que se conecta nesse modo permanece no código gerenciado .NET e não faz chamadas para serviços nativos. Para obter informações adicionais sobre código gerenciado, consulte a documentação do Microsoft.

Existem várias limitações no uso do cliente gerenciado. Para obter mais informações sobre estes, consulte “Conexões do Cliente Gerenciadas” na página 589.

Aplicativos de amostra para o .NET

Para executar seus próprios aplicativos .NET, use as instruções para os programas de verificação substituindo o nome do aplicativo no local dos aplicativos de amostra.

Os seguintes aplicativos de amostra são fornecidos:

- Um aplicativo de entrada de mensagem
- Um aplicativo de obtenção de mensagem
- Um aplicativo 'hello world'
- Um aplicativo de publicação/assinatura
- Um aplicativo que usa propriedades de mensagem

Todos esses aplicativos de amostra são fornecidos na linguagem C, e alguns também são fornecidos em C++ e Visual Basic. É possível gravar aplicativos em qualquer idioma suportado por .NET.

Programa de "entrada de mensagem" SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)

Este programa mostra como colocar uma mensagem em uma fila nomeada. O programa possui três parâmetros:

- O nome de uma fila (necessário), por exemplo, SYSTEM.DEFAULT.LOCAL.QUEUE
- O nome de um gerenciador de filas (opcional)
- A definição de um canal (opcional), por exemplo, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Se nenhum nome do gerenciador de filas for fornecido, o gerenciador de filas será padronizado com o gerenciador de filas locais padrão. Se um canal for definido, ele tem o mesmo formato que a variável de ambiente MQSERVER.

Programa de "obtenção de mensagem" SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
Este programa mostra como obter uma mensagem de uma fila nomeada. O programa possui três parâmetros:

- O nome de uma fila (necessário), por exemplo, SYSTEM.DEFAULT.LOCAL.QUEUE
- O nome de um gerenciador de filas (opcional)
- A definição de um canal (opcional), por exemplo, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Se nenhum nome do gerenciador de filas for fornecido, o gerenciador de filas será padronizado com o gerenciador de filas locais padrão. Se um canal for definido, ele tem o mesmo formato que a variável de ambiente MQSERVER.

Programa "Hello World" (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
Este programa mostra como colocar e obter uma mensagem. O programa possui três parâmetros:

- O nome de uma fila (opcional), por exemplo, SYSTEM.DEFAULT.LOCAL.QUEUE ou SYSTEM.DEFAULT.MODEL.QUEUE
- O nome de um gerenciador de filas (opcional)
- Uma definição de canal (opcional), por exemplo, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Se nenhum nome de fila for fornecido, o nome padrão será SYSTEM.DEFAULT.LOCAL.QUEUE. Se nenhum nome do gerenciador de filas for fornecido, o gerenciador de filas será padronizado com o gerenciador de filas locais padrão.

Programa de "publicação/assinatura" (MQPubSubSample.cs)
Esse programa mostra como usar o IBM MQ de publicação/assinatura. Ele é fornecido somente em C#. O programa possui dois parâmetros:

- O nome de um gerenciador de filas (opcional)
- Uma definição de canal (opcional)

Programa de "propriedades de mensagem" (MQMessagePropertiesSample.cs)
Este programa mostra como usar propriedades de mensagem. Ele é fornecido somente em C#. O programa possui dois parâmetros:

- O nome de um gerenciador de filas (opcional)
- Uma definição de canal (opcional)

É possível verificar sua instalação compilando e executando estes aplicativos.

Locais de Instalação

Os aplicativos de amostra estão instalados nos seguintes locais, de acordo com a linguagem na qual são gravados. O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

C++ Gerenciado

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

Construindo os aplicativos de amostra

Para construir os aplicativos de amostra, um arquivo em lote é fornecido para cada linguagem.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

O arquivo bldcssamp.bat contém uma linha para cada amostra, que é tudo o que é necessário para construir este programa de amostra:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

C++ Gerenciado

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

O arquivo bldmcpsamp.bat contém uma linha para cada amostra, que é tudo o que é necessário para construir este programa de amostra:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Se você desejar compilar esses aplicativos no Microsoft Visual Studio 2003/.NET SDKv1.1, substitua o comando de compilação:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

por

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

O arquivo bldvbsamp.bat contém uma linha para cada amostra, que é tudo o que é necessário para construir este programa de amostra:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Amostras para usar o IBM MQ com o Microsoft .NET Core

IBM MQ suporta .NET Core para IBM MQ .NET aplicativos em ambientes Windows . O IBM MQ classes for .NET Standard, incluindo amostras, são instalados por padrão como parte da instalação padrão do IBM MQ.

Os aplicativos de amostra para IBM MQ .NET são instalados em `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`. Um script também é fornecido, que pode ser usado para compilar as amostras.

É possível construir as amostras usando os arquivos `build.bat` fornecidos. Há um `build.bat` para cada amostra no local a seguir em Windows:

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux O IBM MQ também suporta Core para aplicativos em ambientes Linux .

Para obter mais informações sobre como usar o IBM MQ com o Microsoft .NET Core, consulte “Instalando IBM MQ classes for .NET” na página 566.

Configurando seu Gerenciador de Filas para Aceitar Conexões do Cliente TCP/IP

Configure um gerenciador de filas para aceitar pedidos de conexão recebidos dos clientes.

Sobre esta tarefa

Essa tarefa explica as etapas básicas para configurar um gerenciador de filas para aceitar conexões do cliente TCP/IP. Para um sistema de produção, deve-se também considerar as implicações de segurança ao configurar gerenciadores de filas.

Procedimento

1. Defina um canal de conexão do servidor:
   a. Inicie o gerenciador de filas.
   b. Defina um canal de amostra chamado NET.CHANNEL:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   Importante: Essa amostra é destinada a uso em um ambiente de modo seguro apenas, uma vez que não inclui nenhuma consideração de implicações de segurança. Para um sistema de produção, considere usar TLS ou uma saída de segurança. Consulte Protegendo IBM MQ para obter mais informações.
2. Inicie um listener:

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   Nota: Os colchetes indicam parâmetros opcionais; *qmname* não é necessário para o gerenciador de filas padrão e o número da porta *portnum* não é necessário se você estiver usando o padrão (1414).

Transações distribuídas em .NET

As transações distribuídas ou transações globais permitem que os aplicativos clientes incluam várias origens diferentes de dados em dois ou mais sistemas em rede em uma transação.

Em transações distribuídas, um gerenciador de transações coordena e gerencia a transação entre dois ou mais gerenciadores de recursos.

Transações podem ser um processamento de single-phase ou two-phase commit. O single-phase commit é um processo no qual somente um gerenciador de recursos participa na transação e o processo two-phase commit é quando há mais de um gerenciador de recursos participando da transação. No processo two-phase commit, o gerenciador de transações envia uma chamada de preparação para verificar se todos os gerenciadores de recursos estão preparados para confirmar. Quando recebe o reconhecimento de todos os gerenciadores de recursos, a chamada de confirmação é emitida. Caso contrário, acontece um retrocesso de toda a transação. Consulte gerenciamento de transações e suporte para obter mais detalhes. Os gerenciadores de recursos devem informar aos gerenciadores de transação de sua participação na transação. Quando o gerenciador de recursos informa ao gerenciador de transações de sua participação, o gerenciador de recursos obtém retornos de chamada do gerenciador de transações quando a transação for confirmada ou retrocedida.

O IBM MQ .NET Classes já suporta transações distribuídas em conexões de modo não gerenciado e de ligações de servidores. Nesses modos, o IBM MQ .NET Classes delega todas as suas chamadas ao cliente de transações estendido C, que gerencia o processamento de transações em nome de .NET.

IBM MQ.NET Classes agora suporta transações distribuídas no modo gerenciado em que o IBM MQ .NET Classes usa o namespace System.Transactions para o suporte a transações distribuídas. A infraestrutura de System.Transactions torna a programação transacional simples e eficiente, suportando as transações iniciadas em todos os gerenciadores de recursos, incluindo o IBM MQ. O aplicativo IBM MQ .NET pode colocar e obter mensagens usando o modelo de programação de transação implícita ou de programação de transação explícita do .NET. Nas transações implícitas, os limites de transações são criados pelo programa de aplicativo que decide quando confirmar, retroceder (para transações explícitas) ou concluir a transação. Em transações explícitas, é necessário especificar explicitamente se deseja confirmar, retroceder e concluir a transação.

O IBM MQ.NET usa o Microsoft distributed transaction coordinator (MS DTC) como o gerenciador de transações, que coordena e gerencia a transação entre vários gerenciadores de recursos. O IBM MQ é usado como o gerenciador de recursos. Observe que não é possível usar TLS com transações XA. Deve-se usar CCDT. Para obter mais informações, veja Usando o cliente transacional estendido com canais TLS.

O IBM MQ.NET segue o modelo X/Open Distributed Transaction Processing (DTP). O modelo X/Open Distributed Transaction Processing é um modelo de processamento de transação distribuída proposto pelo Open Group, um consórcio de fornecedores. Esse modelo é um padrão entre a maioria dos fornecedores comerciais nos domínios de banco de dados e processamento de transações. A maioria dos produtos comerciais de gerenciamento de transações suporta o modelo X/DTP.

Modos de transação

- “Transações distribuídas no modo gerenciado do .NET” na página 578
- Transações distribuídas para o modo não gerenciado

Coordenando transações em vários cenários

- Uma conexão pode participar de várias transações, mas somente uma transação está ativa a qualquer momento.
- Durante uma transação, a chamada MQQueueManager.Disconnect é honrada. Nesse caso, é solicitado o o retrocesso da transação.
- Durante uma transação, a chamada MQQueue.Close ou MQTopic.Close é honrada. Nesse caso, é solicitado o retrocesso da transação.
- Os limites de transações são criados pelo programa de aplicativo que decide quando confirmar, retroceder (para transações explícitas) ou concluir (para transações implícitas) a transação.
- Se o aplicativo cliente quebrar durante uma transação com um erro inesperado antes de emitir uma chamada Put ou Get em uma chamada de queue ou topic, a transação será retrocedida e uma MQException será lançada.
- Se o código de razão MQCC_FAILED for retornado durante uma chamada Put ou Get em uma chamada queue ou topic, uma MQException será lançada com o código de razão e a transação será movimentada. Se uma chamada prepare já tiver sido emitida pelo gerenciador de transações, então, o IBM MQ .NET

retornará a solicitação de preparação retrocedendo a transação de modo forçado. Em seguida, o gerenciador da transações DTC causa um retrocesso no trabalho atual com todos os gerenciadores de recursos em transações de ambiente atuais.

- Durante uma transação que envolve diversos gerenciadores de recursos, se alguma razão ambiental fizer com que a chamada Put ou Get seja interrompida indefinidamente, o gerenciador de transações espera um tempo estipulado. Após decorrer esse tempo, causa o retrocesso de todo o trabalho atual com todos os gerenciadores de recursos em transações de ambiente atuais. Se essa espera indefinida ocorrer durante a fase de preparação, o gerenciador de transações pode atingir o tempo limite ou emitir uma chamada em dúvida no recurso, nesse caso, a transação é retrocedida.
- Aplicativos usando transações devem efetuar Put ou Get de mensagens sob SYNC_POINT. Se uma chamada Put ou Get de mensagem for emitida em um contexto transacional que não esteja sob SYNC_POINT, a chamada falhará com o código de razão MQRC_UNIT_OF_WORK_NOT_STARTED.

Diferenças comportamentais entre suporte de transação de cliente gerenciado e não gerenciado suport usando o namespace Microsoft.NET System.Transactions

As transações aninhadas têm um TransactionScope dentro de outro TransactionScope

- O cliente totalmente gerenciado IBM MQ .NET suporta TransactionScope aninhado
- O cliente não gerenciado IBM MQ .NET não suporta TransactionScope aninhado

Transações dependentes de System.Transactions

- O cliente totalmente gerenciado IBM MQ .NET suporta o recurso de transações dependentes fornecido por System.Transactions.
- O cliente não gerenciado IBM MQ .NET não suporta o recurso de transações dependentes fornecido por System.Transactions.

Amostras do produto

Amostras de produto `SimpleXAPut`, e `SimpleXAGet` estão disponíveis sob `WebSphere MQ\tools\dotnet\samples\cs\base`. As amostras são aplicativos C#, que demonstram como usar MQPUT e MQGET sob Transações distribuídas usando o namespace SystemTransactions. Para obter mais informações sobre essas amostras, consulte “Criando mensagens simples de put e get em um TransactionScope” na página 581.

Transações distribuídas no modo gerenciado do .NET

As classes do IBM MQ .NET usam o namespace System.Transactions para o suporte das transações distribuídas no modo gerenciado. No modo gerenciado, MS DTC coordena e gerencia as transações distribuídas em todos os servidores envolvidos em uma transação.

As classes do IBM MQ .NET fornecem um modelo de programação explícito com base na classe System.Transactions.Transaction e um modelo de programação implícita utilizando o System.Transactions.TransactionScope, a classe em que as transações são gerenciadas automaticamente pela infraestrutura.

Transação Implícita

A parte de código a seguir descreve como um aplicativo IBM MQ .NET coloca uma mensagem usando a programação de transação implícita do .NET.

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

Explicação do fluxo de código de transação implícita

O código cria *TransactionScope* e coloca a mensagem sob o escopo. Ele, então, chama *Concluído* para informar o coordenador de transação da conclusão da transação. O coordenador de transação agora emite *prepare* e *commit* para concluir a transação. Se um problema for detectado, um *retrocesso* é chamado.

Transação Explícita

O código a seguir descreve como um aplicativo IBM MQ .NET coloca mensagens usando o modelo de programação de transação explícita do .NET.

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

Explicação do fluxo de código de transação explícita

A parte do código cria transação usando a classe *CommitableTransaction*. Ele coloca uma mensagem sob esse escopo e, em seguida, chama explicitamente *commit* para concluir a transação. Se houver qualquer problema, o *retrocesso* é chamado.

Transações Distribuídas no Modo Não Gerenciado do .NET

As classes do IBM MQ.NET suportam as de conexões não gerenciadas (cliente) usando o cliente de transação estendido e COM+/MTS como o coordenador de transação, usando o modelo de programação de transação implícita ou explícita. No modo não gerenciado, as classes do IBM MQ .NET delegam todas suas chamadas ao cliente de transação estendido C que gerencia o processamento de transações em nome do .NET.

O processamento de transações é controlado por um gerenciador de transações externo, coordenando a unidade de trabalho global sob o controle da API do gerenciador de transações. Os verbos MQBEGIN, MQCMIT e MQBACK estão indisponíveis.IBM MQ .NET classes expõem esse suporte por meio de seu modo de transporte não gerenciado (cliente C). Consulte Configurando gerenciadores de transações compatíveis com XA

MTS é desenvolvido como um sistema de processamento de transações (TP) para fornecer os mesmos recursos no Windows NT como disponível no CICS, Tuxedo e em outras plataformas. Quando o MTS é instalado, um serviço separado é incluído ao Windows NT chamado de Microsoft Coordenador de Transação Distribuída (MSDTC). O MSDTC coordena as transações que abrangem armazenamentos de dados ou recursos separados. Para funcionar, ele requer que cada armazenamento de dados implemente seu próprio gerenciador de recursos proprietário.

O IBM MQ se torna compatível com o MSDTC implementando uma interface (interface do gerenciador de recursos do proprietário) na qual ele gerencia para mapear chamadas DTC XA para chamadas do IBM MQ(X/Open). IBM MQ executa a função de um gerenciador de recursos.

Quando um componente como COM+ solicitar o acesso a um IBM MQ, o COM geralmente verificará com o objeto de contexto MTS apropriado se uma transação for requerida. Se uma transação for necessária, o COM informará ao DTC e automaticamente iniciará uma transação do IBM MQ integral para esta operação. Em seguida, o COM funciona com os dados através do software MQMTS, colocando e obtendo mensagens conforme necessário. A instância do objeto obtida do COM chamará o método SetComplete ou SetAbort,

após todas as ações nos dados serem finalizadas. Quando o aplicativo emitir SetComplete, a chamada sinalizará o DTC que o aplicativo concluiu a transação e o DTC poderá prosseguir com o processo de two-phase commit. O DTC, em seguida, emite chamadas para MQMTS que, por sua vez, emite chamadas para IBM MQ para confirmar ou retroceder a transação.

Escrevendo um aplicativo IBM MQ .NET usando um cliente não gerenciado

Para executar no contexto do COM+, uma classe do .NET deve ser herdada a partir de System.EnterpriseServices.ServicedComponent. As regras e as recomendações para criarem conjuntos que usam componentes atendidos são as seguintes:

Nota: As etapas a seguir serão relevantes somente se você estiver usando o modo System.EnterpriseServices.

- A classe e o método que estão sendo iniciados em COM+ devem ambos serem públicos (sem classes internas e nenhum método estático ou protegido).
- Os atributos de método e classe: o atributo TransactionOption determina o nível de transação da classe, ou seja, se as transações são desativadas, suportadas ou necessárias. O atributo AutoComplete no método ExecuteUOW() instruirá o COM+ a confirmar a transação, se nenhuma exceção não manipulada for lançada.
- Nomeando fortemente uma montagem: o conjunto deve ser fortemente nomeado e registrado no Global Assembly Cache (GAC). A montagem será registrada em COM+ explicitamente ou por registro lento depois que ela for registrada no GAC.
- Registrando uma montagem em COM: prepare o conjunto a ser exposto aos clientes COM. Em seguida, crie uma biblioteca de tipos usando a ferramenta Assembly Registration, regasm.exe.

```
regasm UnmanagedToManagedXa.dll
```

- Registre o conjunto no GAC `gacutil /i UnmanagedToManagedXa.dll`.
- Registre o conjunto em COM+ usando a ferramenta do instalador de serviços do .NET, regsvcs.exe. Consulte a biblioteca de tipos criada por regasm.exe:

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- A montagem é implementada no GAC e posteriormente registrada em COM+ pelo registro lento. A estrutura do .NET cuidará do registro após o código ser executado pela primeira vez.

O fluxo de código de exemplo usando o modelo System.EnterpriseServices e System.Transactions com COM+ são descritos nas seções a seguir:

Fluxo de código de exemplo usando o modelo System.EnterpriseServices

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
```

```
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

Fluxo de código de exemplo usando System.Transactions para interações com COM+

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

Criando mensagens simples de put e get em um TransactionScope

Aplicativos C# de amostra do produto estão disponíveis no IBM MQ. Estes aplicativos simples demonstram a colocação e obtenção de mensagens em um TransactionScope. Ao final da tarefa, será possível colocar e obter mensagens de uma fila ou tópico.

Antes de começar

O serviço MSDTC deve estar em execução e ativado para Transações XA.

Sobre esta tarefa

O exemplo é um aplicativo simples, `SimpleXAPut` e `SimpleXAGet`. Os programas `SimpleXAPut` e `SimpleXAGet` são aplicativos C# estão disponíveis no IBM MQ. O `SimpleXAPut` demonstra o uso de MQPUT sob Transações distribuídas usando o namespace SystemTransactions. `SimpleXAGet` demonstra o uso de MQGET sob nomes transações distribuídas usando o namespace SystemTransactions.

`SimpleXAPut` está localizado em `MQ\tools\dotnet\samples\cs\base`

Procedimento

Os aplicativos podem ser executados com os parâmetros da linha de comandos a partir de `tools\dotnet\samples\cs\base\bin`

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

em que os parâmetros são:

-destinationURI

Isso pode ser fila ou tópico. Para uma fila, especifique como `queue://queueName` e para um tópico especifique como `topic://topicName`.

-host

Isso pode ser um nome do host, como host local, ou um endereço IP.

-port

A porta na qual o gerenciador de filas está em execução.

-channel

O canal de conexão que está sendo usado. O padrão é `SYSTEM.DEF.SVRCONN`

-transaction

O resultado da transação, por exemplo de confirmação ou retrocesso.

-mode

O modo de transporte, por exemplo, gerenciado ou não gerenciado.

-numberOfMsgs

O número de mensagens. O padrão é 1.

Exemplo

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

Recuperando transações no IBM MQ .NET

Esta seção descreve o processo de recuperação de transações no IBM MQ .NET XA usando o modo gerenciado.

Sobre esta tarefa

No processamento de transações distribuídas, as transações podem ser concluídas com sucesso, mas pode haver cenários em que uma transação pode falhar por muitas razões. Esses motivos podem incluir uma falha do sistema, falha de hardware, erro de rede, dados incorretos ou inválidos, erros de aplicação ou desastres naturais ou artificiais. Não é possível evitar falhas de transação. O sistema de transação distribuída deve ser capaz de manipular estas falhas. Ele deverá ser capaz de detectar e corrigir erros quando eles ocorrem. Esse processo é conhecido como Recuperação da transação.

Um aspecto importante do Distributed Transaction Processing é recuperar as transações incompletas ou indeterminadas. É essencial para executar a recuperação como parte da Unidade de trabalho de uma transação específica mantida bloqueada até que seja recuperada. Microsoft.NET a partir de sua biblioteca de classe System.Transactions fornece a opção para recuperar transações incompletas/indeterminadas. Este suporte a recuperação espera o Resource Manager para manter os logs de transações e executar a recuperação quando necessário.

No modelo de recuperação de transação do Microsoft .NET, o gerenciador de transações (System.Transactions ou coordenador do Microsoft Distributed Transaction (MS DTC) ou ambos), inicia, coordena e controla a recuperação da transação. O protocolo OLE Tx (o protocolo Microsoft XA) baseado nos gerenciadores de recursos fornece as opções para configurar o DTC para conduzir, coordenar e controlar a recuperação para eles. Para fazer isso, os gerenciadores de recurso devem registrar XA_Switch com o MS DTC usando a interface nativa.

XA_Switch fornece os pontos de entrada de funções XA como xa_start, xa_end e xa_recover no gerenciador de recursos para o Distributed Transaction Coordinator.

Recuperação usando o Microsoft Distributed Transaction Coordinator (DTC):

O Microsoft Distributed Transaction Coordinator fornece dois tipos de processos de recuperação.

Recuperação inativa

A recuperação inativa será executada se o processo do gerenciador da transação falhar enquanto uma conexão com um gerenciador de recursos XA estiver aberta. Quando o gerenciador de transações for reiniciado, ele lerá os logs do gerenciador de transações e restabelecerá a conexão com o gerenciador de recursos XA e, em seguida, iniciará a recuperação.

Recuperação ativa

A recuperação ativa será executada, se o gerenciador de transações permanecer ativo enquanto a conexão entre o gerenciador de transações e o gerenciador de recursos XA falhar porque o gerenciador de recursos XA ou a rede falhou. Após a falha, o gerenciador de transações tentará periodicamente se reconectar ao gerenciador de recursos XA. Quando a conexão for restabelecida, o gerenciador de transações iniciará a recuperação XA.

O namespace de System.Transactions fornece implementação gerenciada de transações distribuídas baseadas em MS DTC como o gerenciador de transações. Ele fornece recursos semelhantes à interface nativa do MS DTC, mas em um ambiente totalmente gerenciado. A única diferença é sobre a recuperação da transação. System.Transactions espera o gerenciadores de recursos para conduzir a recuperação por si só, então, coordena com o Transaction Managers (MS DTC). O gerenciador de recursos deve solicitar a recuperação de uma transação específica incompleta e, em seguida, o Transaction Manager a aceita e coordena baseado no resultado real dessa transação específica.

Processo de recuperação de transação para IBM MQ .NET

Esta seção descreve como as transações distribuídas podem ser recuperadas com as classes IBM MQ .NET.

Visão Geral

Para recuperar uma transação incompleta, as informações de recuperação são necessárias. As informações de recuperação da transação devem ser registradas para armazenamento pelos gerenciadores de recursos.IBM MQ Classes .NET seguem um caminho semelhante. As informações de recuperação da transação serão registradas em uma fila do sistema chamada SYSTEM.DOTNET.XARECOVERY.QUEUE.

A recuperação da transação em IBM MQ .NET é um processo de dois estágios:

1. Criação de login de informações de recuperação de transações no SYSTEM.DOTNET.XARECOVERY.QUEUE.
2. Recuperando transações usando o aplicativo XA Monitor WmqDotnetXAMonitor.

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM.DOTNET.XARECOVERY.FILA é uma fila de sistema que mantém informações de recuperação de transações para transações incompletas. Essa fila é criada quando um gerenciador de filas é criado.

Para cada transação, durante a fase de preparação, uma mensagem persistente contendo as informações de recuperação é incluída em SYSTEM.DOTNET.XARECOVERY.QUEUE. A mensagem é excluída se a chamada de confirmação for bem-sucedida.

Nota: Você não deve excluir a fila SYSTEM.DOTNET.XARECOVERY.QUEUE.

Aplicativo WMQDotnetXAMonitor

IBM MQ .NET O aplicativo XA Monitor, WmqDotnetXAMonitor, é um aplicativo gerenciado .NET que monitora um gerenciador de filas, processa mensagens em SYSTEM.DOTNET.XARECOVERY.queue e recupera transações incompletas

Se o agente do canal de mensagens (MCA) não conseguir colocar a mensagem na fila de destino, ele gera um relatório de exceção contendo a mensagem original e a coloca em uma fila de transmissão para ser enviada para a fila de resposta especificada na mensagem original. (Se a fila de resposta estiver no mesmo gerenciador de filas que o MCA, a mensagem será colocada diretamente nessa fila, e não em uma fila de transmissão)

As seguintes são consideradas transações incompletas e são recuperadas:

- Se a transação estiver preparada, mas COMMIT não foi concluído dentro do período de tempo limite.
- Se a transação estiver preparada, mas o gerenciador de filas do IBM MQ tiver sido desativado.
- Se a transação estiver preparada, mas, em seguida, o Transaction Manager tiver sido desativado.

  O aplicativo XA Monitor deve ser executado a partir do mesmo sistema em que o seu aplicativo de cliente IBM MQ .NET está sendo executado. Se houver aplicativos que estão em execução em vários sistemas e se conectar ao mesmo gerenciador de filas, o aplicativo WmqDotnetXAMonitor deve ser executado a partir de todos os sistemas. Apesar de cada máquina do cliente possuir uma instância do aplicativo XA Monitor em execução para recuperar o aplicativo, cada instância do XA Monitor deverá ser capaz de identificar a mensagem que corresponde a transação que o MS DTC local do atual XA Monitor estava coordenando para que ele possa voltar a se alistar e completá-lo.

Conceitos relacionados

“Processos de uso de recuperação de transação para IBM MQ .NET.” na página 584
Há vários casos de uso diferentes a partir dos quais as transações podem precisar ser recuperadas.

Tarefas relacionadas

“Usando o aplicativo WMQDotnetXAMonitor” na página 585
O cliente IBM MQ .NET fornece um aplicativo XA Monitor, WmqDotnetXAMonitor, que você pode usar para recuperar qualquer transação distribuída incompleta. O aplicativo WmqDotnetXAMonitor estabelece uma conexão com o gerenciador de filas onde as transações estão em dúvida e, em seguida, resolve a transação com base nos parâmetros que você configura.

Processos de uso de recuperação de transação para IBM MQ .NET.

Há vários casos de uso diferentes a partir dos quais as transações podem precisar ser recuperadas.

- **IBM MQ O aplicativo utilizando um único DTC e uma única instância de gerenciador de filas:** Nesse caso de uso, ao se conectar ao gerenciador de filas e executar a Unidade de Trabalho (UoW) sob transação, e se a transação falhar e se tornar incompleta, o aplicativo XA Monitor recupera a transação e a conclui.

  Neste caso de uso, haverá uma única instância do aplicativo XA Monitor em execução, uma vez que um único gerenciador de filas está associado às transações.
- **Vários aplicativos IBM MQ usando um DTC único e uma instância única de gerenciador de filas:** Nesse caso de uso, há mais de um aplicativo IBM MQ sob um DTC único e todos estão conectados ao mesmo gerenciador de filas e executando UoW sob transações.

Se as transações falharem e se tornarem incompletas, o aplicativo XA Monitor as recupera e conclui as transações referentes a todos os aplicativos.

Nesse caso de uso, uma única instância do aplicativo do XA Monitor é executada, já que um gerenciador de filas é usado em transações.

- **Aplicativos múltiplos IBM MQ, vários DTCs, instâncias diferentes do gerenciador de filas:** Nesse caso de uso, há mais de um aplicativo IBM MQ sob diferentes DTCs (ou seja, cada aplicativo está sendo executado em uma máquina diferente) e se conectando a diferentes gerenciadores de filas.

  Se a falha ocorrer e a transação tornar-se incompleta, o aplicativo de monitor verificará o TransactionManagerWhereabouts na mensagem para determinar o endereço DTC. Se o valor TransactionManagerWhereabouts corresponder ao endereço DTC no qual o monitor está em execução, ele concluirá a recuperação, continuará a procura até que a mensagem correspondente ao DTC seja localizada.

  Neste caso de uso, haverá apenas uma instância do aplicativo XA Monitor sendo executada por cliente (usuário ou computador) já que cada cliente tem seu próprio gerenciador de filas usado em transações.
- **Aplicativos múltiplos IBM MQ, vários DTCs, várias instâncias do gerenciador de filas:** Nesse caso de uso, há mais de um aplicativo IBM MQ sob diferentes DTCs (ou seja, cada aplicativo está sendo executado em uma máquina diferente) e todos estão se conectando com o mesmo gerenciador de filas.

  Se a falha ocorrer e a transação se tornar incompleta, o aplicativo de monitor verificará o TransactionManagerWhereabouts na mensagem para verificar se o endereço DTC e o valor correspondem com o DTC no qual o monitor está executando. Se ambos os valores corresponderem, ele concluirá a outra recuperação que continuará a procura até localizar a mensagem correspondente a seu DTC.

  Neste caso de uso, haverá apenas uma instância do aplicativo XA Monitor sendo executada por cliente (usuário ou computador) já que cada cliente tem sua própria associação de gerenciador de filas usada em transações.
- **Aplicativos múltiplos IBM MQ, DTC único, instâncias diferentes do gerenciador de filas:** Nesse caso de uso, há mais de um aplicativo IBM MQ sob um único DTC (ou seja, em um computador, há mais de um aplicativo IBM MQ em execução) e se conectando com diferentes gerenciadores de filas.

  Se a transação falhar e se tornar incompleta, o aplicativo de monitor recuperará a transação.

  Nesse caso de uso, haverá várias instâncias de aplicativo de monitor em execução como os gerenciadores de filas conectados, pois cada aplicativo possui seu próprio gerenciador de filas usado em transações e cada um deve ser recuperado.

  Nota: Se o aplicativo XA Monitor não estiver sendo executado em segundo plano, você poderá iniciá-lo.

Conceitos relacionados

"Processo de recuperação de transação para IBM MQ .NET" na página 583
Esta seção descreve como as transações distribuídas podem ser recuperadas com as classes IBM MQ .NET.

Tarefas relacionadas

"Usando o aplicativo WMQDotnetXAMonitor" na página 585
O cliente IBM MQ .NET fornece um aplicativo XA Monitor, WmqDotnetXAMonitor, que você pode usar para recuperar qualquer transação distribuída incompleta. O aplicativo WmqDotnetXAMonitor estabelece uma conexão com o gerenciador de filas onde as transações estão em dúvida e, em seguida, resolve a transação com base nos parâmetros que você configura.

Usando o aplicativo WMQDotnetXAMonitor

O cliente IBM MQ .NET fornece um aplicativo XA Monitor, WmqDotnetXAMonitor, que você pode usar para recuperar qualquer transação distribuída incompleta. O aplicativo WmqDotnetXAMonitor estabelece uma conexão com o gerenciador de filas onde as transações estão em dúvida e, em seguida, resolve a transação com base nos parâmetros que você configura.

Sobre esta tarefa

O aplicativo WMQDotnetXAMonitor deve ser executado manualmente. Ele pode ser iniciado a qualquer momento. Você pode iniciá-lo ao ver as mensagens no SYSTEM.DOTNET.XARECOVERY.FILA ou você pode mantê-lo ativo em segundo plano antes de fazer qualquer trabalho transacional com os aplicativos que são escritos usando classes IBM MQ .NET.

Você pode configurar os valores de parâmetro para WMQDotnetXAMonitor através da linha de comando ou ao usar um arquivo de configuração do aplicativo. Valores que são fornecidos por meio do arquivo de configuração do aplicativo têm precedência sobre valores configurados através da linha de comando.

Antes do IBM MQ 9.3.0, a conexão que o WMQDotnetXAMonitor estabelece é uma conexão não segura

A partir de IBM MQ 9.3.0, você tem a opção de estabelecer uma conexão segura com o gerenciador de filas, ao configurar parâmetros adicionais para o WMQDotnetXAMonitor.

Procedimento

- Para fornecer entrada para WmqDotNETXAMonitor usando um arquivo de configuração do aplicativo, consulte "Configurações do arquivo de configuração de aplicativo WmqDotNETXAMonitor" na página 587.
- Para iniciar o aplicativo WMQDotnetXAMonitor a partir da linha de comando, use o seguinte comando com os parâmetros que você requer:

  Antes de IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  De IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  Os parâmetros que você pode especificar são os seguintes:

  – **-m *QueueManagerName***
  O nome do gerenciador de filas.

  Opcional

  -n *ConnectionName*
  O nome de conexão em formato host (porta). *ConnectionName* pode conter mais de um nome de conexão. Diversos nomes de conexão devem ser fornecidos em uma lista separada por vírgula, por exemplo, `localhost (1414), localhost (1415), localhost (1416)`. O aplicativo WMQDotnetXAMonitor executa a recuperação para cada um dos nomes de conexão especificados na lista separada por vírgula.

  -c *ChannelName*
  O nome de canal.

  -i
  Conclusão da ramificação heurística.

  Opcional

  -k *Repositório de Chaves SSL*
  O nome do repositório de chaves SSL. Os valores suportados são:

  - *SISTEMA (este é o valor padrão)
  - *USER

  Opcional

-s *Especificação de código*
A CipherSpec que você configurar deve ser uma das CipherSpecs para a versão suportada e pode ser, preferencialmente, a mesma que a especificado na Política de Grupo Windows. Para obter mais informações, consulte “Suporte a CipherSpec para o cliente gerenciado do .NET” na página 608.

Obrigatório para estabelecer uma conexão segura com o gerenciador de filas.

-dn *Nome do SSLPeer*
O nome do par SSL usado para verificar o Nome Distinto (DN) do certificado a partir do gerenciador de filas de pares.

Opcional

-cl *Rótulo certificado*
O nome do rótulo que identifica o certificado.

Opcional

-sn *OutboundSNI*
Se a Indicação de Nome do Servidor (SNI) deve ser definida com o nome do canal de destino IBM MQ para o sistema remoto ao iniciar uma conexão TLS, ou para o nome do host. Os valores suportados para esta opção são:

- CANAL (este é o valor padrão)
- HOSTNAME
- *

Se nenhum valor for configurado então o valor padrão, ou seja CANAL, será usado.

Opcional

-cr *Verificação de Revocação de Certificados*
Se a verificação de revogação de certificado deve ser feita. Os valores suportados para esta opção são:

- true
- falso (este é o valor padrão)

Opcional

-kr *KeyResetCount*
O número total de bytes não criptografados que são enviados e recebidos no canal antes que a chave secreta usada para criptografia seja renegociada.

O valor padrão de 0 indica que as chaves secretas nunca são renegociadas

Opcional

O aplicativo WMQDotnetXAMonitor executa as seguintes ações:

1. Verifica a profundidade da fila de SYSTEM.DOTNET.XARECOVERY.QUEUE a um intervalo de 100 segundos.
2. Se a profundidade da fila for maior que zero, navegue na fila para mensagens e verificações se as mensagens satisfazem os critérios de transação incompletos.
3. Se uma mensagem satisfaz os critérios de transação incompletos, extrai e recupera as informações de recuperação da transação.
4. Determina se as informações de recuperação estão relacionadas ao Microsoft Distributed Transaction coordinator (MS DTC) local. Se este for o caso, então o WMQDotnetXAMonitor procede para recuperar a transação, caso contrário, ele volta a procurar na próxima mensagem.
5. Faz chamadas para que o gerenciador de filas recupere a transação incompleta.

Configurações do arquivo de configuração de aplicativo WmqDotNETXAMonitor
Você pode fornecer entrada para o aplicativo XA MonitorIBM MQ.NET WmqDotNETXAMonitor, usando um arquivo de configuração do aplicativo. Um arquivo de configuração de aplicativo de amostra é enviado com o IBM MQ .NET. Você pode modificar este arquivo de amostra de acordo com os seus requisitos.

Os valores de entrada fornecidos através do arquivo de configuração do aplicativo assumem a maior precedência. Se você fornecer valores de entrada tanto na linha de comando conforme descrito em "Usando o aplicativo WMQDotnetXAMonitor" na página 585 e no arquivo de configuração do aplicativo, então os valores do arquivo de configuração do aplicativo terão precedência.

Arquivo de configuração do aplicativo de amostra para antes de IBM MQ 9.3.0

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

Arquivo de configuração de aplicativo de amostra de IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

Log do aplicativo WmqDotNetXAMonitor

O aplicativo de monitor cria um arquivo de log no diretório do aplicativo para a criação de log do progresso do monitor e status de recuperação da transação. A criação de log é iniciada com o nome da conexão e os detalhes do canal para mostrar o gerenciador de filas atual para a qual a recuperação está em execução.

Após a recuperação ser iniciada, o MessageId da mensagem de recuperação da transação, o TransactionId da transação incompleta e o resultado real da transação também pela Coordenação do gerenciador de transação serão registrados.

Arquivo de log de amostra:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
```

```
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

Gravando e implementando programas do IBM MQ .NET

Para usar o IBM MQ classes for .NET para acessar filas do IBM MQ, grave programas em qualquer idioma suportado por .NET contendo chamadas que colocam e obtém mensagens na/das filas do IBM MQ.

Antes de começar

Deprecated V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, em IBM MQ classes for .NET, os métodos WriteObject(), ReadObject(), CreateObjectMessage () e as classes ObjectMessage e XmsObjectMessageImpl usados para serialização e desserialização de dados foram descontinuados.

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do IBM MQ .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

Sobre esta tarefa

A documentação do IBM MQ contém informações apenas nas linguagens C++, C# e Visual Basic.

Os tópicos nesta seção fornecem informações para ajudar a gravar aplicativos para interagir com sistemas IBM MQ . Para obter detalhes de classes individuais, consulte Classes e interfaces do IBM MQ .NET.

Diferenças de Conexão

A maneira como você programa o IBM MQ.NET tem algumas dependências dos modos de conexão que você deseja usar.

Quando IBM MQ classes for .NET são usados como um cliente gerenciado, há várias diferenças de um padrão do IBM MQ MQI client, pois alguns recursos não estão disponíveis para um cliente gerenciado.

O IBM MQ.NET determina qual tipo de conexão usar a partir das configurações que você especificar para nome de conexão, nome de canal, valor de customização NMQ_MQ_LIB e propriedade MQC.TRANSPORT_PROPERTY.

Conexões do Cliente Gerenciadas

Quando o IBM MQ classes for .NET é usado como um cliente gerenciado, há uma série de diferenças a partir de um padrão do IBM MQ MQI client.

Os recursos a seguir não estão disponíveis para um cliente gerenciado:

- Compactação de canal
- Encadeamento de saída do canal

Se você tentar usar estes recursos com um cliente gerenciado, isto retornará uma MQException. Se o erro for detectado na extremidade do cliente de uma conexão, ele usará o código de razão MQRC_ENVIRONMENT_ERROR. Se ele for detectado na extremidade do servidor, o código de razão retornado pelo servidor será usado.

Saídas de canal gravadas para um cliente não gerenciado não funcionam. É necessário gravar novas saídas especificamente para o cliente gerenciado. Verifique se não há saídas de canal inválidas especificadas em sua tabela de client channel definition table (CCDT).

O nome de uma saída do canal gerenciada pode ter até 999 caracteres de comprimento. Entretanto, se você usar a CCDT para especificar o nome de saída do canal, ele será limitado a 128 caracteres.

Comunicação é suportada somente sobre TCP/IP.

Quando você para um gerenciador de filas usando o comando **endmqm**, um canal de conexão do servidor para um cliente gerenciado .NET pode demorar mais para fechar do que os canais de conexão do servidor para outros clientes.

Se você tiver configurado *NMQ_MQ_LIB* para `gerenciado` para usar diagnósticos de problemas gerenciados do IBM MQ, nenhum dos parâmetros -i, -p, -s, -b ou -c do comando **strmqtrc** será suportado.

Um aplicativo do .NET gerenciado usando transações XA não funcionará com um gerenciador de filas do z/OS. Um cliente gerenciado .NET tentando se conectar a um gerenciador de filas do z/OS falhará com um erro, MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354), na chamada MQOPEN. No entanto, um aplicativo gerenciado .NET usando transações XA funcionará com o gerenciador de filas distribuídos.

Definindo qual Tipo de Conexão Usar

O tipo de conexão é determinado pela configuração do nome de conexão, nome de canal, do valor de customização NMQ_MQ_LIB e da propriedade MQC.TRANSPORT_PROPERTY.

É possível especificar o nome de conexão conforme a seguir:

- Explicitamente em um construtor MQQueueManager:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Configurando as propriedades MQC.HOST_NAME_PROPERTY e, opcionalmente, MQC.PORT_PROPERTY em uma entrada hashtable em um construtor MQQueueManager:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Como valores de MQEnvironment explícitos

```
MQEnvironment.Hostname
```

  `MQEnvironment.Port`(opcional).
- Configurando as propriedades MQC.HOST_NAME_PROPERTY e, opcionalmente, MQC.PORT_PROPERTY na hashtable MQEnvironment.properties.

É possível especificar o nome do canal conforme a seguir:

- Explicitamente em um construtor MQQueueManager:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Configurando a propriedade MQC.CHANNEL_PROPERTY em uma entrada de hashtable em um construtor MQQueueManager:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Como um valor de MQEnvironment explícito

```
MQEnvironment.Channel
```

- Configurando a propriedade MQC.CHANNEL_PROPERTY na hashtable MQEnvironment.properties.

É possível especificar a propriedade de transporte conforme a seguir:

- Configurando a propriedade MQC.TRANSPORT_PROPERTY em uma entrada de hashtable em um construtor MQQueueManager:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Configurando a propriedade MQC.TRANSPORT_PROPERTY na hashtable MQEnvironment.properties.

Selecione o tipo de conexão que você requer usando um dos seguintes valores:

MQC.TRANSPORT_MQSERIES_BINDINGS - conectar como servidor
MQC.TRANSPORT_MQSERIES_CLIENT - conectar como cliente não XA
MQC.TRANSPORT_MQSERIES_XACLIENT - conectar como cliente XA
MQC.TRANSPORT_MQSERIES_MANAGED - conectar como cliente gerenciado não XA

É possível configurar o valor de customização NMQ_MQ_LIB para escolher explicitamente o tipo de conexão conforme mostrado na tabela a seguir.

Valor de NMQ_MQ_LIB	**Tipo de conexão**
mqic.dll	Conectar como um cliente não XA
mqicxa.dll	Conectar como um cliente XA
mqm.dll	Conectar como um servidor ou como um cliente não XA
managed	Conectar como um cliente gerenciado não XA

Nota: Valores de mqic32.dll e mqic32xa.dll são aceitos como sinônimos de mqic.dll e mqicxa.dll para compatibilidade com liberações anteriores. Entretanto, mqm.dll e mqm.pdb são apenas parte do pacote do cliente de IBM WebSphere MQ 7.1 em diante.

Se você escolher um tipo de conexão que esteja indisponível em seu ambiente, por exemplo, especificar mqic32xa.dll e não tiver suporte a XA, o IBM MQ.NET lançará uma exceção.

Configurar NMQ_MQ_LIB como "gerenciado" faz com que o cliente use os testes de diagnóstico de problemas gerenciados do IBM MQ, a conversão de dados do .NET e outras funções de nível inferior gerenciadas do IBM MQ.

Todos os outros valores para NMQ_MQ_LIB fazem com que o processo .NET use testes de diagnóstico de problemas IBM MQ não gerenciados e conversão de dados, e outras funções de nível inferior não gerenciadas do IBM MQ (supondo um IBM MQ MQI client ou servidor esteja instalado no sistema).

O IBM MQ.NET escolhe o tipo de conexão conforme a seguir:

1. Se MQC.TRANSPORT_PROPERTY for especificado, ele conecta-se de acordo com o valor de MQC.TRANSPORT_PROPERTY.

   Observe, entretanto, que configurar MQC.TRANSPORT_PROPERTY com MQC.TRANSPORT_MQSERIES_MANAGED não garante que o processo do cliente seja executado gerenciado. Mesmo com esta configuração, o cliente não é gerenciado nos seguintes casos:

   - Se outro encadeamento no processo tiver se conectado com MQC.TRANSPORT_PROPERTY configurada como algo diferente de MQC.TRANSPORT_MQSERIES_MANAGED.
   - Se NMQ_MQ_LIB não estiver configurado como "gerenciado", testes de diagnóstico de problemas, conversão de dados e outras funções de nível inferior não serão totalmente gerenciadas (presumindo que um IBM MQ MQI client ou servidor esteja instalado no sistema).
2. Se um nome de conexão tiver sido especificado sem um nome de canal, ou um nome de canal tiver sido especificado sem um nome de conexão, ele lançará um erro.
3. Se um nome de conexão e um nome de canal tiverem sido especificados:
   - Se NMQ_MQ_LIB for configurado como mqic32xa.dll, ele se conectará como um cliente XA.

- Se NMQ_MQ_LIB for configurado como gerenciado, ele se conectará como um cliente gerenciado.
- Caso contrário, ele se conectará como um cliente não XA.

4. Se NMQ_MQ_LIB for especificado, ele se conectará de acordo com o valor de NMQ_MQ_LIB.
5. Se um servidor IBM MQ estiver instalado, ele se conectará como um servidor.
6. Se um IBM MQ MQI client estiver instalado, ele se conectará como um cliente não XA.
7. Caso contrário, ele se conectará como um cliente gerenciado.

Windows Usando o modelo de projeto IBM MQ .NET

O cliente IBM MQ .NET oferece a você a capacidade de usar um modelo de projeto para auxiliá-lo no desenvolvimento de seus aplicativos .NET Core.

Antes de começar

Deve-se ter o Microsoft Visual Studio 2017, ou mais recente, e o .NET Core 2.1 instalado no seu sistema.

Deve-se copiar o modelo .NET do diretório

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip

para o diretório

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

em que:

- *&MQ_INSTALL_ROOT* é o diretório-raiz da sua instalação
- *& USER_HOME_DIRECTORY* é o seu diretório inicial.

Deve-se parar e reiniciar Microsoft Visual Studio para escolher o modelo.

Sobre esta tarefa

O modelo de projeto .NET inclui algum código comum que pode ser usado para ajudar a desenvolver os seus aplicativos. Com o código integrado, é possível conectar-se ao gerenciador de filas IBM MQ e executar uma operação put ou get, simplesmente modificando as propriedades no código integrado.

Procedimento

1. Abra o Microsoft Visual Studio.
2. Clique em **Arquivo**, seguido por **Novo** e, em seguida, **Projeto**.
3. Na *janela Criar um novo projeto*, selecione `IBM MQ .NET Client App (.NET Core)` e clique em **Avançar**..
4. Na janela *Configurar seu novo projeto*, mude o *Nome do projeto* de seu projeto, se desejar, e clique em **Criar** para criar o projeto do .NET.

   `MQDotnetApp.cs` é o arquivo que é criado junto com o arquivo do projeto. Esse arquivo contém o código que se conecta ao gerenciador de filas e executa as operações put e get.

   As propriedades de conexão são configuradas para valores padrão:

   - MQC.CONNECTION_NAME_PROPERTY é configurada como *localhost(1414)*
   - MQC.CHANNEL_PROPERTY é configurada como *DOTNET.SVRCONN*

   A fila é configurada como *Q1* e é possível modificar essas propriedades adequadamente.
5. Compile e execute o aplicativo.

Conceitos relacionados

Componentes e recursos do IBM MQ

Aplicativo de tempo de execução do.NET - Windows somente

Arquivos de configuração para IBM MQ classes for .NET

Um aplicativo cliente do .NET pode usar um arquivo de configuração do IBM MQ MQI client e, se você estiver usando o tipo de conexão gerenciada, um arquivo de configuração do aplicativo do .NET. As configurações no arquivo de configuração de aplicativo têm prioridade.

Arquivo de Configuração do Cliente

Um aplicativo cliente IBM MQ classes for .NET pode usar um arquivo de configuração do cliente da mesma maneira que qualquer outro IBM MQ MQI client. Normalmente esse arquivo é chamado `mqclient.ini`, mas é possível especificar um nome de arquivo diferente. Para obter mais informações sobre o arquivo de configuração do cliente, consulte IBM MQ MQI client arquivo de configuração `mqclient.ini`.

Somente os atributos a seguir em um arquivo de configuração do IBM MQ MQI client são relevantes para IBM MQ classes for .NET. Se você especificar outros atributos, isto não terá efeito.

Tabela 77. Atributos do arquivo de configuração do cliente que são relevantes para o IBM MQ classes for .NET

Sub-rotina	**Atributo**
Canais	CCSID
Canais	ChannelDefinitionDirectory
Canais	ChannelDefinitionFile
Canais	ReconDelay
Canais	DefRecon
Canais	MQReconnectTimeout
Canais	ServerConnectionParms
Canais	Put1DefaultAlwaysSync
Canais	PasswordProtection
ClientExitPath	Caminho Padrão das Saídas
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
Segurança	Arquivo MQIInitialKey
SSL	SSLKeyRepository
SSL	SSLKeyRepositorySenha
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

É possível substituir qualquer um destes atributos usando a variável de ambiente apropriada.

Arquivo de Configuração de Aplicativo

Se você estiver executando com o tipo de conexão gerenciada, também é possível substituir o arquivo de configuração do cliente IBM MQ e as variáveis de ambiente equivalentes usando o arquivo de configuração de aplicativo .NET.

As configurações de arquivo de configuração de aplicativo .NET são usadas apenas quando em execução com o tipo de conexão gerenciada e são ignoradas para outros tipos de conexão.

O arquivo de configuração do aplicativo .NET e seu formato são definidos pelo Microsoft para uso geral dentro da estrutura do .NET, mas os nomes de seção, chaves e valores específicos mencionados nesta documentação são específicos para IBM MQ.

O formato do arquivo de configuração do aplicativo .NET é um número de *seções*. Cada seção contém uma ou mais *chaves* e cada chave tem um *valor* associado. O exemplo a seguir mostra as seções, as chaves e os valores usados em um aplicativo .NET no arquivo de configuração para controlar a propriedade TCP/IP KeepAlive:

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

As palavras-chave usadas nos nomes e chaves da seção do arquivo de configuração do aplicativo .NET correspondem exatamente às palavras-chave para as sub-rotinas e atributos definidos no arquivo de configuração do cliente.

A seção <configSections> deve ser o primeiro elemento-filho do elemento <configuration>.

Consulte a documentação do Microsoft para obter informações adicionais.

Fragmento de código C# de exemplo para uso com o .NET

Um fragmento de código C# que demonstra que um aplicativo se conecta a um gerenciador de filas coloca uma mensagem em uma fila e recebe uma resposta.

O fragmento de código C# a seguir demonstra um aplicativo que executa três ações:

1. Conecta-se a um gerenciador de filas
2. Coloca uma mensagem em SYSTEM.DEFAULT.LOCAL.QUEUE
3. Obtém a mensagem de volta

Ele também mostra como alterar o tipo de conexão.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;
```

```
// Define the name of the queue manager to use (applies to all connections)
const String qManager = "your_Q_manager";

// Define the name of your host connection (applies to client connections only)
const String hostName = "your_hostname";

// Define the name of the channel to use (applies to client connections only)
const String channel = "your_channelname";



/// <summary>
/// Initialise the connection properties for the connection type requested
/// </summary>
/// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
static Hashtable init(String connectionType)
{
  Hashtable connectionProperties = new Hashtable();

  // Add the connection type
  connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

  // Set up the rest of the connection properties, based on the
  // connection type requested
  switch(connectionType)
  {
    case MQC.TRANSPORT_MQSERIES_BINDINGS:
      break;
    case MQC.TRANSPORT_MQSERIES_CLIENT:
    case MQC.TRANSPORT_MQSERIES_XACLIENT:
    case MQC.TRANSPORT_MQSERIES_MANAGED:
      connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
      connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
      break;
  }

  return connectionProperties;
}
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static int Main(string[] args)
{
  try
  {
    Hashtable connectionProperties = init(connectionType);

    // Create a connection to the queue manager using the connection
    // properties just defined
    MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

    // Set up the options on the queue we want to open
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

    // Now specify the queue that we want to open,and the open options
    MQQueue system_default_local_queue =
      qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

    // Define an IBM MQ message, writing some text in UTF format
    MQMessage hello_world = new MQMessage();
    hello_world.WriteUTF("Hello World!");

    // Specify the message options
    MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                         // same as MQPMO_DEFAULT

    // Put the message on the queue
    system_default_local_queue.Put(hello_world, pmo);



    // Get the message back again

    // First define an IBM MQ message buffer to receive the message
    MQMessage retrievedMessage =new MQMessage();
    retrievedMessage.MessageId =hello_world.MessageId;

    // Set the get message options
    MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                        //same as MQGMO_DEFAULT
```

```
      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

Configurando o ambiente IBM MQ

Antes de usar a conexão do cliente para se conectar a um gerenciador de filas, deve-se configurar o ambiente do IBM MQ.

Nota: Essa etapa não é necessária ao usar o IBM MQ classes for .NET no modo de ligações do servidor.

A interface de programação do .NET permite usar o valor de customização NMQ_MQ_LIB, mas também inclui uma classe MQEnvironment. Esta classe permite que você especifique detalhes que devem ser usados durante a tentativa de conexão, tal como os itens na lista a seguir:

- Nome do canal
- Nome do host
- Número da Porta
- Saídas do canal
- Parâmetros de SSL
- ID do Usuário e Senha

Para obter informações completas sobre a classe MQEnvironment, consulte MQEnvironment.NET class

Para especificar o nome do canal e o nome do host, use o código a seguir:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

Por padrão, os clientes tentam se conectar a um listener do IBM MQ na porta 1414. Para especificar uma porta diferente, use o código:

```
MQEnvironment.Port = nnnn;
```

Conectando-se e desconectando-se de um gerenciador de filas

Quando você tiver configurado o ambiente do IBM MQ, estará pronto para se conectar a um gerenciador de filas.

Para conectar-se a um gerenciador de filas, crie uma nova instância da classe MQQueueManager:

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

Para desconectar-se de um gerenciador de filas, chame o método `Disconnect` no gerenciador de filas:

```
queueManager.Disconnect();
```

Deve-se ter autoridade de consulta (`inq`) no gerenciador de filas ao tentar se conectar ao gerenciador de filas. Sem autoridade de consulta, a tentativa de conexão falhará.

Se você chamar o método `Disconnect`, todas as filas e processos abertos que você acessou usando esse gerenciador de filas serão encerrados. Entretanto, é uma boa prática de programação fechar estes recursos explicitamente quando você terminar de usá-los. Para fechar os recursos, use o método `Close` no objeto associado a cada recurso.

Os métodos `Commit` e `Backout` em um gerenciador de filas substituem as chamadas MQCMIT e MQBACK que são usadas com a interface processual.

Acessando Filas e Tópicos

É possível acessar filas e tópicos usando métodos de MQQueueManager ou construtores apropriados.

Para acessar filas, use os métodos da classe MQQueueManager. O MQOD (estrutura do descritor de objeto) é reduzido nos parâmetros destes métodos. Por exemplo, para abrir uma fila em um gerenciador de filas representado por um objeto MQQueueManager chamado queueManager, use o seguinte código:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

O parâmetro *options* é o mesmo que o parâmetro Options na chamada MQOPEN.

O método AccessQueue retorna um novo objeto da classe MQQueue.

Quando você tiver concluído o uso da fila, use o método Close() para fechá-la, como no exemplo a seguir:

```
queue.Close();
```

Com o IBM MQ .NET, também é possível criar uma fila usando o construtor MQQueue. Os parâmetros são exatamente os mesmos que para o método accessQueue, com a adição de um parâmetro do gerenciador de filas especificando o objeto MQQueueManager instanciado para uso. Por exemplo:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

Construir um objeto de fila desta maneira permite que você grave suas próprias subclasses de MQQueue.

De modo semelhante, também é possível acessar tópicos usando os métodos da classe MQQueueManager. Use um método AccessTopic() para abrir um tópico. Isto retorna um novo objeto da classe MQTopic. Quando você tiver concluído o uso do tópico, use o método Close() de MQTopic para fechá-lo.

Também é possível criar um tópico usando um construtor MQTopic. Há diversos construtores para tópicos; para obter mais informações, consulte Classe MQTopic.NET.

Manipulando Mensagens

As mensagens são tratadas usando os métodos das classes de fila ou tópico. Para criar uma nova mensagem, crie um novo MQMessageobject.

Coloque mensagens nas filas ou tópicos usando o método Put() da classe MQQueue ou MQTopic. Obtenha mensagens das filas ou tópicos usando o método Get() da classe MQQueue ou MQTopic. Diferente da interface processual, em que MQPUT e MQGET colocam e obtêm matrizes de bytes, o IBM MQ classes for .NET coloca e obtém instâncias da classe MQMessage. A classe MQMessage encapsula o buffer de dados que contém os dados da mensagem reais, junto com todos os parâmetros MQMD (descritor de mensagens) que descrevem essa mensagem.

Para criar uma nova mensagem, crie uma nova instância da classe MQMessage e use os métodos WriteXXX para colocar dados no buffer de mensagem.

Quando a nova instância de mensagem for criada, todos os parâmetros MQMD serão automaticamente configurados com seus valores padrão, conforme definido em valores iniciais do e declarações de idioma para MQMD. O método Put() de MQQueue também utiliza uma instância da classe MQPutMessageOptions como um parâmetro. Esta classe representa a estrutura MQPMO. O exemplo a seguir cria uma mensagem e a coloca em uma fila:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage,pmo);
```

O método Get() de MQQueue retorna uma nova instância de MQMessage, que representa a mensagem recém-obtida da fila. Ele também utiliza uma instância da classe MQGetMessageOptions como um parâmetro. Esta classe representa a estrutura de MQGMO.

Não é necessário especificar um tamanho de mensagem máximo, porque o método Get() ajusta automaticamente o tamanho de seu buffer interno para ajustar a mensagem recebida. Use os métodos ReadXXX da classe MQMessage para acessar os dados na mensagem retornada.

O exemplo a seguir mostra como obter uma mensagem de uma fila:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

É possível alterar o formato numérico que os métodos de leitura e gravação usam configurando a variável de membro *encoding*.

É possível alterar o conjunto de caracteres para usar para ler e gravar sequências configurando a variável de membro *characterSet*.

Consulte a classe MQMessage.NET para obter mais detalhes.

Nota: O método WriteUTF() de MQMessage codifica automaticamente o comprimento da sequência bem como os bytes Unicode que ela contém. Quando sua mensagem for lida por outro programa .NET (usando readUTF()), essa é a maneira mais simples de enviar informações de sequência.

Manipulando Propriedades de Mensagem

As propriedades de mensagem permitem que você selecione mensagens ou recupere informações sobre uma mensagem sem acessar seus cabeçalhos. A classe MQMessage contém métodos para obter e configurar propriedades.

É possível usar propriedades de mensagem para permitir que um aplicativo selecione mensagens para processar ou recupere informações sobre uma mensagem sem acessar cabeçalhos MQMD ou MQRFH2. Elas também facilitam a comunicação entre aplicativos IBM MQ e JMS. Para obter mais informações sobre propriedades de mensagens no IBM MQ, veja Propriedades de Mensagem.

A classe MQMessage fornece vários métodos para obter e configurar propriedades, de acordo com o tipo de dado da propriedade. Os métodos get possuem nomes no formato Get*Property e os métodos set possuem nomes no formato Set*Property, em que o asterisco (*) representa uma das sequências a seguir:

- Booleana
- Byte
- Bytes
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Long
- Object
- Curta
- Sequência

Por exemplo, para obter a propriedade myproperty do IBM MQ (uma sequência de caracteres), use a chamada `message.GetStringProperty('myproperty')`. É possível, opcionalmente, transmitir um descritor de propriedade, que o IBM MQ irá concluir.

Manipulando Erros

Erros de manipulador que surgem do IBM MQ classes for .NET usando os blocos `try` e `catch`.

Os métodos na interface do .NET não retornam um código de conclusão e código de razão. Em vez disso, eles lançam uma exceção sempre que o código de conclusão e código de razão resultam de uma chamada do IBM MQ e não são ambos zero. Isso simplifica a lógica do programa de forma que não seja necessário verificar os códigos de retorno após cada chamada para o IBM MQ. É possível decidir em quais pontos em seu programa você deseja lidar com a possibilidade de falha. Nestes pontos, é possível cercar seu código com blocos `try` e `catch`, como no exemplo a seguir:

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                        "CC = " + ex.CompletionCode +
                        "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

Obtendo e Configurando Valores de Atributos

As classes MQManagedObject, MQQueue, and MQQueueManager contêm métodos que permitem que você obtenha e configure seus valores de atributos. Observe que, para MQQueue, os métodos funcionam somente se você especificar a consulta apropriada e configurar sinalizadores quando abrir a fila.

Para obter atributos comuns, as classes MQQueueManager e MQQueue herdam de uma classe chamada MQManagedObject. Esta classe define as interfaces Inquire() e Set().

Quando você cria um novo objeto do gerenciador de filas usando o operador *new*, ele é aberto automaticamente para consulta. Quando você usa o método AccessQueue() para acessar um objeto de fila, esse objeto não é aberto automaticamente para operações de consulta ou de configuração; isso pode causar problemas com alguns tipos de filas remotas. Para usar os métodos Inquire e Set e para configurar propriedades em uma fila, é necessário especificar os sinalizadores de consulta e configuração apropriados no parâmetro openOptions do método AccessQueue().

Os métodos inquire e set utilizam três parâmetros:

- matriz de seletores
- matriz intAttrs
- matriz charAttrs

Não são necessários os parâmetros SelectorCount, IntAttrCount e CharAttrLength, que estão localizados em MQINQ, porque o comprimento de uma matriz é sempre conhecido. O exemplo a seguir mostra como fazer uma consulta em uma fila:

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

Todos os atributos destes objetos podem ser consultados. Um subconjunto de atributos é exposto como as propriedades de um objeto. Para obter uma lista dos atributos do objeto, veja Atributos de objetos. Para propriedades de objetos, consulte a descrição da classe apropriada.

Programas Multiencadeados

O ambiente de tempo de execução do .NET é inerentemente multiencadeado. O IBM MQ classes for .NET permite que um objeto do gerenciador de filas seja compartilhado em múltiplos encadeamentos, mas assegura que todo acesso ao gerenciador de filas de destino seja sincronizado.

Considere um programa simples que se conecta a um gerenciador de filas e abre uma fila na inicialização. O programa exibe um único botão na tela. Quando um usuário clica nesse botão, o programa busca uma mensagem da fila. Nesta situação, a inicialização do aplicativo ocorre em um encadeamento e o código que é executado em resposta ao pressionamento do botão é executado em um encadeamento separado (o encadeamento da interface com o usuário).

A implementação do IBM MQ .NET assegura que, para uma conexão específica (instância do objeto MQQueueManager), todo acesso ao gerenciador de filas de destino do IBM MQ seja sincronizado. O comportamento padrão é que um encadeamento que deseje emitir uma chamada a um gerenciador de filas seja bloqueado até todas as outras chamadas em andamento para essa conexão sejam concluídas. Se você requerer acesso simultâneo ao mesmo gerenciador de filas a partir de múltiplos encadeamentos em seu programa, crie um novo objeto MQQueueManager para cada encadeamento que requer acesso simultâneo. (Isto é equivalente a emitir uma chamada MQCONN separada para cada encadeamento.)

Se as opções de conexão padrão forem substituídas por MQC.MQCNO_HANDLE_SHARE_NONE ou MQC.MQCNO_SHARE_NO_BLOCK, o gerenciador de filas não será mais sincronizado.

Usando uma tabela de definição de canal do cliente com o .NET

É possível usar uma tabela de definição de canal do cliente (CCDT) com IBM MQ classes for .NET. Você especifica o local da CCDT de diferentes maneiras, dependendo de você estar usando uma conexão gerenciada ou uma conexão não gerenciada.

Tipo de conexão do cliente não gerenciada XA ou não XA

Com um tipo de conexão não gerenciado, é possível especificar o local da CCDT de duas maneiras:

- Usando as variáveis de ambiente MQCHLLIB para especificar o diretório no qual a tabela está localizada e MQCHLTAB para especificar o nome do arquivo da tabela.
- Usando o arquivo de configuração do cliente. Na sub-rotina CHANNELS, use os atributos **ChannelDefinitionDirectory** para especificar o diretório no qual a tabela está localizada e **ChannelDefinitionFile** para especificar o nome do arquivo..

Se o local for especificado das duas maneiras, no arquivo de configuração do cliente e usando variáveis de ambiente, as variáveis de ambiente terão prioridade. É possível usar este recurso para especificar um local padrão no arquivo de configuração do cliente e substituí-lo usando variáveis de ambiente quando necessário.

Tipo de conexão do cliente gerenciado

Com um tipo de conexão gerenciado, é possível especificar o local da CCDT de três maneiras:

- Usando o arquivo de configuração de aplicativo .NET. Na seção CHANNELS, use as chaves **ChannelDefinitionDirectory** para especificar o diretório onde a tabela está localizada e **ChannelDefinitionFile** para especificar o nome do arquivo.
- Usando as variáveis de ambiente MQCHLLIB para especificar o diretório no qual a tabela está localizada e MQCHLTAB para especificar o nome do arquivo da tabela.
- Usando o arquivo de configuração do cliente. Na sub-rotina CHANNELS, use os atributos **ChannelDefinitionDirectory** para especificar o diretório no qual a tabela está localizada e **ChannelDefinitionFile** para especificar o nome do arquivo..

Se o local for especificado de mais de uma dessas maneiras, as variáveis de ambiente terão prioridade sobre o arquivo de configuração do cliente, e o .NET Arquivo de Configuração de Aplicativo terá prioridade sobre os outros dois métodos. É possível usar este recurso para especificar um local padrão no arquivo de configuração do cliente e substituí-lo usando variáveis de ambiente ou o arquivo de configuração de aplicativo quando necessário.

No IBM MQ 9.3.0, o cliente .NET se comporta da mesma maneira que os clientes C e Java e retorna o MQRC_Q_MGR_NAME_ERROR ao usar uma CCDT com agrupamento do gerenciador de filas.

Como um aplicativo .NET determina qual definição de canal usar

No ambiente do cliente do IBM MQ .NET, a definição de canal a ser usada pode ser especificada de várias maneiras diferentes. Podem existir várias especificações da definição de canal. Um aplicativo deriva a definição de canal de uma ou mais origens.

Se existir mais de uma definição de canal, aquela usada será selecionada na seguinte ordem de prioridade:

1. Propriedades especificadas no construtor MQQueueManager, seja explicitamente ou incluindo *MQC.CHANNEL_PROPERTY* na hashtable de propriedades
2. Uma propriedade *MQC.CHANNEL_PROPERTY* na hashtable MQEnvironment.properties
3. A propriedade *Channel* em MQEnvironment
4. O arquivo de configuração de aplicativo .NET, nome de seção CHANNELS, chave ServerConnectionParms (aplica-se apenas a conexões gerenciadas)
5. A variável de ambiente *MQSERVER*

6. O arquivo de configuração do cliente, sub-rotina CHANNELS, Atributo ServerConnectionParms
7. A Client Channel Definition Table (CCDT). O local do CCDT é especificado no arquivo de configuração do aplicativo .NET (aplica-se apenas a conexões gerenciadas)
8. A Client Channel Definition Table (CCDT). O local da CCDT é especificado usando as variáveis de ambiente *MQCHLIB* e *MQCHLTAB*
9. A Client Channel Definition Table (CCDT). O local da CCDT é e especificado usando o arquivo de configuração do cliente

Para os itens 1-3, a definição de canal é construída campo por campo a partir de valores fornecidos pelo aplicativo. Esses valores podem ser fornecidos usando diferentes interfaces e podem existir vários valores para cada uma. Os valores do campo são incluídos na definição de canal seguindo a ordem de prioridade fornecida:

1. O valor de *connName* no construtor MQQueueManager
2. Valores de propriedades da hashtable MQQueueManager.properties
3. Valores de propriedades da hashtable MQEnvironment.properties
4. Valores configurados como campos MQEnvironment (por exemplo, MQEnvironment.Hostname, MQEnvironment.Port)

Para os itens 4-6, a definição de canal inteira é fornecida como o valor. Campos não especificados na definição de canal assumem os padrões do sistema. Nenhum valor de outros métodos de definição de canais e seus campos é fundido com estas especificações.

Para os itens 7-9, a definição de canal inteira é tomada a partir da CCDT. Os campos que não foram explicitamente especificados quando o canal foi definido tomam os padrões do sistema. Nenhum valor de outros métodos de definição de canais e seus campos é fundido com estas especificações.

Usando saídas de canal em IBM MQ .NET

Se você usar ligações do cliente, poderá usar saídas do canal como para qualquer outra conexão do cliente. Se você usar ligações gerenciadas, deverá gravar um programa de saída que implementa uma interface apropriada.

Ligações do Cliente

Se você usar ligações do cliente, poderá usar saídas do canal conforme descrito em Saídas do canal. Não é possível usar saídas do canal gravadas para ligações gerenciadas.

Ligações Gerenciadas

Se você usar uma conexão gerenciada, para implementar uma saída, defina uma nova classe .NET que implemente a interface apropriada. Três interfaces de saída são definidas no pacote do IBM MQ:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Nota: As saídas de usuário gravadas usando estas interfaces não são suportadas como saídas do canal no ambiente não gerenciado.

A amostra a seguir define uma classe que implementa todas as três:

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
```

```
      // complete the body of the send exit here
   }


   // This method comes from the receive exit
   byte[] ReceiveExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefinition,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
   {
      // complete the body of the receive exit here
   }


   // This method comes from the security exit
   byte[] SecurityExit(MQChannelExit       channelExitParms,
                       MQChannelDefinition channelDefParms,
                       byte[]              dataBuffer
                       ref int             dataOffset
                       ref int             dataLength
                       ref int             dataMaxLength)
   {
      // complete the body of the security exit here
   }

}
```

Cada saída passa por uma instância de objeto MQChannelExit e MQChannelDefinition. Estes objetos representam as estruturas MQCXP e MQCD definidas na interface processual.

Os dados a serem enviados por uma saída de envio e os dados recebidos em uma saída de segurança ou recebimento são especificados usando os parâmetros de saída.

Na entrada, os dados no deslocamento *dataOffset* com comprimento *dataLength* na matriz de byte *dataBuffer* são os dados que estão prestes a serem enviados por uma saída de envio e os dados recebidos em uma saída de segurança ou recebimento. O parâmetro *dataMaxLength* fornece o comprimento máximo (de *dataOffset*) disponível para a saída em *dataBuffer*. Nota: Para uma saída de segurança, é possível que o dataBuffer seja nulo se esta for a primeira vez que a saída é chamada ou se a extremidade do parceiro elegeu para não enviar dados.

No retorno, o valor de *dataOffset* e *dataLength* deve ser configurado para apontar para o deslocamento e o comprimento na matriz de bytes retornada que as classes .NET deverão então usar. Para uma saída de envio, isto indica os dados que ela deve enviar e para uma saída de segurança ou recebimento, os dados que devem ser interpretados. A saída normalmente deve retornar uma matriz de byte; exceções são uma saída de segurança que poderia escolher não enviar dados e qualquer saída chamada com as razões INIT ou TERM. A forma mais simples de saída que pode ser gravada, portanto, é uma que não faz nada mais que retornar dataBuffer:

O corpo de saída mais simples possível é:

```
{
  return dataBuffer;
}
```

Classe MQChannelDefinition

O ID do usuário e a senha que são especificados com o aplicativo cliente .NET gerenciado são configurados na classe MQChannelDefinition do IBM MQ .NET que é passada para a saída de segurança do cliente. A saída de segurança copia o ID do usuário e a senha nos campos MQCD.RemoteUserIdentifier e MQCD.RemotePassword (veja “Escrevendo uma saída de segurança” na página 986).

Especificando Saídas do Canal (Cliente Gerenciado)

Se você especificar um nome de canal e nome de conexão ao criar seu objeto MQQueueManager (no construtor MQEnvironment ou MQQueueManager) será possível especificar saídas do canal de duas maneiras.

Na ordem de precedência, eles são:

1. Transmitir propriedades hashtable MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY ou MQC.RECEIVE_EXIT_PROPERTY no construtor MQQueueManager.
2. Configurar as propriedades MQEnvironment SecurityExit, SendExit ou ReceiveExit.

Se você não especificar um nome de canal e nome de conexão, as saídas do canal a usar vêm da definição de canal selecionada a partir de uma tabela de client channel definition table (CCDT). Não é possível substituir os valores armazenados na definição de canal. Veja Client Channel Definition Table e "Usando uma tabela de definição de canal do cliente com o .NET" na página 601 para obter informações adicionais sobre tabelas de definição de canal.

Em cada caso, a especificação tem a forma de uma sequência com o formato a seguir:

```
Assembly_name(Class_name)
```

Class_name é o nome completo, incluindo a especificação de namespace, de uma classe do .NET que implementa a interface IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit ou IBM.WMQ.MQReceiveExit (conforme apropriado). *Assembly_name* é o local completo, incluindo extensão de arquivo, da montagem que hospeda a classe. O comprimento da sequência é limitado a 999 caracteres se você usar as propriedades de MQEnvironment ou MQQueueManager. Entretanto, se o nome de saída do canal for especificado no CCDT, ele será limitado a 128 caracteres. Quando necessário, o código do cliente .NET carrega e cria uma instância da classe especificada analisando a especificação de sequência.

Especificando Dados do Usuário de Saída do Canal (Cliente Gerenciado)

As saídas do canal podem ter dados do usuário associados a elas. Se você especificar um nome de canal e nome de conexão ao criar seu objeto MQQueueManager (no construtor MQEnvironment ou MQQueueManager), poderá especificar os dados do usuário de duas maneiras.

Na ordem de precedência, eles são:

1. Transmitir propriedades hashtable MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY ou MQC.RECEIVE_USERDATA_PROPERTY no construtor MQQueueManager.
2. Configurar as propriedades MQEnvironment SecurityUserData, SendUserData ou ReceiveUserData.

Se você não especificar um nome de canal e um nome de conexão, os valores dos dados do usuário de saída para usar virão da definição de canal selecionada na client channel definition table (CCDT). Não é possível substituir os valores armazenados na definição de canal. Veja Client Channel Definition Table e "Usando uma tabela de definição de canal do cliente com o .NET" na página 601 para obter informações adicionais sobre tabelas de definição de canal.

Em cada caso, a especificação é uma sequência, limitada a 32 caracteres.

Reconexão automática do cliente em .NET

É possível fazer com que seu cliente se reconecte automaticamente a um gerenciador de filas durante uma quebra de conexão inesperada.

Um cliente pode se desconectar inesperadamente a partir de um gerenciador de filas se, por exemplo, o gerenciador de filas parar ou se a rede ou o servidor falhar.

Sem a reconexão automática do cliente, um erro será produzido quando a conexão falhar. É possível usar o código de erro para ajudá-lo a restabelecer a conexão.

Um cliente que usa o recurso de reconexão de cliente automático é denominado um cliente reconectável. Para criar um cliente reconectável, especifique certas opções denominadas opções de reconexão enquanto se conecta ao gerenciador de filas.

Se o aplicativo cliente for um cliente IBM MQ .NET, ele pode optar por obter a reconexão de cliente automática especificando um valor apropriado para CONNECT_OPTIONS_PROPERTY ao usar a classe

MQQueueManager para criar um gerenciador de filas. Consulte Opções de reconexão para obter detalhes dos valores CONNECT_OPTIONS_PROPERTY.

É possível selecionar se o aplicativo cliente sempre se conecta e reconecta a um gerenciador de filas do mesmo nome, com o mesmo gerenciador de filas ou a qualquer conjunto de gerenciadores de filas que foi definido com o mesmo QMNAME na tabela de conexão do cliente (consulte Grupos do gerenciadores de filas na CCDT para obter detalhes).

Suporte a Segurança da Camada de Transporte (TLS) para .NET

Os aplicativos cliente do IBM MQ classes for .NET suportam a criptografia de Segurança da Camada de Transporte (TLS). O protocolo TLS fornece segurança de comunicações sobre a Internet e permite que os aplicativos cliente/servidor se comuniquem de uma forma que seja confidencial e confiável.

Conceitos relacionados

Suporte de TLS do cliente gerenciado pelo IBM MQ.NET

Protocolos de segurança criptográficos: TLS

Suporte a TLS para o cliente não gerenciado do .NET

O suporte TLS para o cliente .NET não gerenciado é baseado no C MQI e IBM Global Security Kit (GSKit). O C MQI manipula as operações TLS e o GSKit implementa os protocolos de soquete seguro TLS.

Ativando o TLS para o cliente .NET não gerenciado

O TLS é suportado somente para conexões do cliente. Para ativar o TLS, deve-se especificar o CipherSpec a ser usado para comunicação com o gerenciador de filas, que deve corresponder ao CipherSpec configurado no canal de destino.

Para ativar o TLS, especifique o CipherSpec usando a variável de membro estático SSLCipherSpec de MQEnvironment. O exemplo a seguir conecta-se a um canal SVRCONN denominado SECURE.SVRCONN.CHANNEL, que foi configurado para requerer o TLS com um CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Veja Especificando CipherSpecs para obter uma lista de CipherSpecs.

A propriedade SSLCipherSpec também pode ser configurada usando MQC.SSL_CIPHER_SPEC_PROPERTY na hashtable das propriedades da conexão.

Para conectar-se com êxito usando TLS, o keystore do cliente deve ser configurado com a cadeia de certificados raiz de Autoridade de Certificação a partir da qual o certificado apresentado pelo gerenciador de filas pode ser autenticado. Similarmente, se SSLClientAuth no canal SVRCONN tiver sido configurado como MQSSL_CLIENT_AUTH_REQUIRED, o keystore do cliente deverá conter um certificado pessoal de identificação que seja confiável pelo gerenciador de filas.

Usando o nome distinto do gerenciador de filas

O gerenciador de filas se identifica usando um certificado TLS, que contém um *Nome distinto* (DN).

Um aplicativo cliente do IBM MQ .NET pode usar este DN para assegurar que ele esteja se comunicando com o gerenciador de filas correto. Um padrão de DN é especificado usando a variável sslPeerName de MQEnvironment. Por exemplo, configurar:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

permite que a conexão seja bem-sucedida somente se o gerenciador de filas apresentar um certificado com um Nome Comum iniciando QMGR., e pelo menos dois nomes de Unidade Organizacional, o primeiro dos quais deve ser IBM e o segundo WEBSPHERE

A propriedade SSLPeerName também pode ser configurada usando MQC.SSL_PEER_NAME_PROPERTY na hashtable das propriedades da conexão. Para obter mais informações sobre Nomes Distintos e regras para configurar nomes de mesmo nível, consulte Protegendo IBM MQ.

Se SSLPeerName estiver configurado, as conexões serão bem-sucedidas somente se ele estiver configurado com um padrão válido e o gerenciador de filas apresentar um certificado correspondente.

Manipulação de erros ao usar o TLS

Os códigos de razão a seguir podem ser emitidos pelo IBM MQ classes for .NET ao conectar-se a um gerenciador de filas usando TLS:

MQRC_SSL_NOT_ALLOWED
A propriedade SSLCipherSpec foi configurada, mas a conexão de ligações foi usada. Somente a conexão do cliente suporta TLS.

MQRC_SSL_PEER_NAME_MISMATCH
O padrão de DN especificado na propriedade SSLPeerName não correspondia ao DN apresentado pelo gerenciador de filas.

MQRC_SSL_PEER_NAME_ERROR
O padrão de DN especificado na propriedade SSLPeerName não era válido.

MQRC_KEY_REPOSITORY_ERROR
O local do repositório de chaves não está especificado, não é válido ou não pode ser acessado

Suporte de TLS para o cliente .NET gerenciado

O cliente .NET gerenciado usa as bibliotecas Microsoft .NET Framework para implementar protocolos de soquete seguro TLS. A classe System.Net.SecuritySslStream do Microsoft opera como um fluxo sobre soquetes TCP conectados e envia e recebe dados por meio dessa conexão do soquete.

O nível mínimo necessário .NET Framework é .NET Framework v3.5. O nível de suporte de Algoritmo de Cifra é baseado no nível .NET Framework que o aplicativo está usando:

- Para aplicativos baseados em .NET Framework níveis 3.5 e 4.0, os protocolos de soquete seguro disponíveis são SSL 3.0 e TSL 1.0.
- Para aplicativos baseados no .NET Framework nível 4.5, os protocolos de soquete seguro disponíveis são SSL 3.0, TLS 1.1 e TLS 1.2.

Pode ser necessário mover aplicativos que esperam suporte de protocolo TLS superior para uma versão mais recente da estrutura, conforme definido para o suporte de Segurança do Microsoft no .NET Framework.

Os principais recursos de suporte de TLS do cliente .NET gerenciado são os seguintes:

Suporte do protocolo TLS
O suporte de TLS para o cliente gerenciado .NET é definido por meio da classe SSLStream do .NET e depende do .NET Framework que o aplicativo está usando. Para obter informações adicionais, consulte “Suporte do protocolo TLS para o cliente .NET gerenciado” na página 608.

Suporte a CipherSpec
As configurações do TLS para o cliente gerenciado .NET são como para o fluxo TLS do Microsoft.NET. Para obter mais informações, consulte “Suporte a CipherSpec para o cliente gerenciado do .NET” na página 608 e “Mapeamentos CipherSpec para o cliente gerenciado .NET” na página 610.

Repositórios de chaves
O repositório de chaves no lado do cliente é um keystore do Windows. O repositório do lado do servidor é um tipo de repositório Cryptographic Message Syntax (CMS). Para obter informações adicionais, consulte “Repositórios de chave para o cliente gerenciado do .NET” na página 612.

Certificados
É possível usar certificados TLS autoassinados para implementar autenticação mútua entre um cliente e um gerenciador de filas. Para obter informações adicionais, consulte “Usando certificados para o cliente .NET gerenciado” na página 612.

SSLPEERNAME
No .NET, os aplicativos podem usar o atributo SSLPEERNAME opcional para especificar um padrão de Nome distinto (DN). Para obter informações adicionais, consulte “SSLPEERNAME” na página 613.

Conformidade com FIPS
Ativar FIPS programaticamente não é suportado pela biblioteca de Segurança do Microsoft.NET. A ativação do FIPS é controlada pela configuração da Política de grupo do Windows.

Conformidade do NSA Suite B
O IBM MQ implementa o RFC 6460. A implementação do Microsoft.NET para NSA suite B é 5430. É suportado do .NET Framework 3.5 em diante.

Reconfiguração ou renegociação de chave secreta
Embora a classe SSLStream não suporte a reconfiguração ou renegociação de chave secreta, para consistência com outros clientes IBM MQ , o cliente gerenciado do .NET permite que os aplicativos sejam configurados como `SSLKeyResetCount` Para obter mais informações, consulte “Reconfiguração ou renegociação de chave secreta para o cliente .NET gerenciado” na página 613.

Verificação de revogação
A classe SSLStream suporta a verificação de revogação de certificado, que é feita automaticamente pelo mecanismo de encadeamento de certificados. Para obter informações adicionais, consulte “Verificação de revogação” na página 613.

Suporte de saída de segurança do IBM MQ
A classe `SSLStream` fornece suporte limitado para saídas de segurança IBM MQ . Consultar certificados locais e remotos para obter `SSLPeerNamePtr`(DN do Assunto) e `SSLRemCertIssNamePtr` (DN do Emissor) é possível, pois isso é suportado no Microsoft.NET. No entanto, não há suporte para obter atributos como DNQ é, UNSTRUCTUREDNAME e UNSTRUCTUREDADDRESS, portanto, esses valores não podem ser recuperados usando as saídas.

Suporte a hardware criptográfico
Hardware criptográfico não é suportado para o cliente .NET gerenciado.

Suporte para TLS1.3 em clientes IBM MQ .NET e XMS .NET gerenciados

V 9.4.0

A partir de IBM MQ 9.4.0, IBM MQ .NET e XMS .NET clientes suportam TLS1.3 desde que o sistema operacional suporte TLS1.3.

O cliente .NET gerenciado usa as bibliotecas Microsoft .NET Framework para implementar protocolos de soquete seguro TLS. A classe Microsoft `System.Net.SecuritySslStream` opera como um fluxo sobre soquetes TCP conectados e envia e recebe dados sobre essa conexão de soquete.

No Windows, .NET usa SCHANEXO L e no Linux .NET usa OpenSSL para Comunicação SSL

Windows Para IBM MQ .NET aplicativos clientes em execução no Windows

O Microsoft anunciou que Windows 11 e Windows Server 2022 suportam cifras TSL1.3 por padrão.

Os conjuntos de cifras TLS_AES_128_GCM_SHA256 e TLS_AES_256_GCM_SHA384 são ativados por padrão em ambas as versões do Windows

Atenção:

- TLS_CHACHA20_POLY1305_SHA256 Cifra Suite não é ativado por padrão, mas é suportado.
- Para um cliente do IBM MQ .NET com TLS1.3 ativado, para conectar-se a um gerenciador de filas com êxito, IBM Global Security Kit (GSKit) 8.0.55.29 é a versão mínima que é a necessária no lado do gerenciador de filas

Linux Para IBM MQ .NET aplicativos clientes em execução no Linux

Como .NET usa OpenSSL no Linux for SSL Communication, para usar TLS1.3, OpenSSL v1.1.1 é o requisito mínimo.

Além disso, como o .NET usa OpenSSL no Linux, todas as cifras suportadas pelo OpenSSL também devem funcionar para .NET

OpenSSL suporta os CipherSpecs a seguir para TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

Conceitos relacionados

"Mapeamentos CipherSpec para o cliente gerenciado .NET" na página 610
A interface do IBM MQ.NET mantém uma tabela de mapeamento de IBM MQ para Microsoft.NET que é usada para determinar a versão do protocolo TLS que o cliente gerenciado precisa usar para estabelecer uma conexão segura com um gerenciador de filas.

Suporte do protocolo TLS para o cliente .NET gerenciado
O suporte de TLS ao IBM MQ.NET é baseado na classe SSLStream do .NET.

Nota: O suporte do protocolo TLS para o cliente .NET gerenciado depende do nível do .NET Framework que o aplicativo está usando. Para obter mais informações, consulte "Suporte de TLS para o cliente .NET gerenciado" na página 606.

Para que a classe SSLStream do Microsoft.NET inicialize o TLS e execute um handshake com o gerenciador de filas, um dos parâmetros necessários que devem ser configurados é **SSLProtocol**, em que deve-se especificar o número da versão do TLS, que deve ser um dos valores a seguir:

- SSL3.0
- TLS1.0
- TLS1.2

O valor desse parâmetro é fortemente acoplado à família de Protocolo à qual o CipherSpec preferencial pertence. Quando SSLStream inicia um handshake TLS com o servidor (gerenciador de filas), ele usa a versão do TLS especificada em **SSLProtocol** para identificar a lista de CipherSpecs a serem usados para negociação.

IBM MQ.NET não disponibiliza nenhuma propriedade para uso pelos aplicativos para configurar esse valor. Em vez disso, o IBM MQ usa uma tabela de mapeamento para mapear internamente o CipherSpec configurado para a família de Protocolo e identifica a versão de SSLProtocol a ser usada. Esta tabela mostra o mapeamento de cada CipherSpec suportado entre Microsoft.NET e IBM MQ e a versão de Protocolo a qual pertencem. Para obter mais informações, consulte "Mapeamentos CipherSpec para o cliente gerenciado .NET" na página 610.

Suporte a CipherSpec para o cliente gerenciado do .NET
As configurações de CipherSpec para um aplicativo são usadas durante o handshake com o servidor.

Os clientes do IBM MQ permitem que você configure um valor CipherSpec usado durante o handshake com o gerenciador de filas. Os clientes do IBM MQ devem configurar um CipherSpec válido para conexão segura para estabelecer, preferencialmente, o CipherSpec especificado na política de grupo do Windows. Deixar esse campo em branco indica um canal de texto simples sem qualquer segurança nos soquetes.

Para o cliente gerenciado IBM MQ.NET, as configurações do TLS são para a classe SSLStream do Microsoft.NET. Para SSLStream, um CipherSpec ou uma lista de preferências de CipherSpecs, pode ser configurado apenas na política de grupo do Windows, que é uma configuração de computador inteiro. SSLStream, em seguida, usa a lista de preferências ou o CipherSpec especificado durante o handshake com o servidor. No caso de outros clientes IBM MQ, a propriedade CipherSpec pode ser configurada no aplicativo na definição de canal do IBM MQ e a mesma configuração é usada para a negociação do

TLS. Como resultado dessa restrição, o handshake TLS pode negociar qualquer CipherSpec suportado, independentemente do que seja especificado na configuração de canal do IBM MQ. Portanto, é provável que isso resultará em erros AMQ9631 no gerenciador de filas. Para evitar este erro, configure o mesmo CipherSpec como aquele que você configurou no aplicativo como a configuração TLS na política de grupo do Windows.

O novo código do cliente TLS do IBM MQ.NET verifica somente se a versão correta de protocolo foi negociada. A versão de protocolo TLS é derivada do CipherSpec que o aplicativo configura e é usada para o handshake TLS com o servidor (gerenciador de filas). Por isso, é requerida pelo design para configurar o CipherSpec no aplicativo cliente gerenciado pelo IBM MQ.NET. Se o CipherSpec configurado pelo cliente IBM MQ for algo diferente daquele dos protocolos SSL 3.0, TLS 1.0 e TLS 1.2 , o cliente IBM MQ gerenciado .NET negociaria por padrão com qualquer uma das cifras dos protocolos SSL 3.0 ou TLS 1.0 e não relataria um erro.

Nota: Se o valor de CipherSpec fornecido pelo aplicativo não for um CipherSpec conhecido por IBM MQ, então o cliente IBM MQ gerenciado .NET desconsidera-o e negocia a conexão com base na política de grupo do sistema Windows.

Configurando um CipherSpec

Há três maneiras de configurar um CipherSpec:

Classe MQEnvironment do .NET

O exemplo a seguir mostra como configurar um CipherSpec com a classe MQEnvironment.

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

Propriedade CipherSpec do TLS

O exemplo a seguir mostra como configurar um CipherSpec incluindo um parâmetro hashtable no construtor MQQueueManager.

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Política de grupo do Windows

Quando uma lista de conjuntos de cifras é configurada por meio do console de gerenciamento de políticas do grupo do Windows, a definição de canal SVRCONN deve especificar um CipherSpec correspondente. Um CipherSpec correspondente pode ser um valor genérico, como "ANY_TLS12_OR_HIGHER", ou um valor específico que mapeia para o Conjunto de Cifras mais alto que seria negociado na lista ordenada. O uso de valores genéricos de CipherSpec é recomendado com clientes .NET porque evita a necessidade de mudar a configuração do CipherSpec SVRCONN quando a ordem da lista de clientes muda.

Uso do CCDT

IBM MQ.NET apenas suporta o Client Channel Definition Tables (arquivos .TAB) que estão em um computador local. Os arquivos CCDT existentes que possuem um conjunto de valores CipherSpec podem ser usados para as conexões do IBM MQ.NET. No entanto, o conjunto de valores CipherSpec no canal de conexão do cliente determina a versão do protocolo TLS e também deve corresponder ao CipherSpec configurado na política de grupo do Windows.

Conceitos relacionados

"Configurando o ambiente IBM MQ" na página 596
Antes de usar a conexão do cliente para se conectar a um gerenciador de filas, deve-se configurar o ambiente do IBM MQ.

"Suporte de TLS para o cliente .NET gerenciado" na página 606
O cliente .NET gerenciado usa as bibliotecas Microsoft .NET Framework para implementar protocolos de soquete seguro TLS. A classe System.Net.SecuritySslStream do Microsoft opera como um fluxo sobre soquetes TCP conectados e envia e recebe dados por meio dessa conexão do soquete.

Tarefas relacionadas

Especificando CipherSpecs

Referências relacionadas

Classe MQEnvironment do .NET

Mapeamentos CipherSpec para o cliente gerenciado .NET

A interface do IBM MQ.NET mantém uma tabela de mapeamento de IBM MQ para Microsoft.NET que é usada para determinar a versão do protocolo TLS que o cliente gerenciado precisa usar para estabelecer uma conexão segura com um gerenciador de filas.

Se um CipherSpec for especificado no canal SVRCONN, depois que o handshake TLS for concluído, o gerenciador de filas tentará corresponder esse CipherSpec com o CipherSpec negociado que o aplicativo cliente está usando. Se o gerenciador de filas não puder localizar um CipherSpec correspondente, a comunicação falhará com o erro AMQ9631.

A interface do IBM MQ.NET mantém uma tabela de mapeamento do IBM MQ para Microsoft.NET CipherSpec. Esta tabela é usada para determinar a versão do protocolo TLS que o cliente deseja usar para estabelecer uma conexão de soquete seguro com o gerenciador de filas. Com base no valor de SSLCipherSpec, a versão SSLProtocol pode ser TLS 1.0 ou TLS 1.2, dependendo de qual versão do Microsoft.NET Framework está sendo usada.

Certifique-se de fornecer o valor correto de SSLCipherSpec, pois especificar um valor incorreto pode resultar no uso dos protocolos SSL 3.0 ou TLS 1.0.

Tabela 78. Tabela de mapeamento de IBM MQ e Microsoft.NET

IBM MQ CipherSpec	Microsoft.NET CipherSpec	Versão do TLS
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2

Tabela 78. Tabela de mapeamento de IBM MQ e Microsoft.NET (continuação)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Versão do TLS**
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.4.0 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
V 9.4.0 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

Notas:

1. Deprecated Esse CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.

Conceitos relacionados

"Suporte de TLS para o cliente .NET gerenciado" na página 606

O cliente .NET gerenciado usa as bibliotecas Microsoft .NET Framework para implementar protocolos de soquete seguro TLS. A classe System.Net.SecuritySslStream do Microsoft opera como um fluxo sobre soquetes TCP conectados e envia e recebe dados por meio dessa conexão do soquete.

Repositórios de chave para o cliente gerenciado do .NET
O repositório de chaves usado por clientes .NET gerenciados é o repositório de chaves Windows. Os certificados e as chaves privadas devem estar disponíveis no repositório de chaves do usuário ou do sistema para poderem ser utilizados pelo aplicativo cliente para identidade e confiança durante um handshake TLS.

Lado do cliente

No aplicativo, é possível configurar um dos valores a seguir para o repositório de chaves:

- `"*USUÁRIO"`: IBM MQ.NET acessa a loja de certificados do usuário atual para recuperar os certificados do cliente.
- `"*SISTEMA"`: IBM MQ.NET acessa a conta do computador local para recuperar os certificados.

Os certificados de cliente devem ser armazenados no Meu armazenamento de certificados do usuário ou da conta do computador. Todos os certificados do servidor (CA) devem ser armazenados no diretório-raiz do armazenamento de certificados.

Nota: Você pode armazenar mais de um certificado em um único arquivo nos seguintes formatos:

- Troca de Informações Pessoal -PKCS #12 (.PFX, .P12)
- Padrão de Sintaxe de Mensagem Criptográfica - Certificados PKCS #7 (.P7B)
- Microsoft Armazenamento de Certificados Serializados (.SST)

Usando certificados para o cliente .NET gerenciado
Para certificados de cliente, o cliente IBM MQ gerenciado .NET acessa o keystore Windows e carrega todos os certificados do cliente que são correspondidos pelo rótulo certificado ou pela sequência.

Ao selecionar um certificado a ser usado, o cliente IBM MQ gerenciado .NET sempre usa o primeiro certificado correspondente para o handshake SSLStream TLS.

Certificados correspondentes por rótulo de certificado

Se você configurar o rótulo do certificado, o cliente IBM MQ gerenciado do .NET procura o armazenamento de certificados Windows com o nome do rótulo fornecido para identificar o certificado de cliente. Carrega todos os certificados correspondentes e usa o primeiro certificado na lista. Há duas opções para configurar o rótulo do certificado:

- O rótulo do certificado pode ser configurado na classe MQEnvironment acessando MQEnvironment.CertificateLabel.
- O rótulo do certificado também pode ser configurado em propriedades de uma tabela hash, fornecido como parâmetro de entrada com o construtor MQQueueManager, conforme mostrado no exemplo a seguir.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

O name("CertificateLabel") e o valor fazem distinção entre maiúsculas e minúsculas.

Correspondendo certificados por sequência

Se o rótulo do certificado não estiver configurado, então, o certificado que corresponder à sequência `"ibmwebspheremq"` e o usuário atual com logon efetuado (em minúsculas) será procurado e usado.

Tarefas relacionadas
Conectando um Cliente a um Gerenciador de Filas de Forma Segura

Referências relacionadas
Classe MQEnvironment do .NET

SSLPEERNAME
O atributo SSLPEERNAME é usado para verificar o Nome Distinto (DN) do certificado do gerenciador de filas de peer.

No IBM MQ.NET, os aplicativos podem usar SSLPEERNAME para especificar um padrão de nome distinto, conforme mostrado no exemplo a seguir.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Como para outros clientes do IBM MQ, SSLPEERNAME é um parâmetro opcional.

Se o valor de SSLPEERNAME não for configurado, o cliente gerenciado do IBM MQ.NET não executará nenhuma validação de certificado Remote(Server) e o cliente gerenciado apenas aceitará o certificado Remote(/server) no estado em que se encontra.

A maneira na qual você configura SSLPEERNAME depende de qual das ofertas de pilha do IBM MQ que você está usando.

IBM MQ classes for .NET
Há três opções conforme a seguir.

1. Configure MQEnvironment.SSLPeerName na classe MQEnvironment.
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. Use o construtor do gerenciador de filas `MQQueueManager (String queueManagerName, Hashtable properties)` Forneça o SSLPEERNAME no `Hashtable properties` como para a opção 2.

XMS .NET
Configure o nome de peer SSL na connection factory:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF
Inclua SslPeerName como um campo de ponto-e-vírgula separado no URI.

Referências relacionadas
Classe MQEnvironment do .NET

Reconfiguração ou renegociação de chave secreta para o cliente .NET gerenciado
A classe SSLStream não suporta reconfiguração ou renegociação de chave secreta. No entanto, para ser consistente com outros clientes IBM MQ , o cliente IBM MQ gerenciado .NET permite que os aplicativos configure **SSLKeyResetCount**.

Quando o limite for alcançado, o IBM MQ.NET se desconectará do gerenciador de filas e os aplicativos serão notificados sobre isso como uma exceção com MQRC_CONNECTION_BROKEN como o código de razão. Os aplicativos podem escolher manipular a exceção e restabelecer as conexões ou ativar a opção MQCNO_RECONNECT para IBM MQ.NET para se reconectar automaticamente ao gerenciador de filas.

Ativar o recurso de reconexão automática do cliente significa que, quando a contagem de reconfiguração de chave é atingida, todas as conexões existentes são desativadas e o cliente IBM MQ.NET recria todas as conexões novamente. Para obter mais informações sobre a reconexão automática do cliente, consulte Reconexão automática do cliente.

Conceitos relacionados
Reconfigurando as chaves secretas SSL e TLS

Verificação de revogação
A classe SSLStream suporta a verificação de revogação de certificado.

A verificação de revogação é feita automaticamente pelo mecanismo de encadeamento de certificado. Isso se aplica ao Online Certificate Status Protocol (OCSP) e às Listas de revogação de certificado (CRLs). A classe SSLStream usa a revogação de certificado que usa apenas o servidor especificado no certificado, que é o servidor ditado pelo próprio certificado. É possível para as extensões do CDP de HTTP e solicitações HTTP do OCSP para proxy através do servidor proxy HTTP.

A maneira na qual você configura a verificação de revogação depende de qual das ofertas de pilha do IBM MQ você está usando.

IBM MQ.NET
A verificação de revogação pode ser configurada, acessando a propriedade **MQEnvironment.SSLCertRevocationCheck** no arquivo de classe `MQEnvironment.cs`.

XMS .NET
A verificação de revogação pode ser configurada no contexto da propriedade da connection factory, conforme mostrado no exemplo a seguir.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF
A verificação de revogação pode ser configurada no URI usando a seguinte convenção de nomenclatura.

```
"SslCertRevocationCheck=true"
```

Configurando o TLS para o IBM MQ .NET gerenciado
Configurar o TLS para o IBM MQ .NET gerenciado consiste em criar os certificados de assinante, em seguida, configurar o lado do servidor, o lado do cliente e o programa de aplicativo.

Sobre esta tarefa

Para configurar o TLS, deve-se criar primeiro os certificados de assinante apropriados. Os certificados de assinante podem ser autoassinados ou podem ser certificados fornecidos por uma autoridade de certificação. Embora certificados autoassinados possam ser usados em um sistema de desenvolvimento, de teste ou de pré-produção, não os use em um sistema de produção. Em um sistema de produção, use certificados que tenha obtido de uma autoridade de certificação (CA) externa confiável.

Procedimento

1. Crie os certificados de assinante.
   a) Para criar certificados autoassinados, use os comandos **runmqakm** ou V 9.4.0 **runmqktool** ..
      Para obter mais informações, consulte Criando um certificado pessoal autoassinado no AIX, Linux, and Windows.
   b) Para obter certificados para o gerenciador de filas e clientes de uma autoridade de certificação (CA), siga as instruções em Obtendo certificados pessoais de uma autoridade de certificação.
2. Configure o lado do servidor.
   a) Configure o TLS no gerenciador de filas, usando IBM Global Security Kit (GSKit), conforme descrito em Conectando um cliente a um gerenciador de filas com segurança.
   b) Configure os atributos TLS do canal SVRCONN:
      - Configure **SSLCAUTH** como `REQUIRED` ou `OPTIONAL`..
      - Configure **SSLCIPH** para um CipherSpec apropriado.

      Para obter informações adicionais, consulte “Ativando o TLS para o cliente .NET não gerenciado” na página 605.

3. Configure o lado do cliente.
   a) Importe os certificados do cliente para o armazenamento de certificados do Windows (sob a conta Usuário/Computador).

      O IBM MQ .NET acessa os certificados do cliente a partir do armazenamento de certificados do Windows, portanto, deve-se importar os certificados para o armazenamento de certificados do Windows para estabelecer uma conexão de soquete segura para o IBM MQ. Para obter mais informações sobre como acessar o keystore do Windows e importar os certificados do lado do cliente, veja Importar ou exportar certificados e chaves privadas.

   b) Forneça o CertificateLabel conforme descrito em Conectando um cliente a um gerenciador de filas de forma segura.

   c) Se necessário, edite a Política de grupo do Windows para configurar o CipherSpec, em seguida, para que as atualizações da Política de grupo do Windows entrem em vigor, reinicie o computador.
4. Configure o programa de aplicativo.
   a) Configure o valor de MQEnvironment ou SSLCipherSpec para denotar a conexão como uma conexão segura.

      O valor que você especifica é usado para identificar o protocolo que está sendo usado (TLS). O CipherSpec configurado deve ser um dos CipherSpecs da versão de SSLProtocol suportada e pode ser, de preferência, o mesmo especificado na Política de grupo do Windows. (A versão de SSLProtocol suportada depende da estrutura do .NET usada. A versão SSLProtocol pode ser TLS 1.0 ou TLS 1.2, dependendo de qual versão do Microsoft .NET Framework está sendo usada.)

      Nota: Se o valor de CipherSpec fornecido pelo aplicativo não for um CipherSpec conhecido por IBM MQ, então o cliente IBM MQ gerenciado .NET desconsidera-o e negocia a conexão com base na política de grupo do sistema Windows.

   b) Configure a propriedade SSLKeyRepository para "*SYSTEM" ou "*USER".

   c) Opcional: Configure SSLPEERNAME para o nome distinto (DN) do certificado do servidor.

   d) Forneça o CertificateLabel conforme descrito em Conectando um cliente a um gerenciador de filas de forma segura.

   e) Configure quaisquer parâmetros opcionais adicionais que precise, como KeyResetCount, CertificationRevocationCheck, e ative o FIPS.

Exemplos de como configurar o protocolo TLS e repositório de chaves TLS

Para o .NET de base, é possível configurar o protocolo TLS e repositório de chaves TLS por meio da classe MQEnvironment conforme mostrado no exemplo a seguir:

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Como alternativa, é possível configurar o protocolo TLS e o repositório de chaves TLS fornecendo uma hashtable como parte do construtor MQQueueManager, conforme mostrado no exemplo a seguir.

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Como proceder a seguir

Para obter mais informações sobre como iniciar o desenvolvimento de aplicativos TLS gerenciados pelo IBM MQ .NET, veja “Gravando um aplicativo simples” na página 616.

Referências relacionadas

Classe MQEnvironment do .NET

KeyResetCount (MQLONG)
Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Gravando um aplicativo simples
Dicas para gravar um aplicativo TLS IBM MQ gerenciado .NET simples, incluindo exemplos para configurar as propriedades SSL para connection factories, criar uma instância do gerenciador de filas, conexão, sessão e destino e enviar uma mensagem de teste.

Antes de começar

Deve-se configurar primeiro o TLS para o IBM MQ.NET gerenciado, conforme descrito em "Configurando o TLS para o IBM MQ .NET gerenciado" na página 614.

Para obter a configuração do programa de aplicativo na base .NET, configure as propriedades SSL usando a classe MQEnvironment ou fornecendo um hashtable como parte do construtor MQQueueManager.

Para obter a configuração do programa de aplicativo em XMS .NET, configure as propriedades SSL no contexto da propriedade das connection factories.

Procedimento

1. Configure as propriedades SSL para as connection factories, conforme mostrado nos seguintes exemplos.

   Exemplo para o IBM MQ.NET

   ```
   properties = new Hashtable();
   properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
   properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
   properties.Add(MQC.PORT_PROPERTY, port);
   properties.Add(MQC.CHANNEL_PROPERTY, channelName);
   properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
   properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
   properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
   properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
   properties.Add("CertificateLabel", "ibmwebspheremq");
   MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
   ```

   Exemplo para XMS .NET

   ```
   cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
   cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
   cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
   cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
   cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
   ```

2. Crie a instância do gerenciador de filas, conexões, sessão e destino, conforme mostrado nos seguintes exemplos.

   Exemplo para MQ .NET

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
   MQC.MQOO_FAIL_IF_QUIESCING);
   Console.WriteLine("done");
   ```

   Exemplo para XMS .NET

   ```
   connectionWMQ = cf.CreateConnection();
   // Create session
   sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);
   ```

```
// Create destination
destination = sessionWMQ.CreateQueue(destinationName);

// Create producer
producer = sessionWMQ.CreateProducer(destination);
```

3. Enviar uma mensagem conforme mostrado nos seguintes exemplos.

   Exemplo para MQ .NET

```
// creating a message object
message = new MQMessage();
message.WriteString(messageString);

// putting messages continuously
for (int i = 1; i <= numberOfMsgs; i++)
{
Console.Write("Message " + i + " <" + messageString + ">.. ");
queue.Put(message);
Console.WriteLine("put");
}
```

   Exemplo para XMS .NET

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. Verifique a conexão TLS.

   Confira o status do canal para verificar se a conexão TLS foi estabelecida e está funcionando corretamente.

Configurando rastreio para SSLStream
Para capturar eventos de rastreio e mensagens relacionadas à classe SSLStream, deve-se incluir uma seção de configuração para diagnóstico do sistema no arquivo de configuração de aplicativo para seu aplicativo.

Sobre esta tarefa

Nota:

Esta tarefa aplica-se apenas a IBM MQ classes for .NET Framework O arquivo de configuração de aplicativo não é suportado em IBM MQ classes for .NET (bibliotecas.NET Standard e .NET 6)...

Se você não incluir uma seção de configuração para diagnósticos do sistema no arquivo de configuração do aplicativo, o cliente IBM MQ gerenciado .NET não capturará quaisquer eventos, rastreios ou pontos de depuração relacionados ao TLS e à classe SSLStream

Nota: Iniciar o rastreio do IBM MQ usando **strmqtrc** não captura todo o rastreio de TLS necessário.

Procedimento

1. Crie um arquivo de configuração de aplicativo (App.Config) para seu projeto do aplicativo.
2. Inclua uma seção de configuração de diagnóstico do sistema conforme mostrado no exemplo a seguir.

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
```

```
                        <listeners>
                              <add name="ExternalSourceTrace"/>
                        </listeners>
                  </source>
                  <source name="System.Net.Security">
                        <listeners>
                              <add name="ExternalSourceTrace"/>
                        </listeners>
                  </source>
                  <source name="System.Security">
                        <listeners>
                              <add name="ExternalSourceTrace"/>
                        </listeners>
                  </source>
            </sources>
            <switches>
                  <add name="System.Net" value="Verbose"/>
                  <add name="System.Net.Sockets" value="Verbose"/>
                  <add name="System.Net.Cache" value="Verbose"/>
                  <add name="System.Security" value="Verbose"/>
                  <add name="System.Net.Security" value="Verbose"/>
            </switches>

            <sharedListeners>
                  <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
             </sharedListeners>
          <trace autoflush="true"/>
     </system.diagnostics>
```

Atenção: O campo `Version` da entrada `add name` precisa ser a versão do arquivo .net `amqmdnet.dll` que está sendo usada.

Tarefas relacionadas

Rastreando clientes IBM MQ classes for .NET Framework usando um arquivo de configuração de aplicativo

Aplicativos de amostra para implementar o TLS no .NET gerenciado

Os aplicativos de amostra são fornecidos para mostrar a implementação de TLS para .NET gerenciado no canal customizado do IBM MQ classes for .NET, XMS .NET e IBM MQ para o WCF.

A tabela a seguir mostra o local dos aplicativos de amostra. O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Tabela 79. Local de aplicativos de amostra para implementar o TLS no .NET gerenciado

Oferta de pilha doIBM MQ.NET	**Localização das amostras**
Baseado no .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
Canal customizado do IBM MQ para o WCF	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs

Windows Usando o .NET Monitor

O .NET Monitor é um aplicativo semelhante a um monitor acionador do IBM MQ.

Importante: Ver Recursos que podem ser usados somente com a instalação primária noWindows para obter informações importantes.

É possível criar componentes do .NET que são instanciados sempre que uma mensagem é recebida em uma fila monitorada e que então processar essa mensagem. O .NET Monitor é iniciado pelo comando **runmqdnm** e interrompido pelo comando **endmqdnm**. Para obter detalhes desses comandos, veja runmqdnm e endmqdnm.

Para usar o .NET Monitor, grave um componente que implemente a interface IMQObjectTrigger, que está definida em amqmdnm.dll.

Componentes podem ser transacionais ou não transacionais. Um componente transacional deve herdar de System.EnterpriseServices.ServicedComponent e ser registrado como RequiresTransaction ou SupportsTransaction. Ele não deve ser registrado como RequiresNew, pois o .NET Monitor já iniciou uma transação.

O componente recebe objetos MQQueueManager, MQQueue e MQMessage de **runmqdnm**. Ele também deve receber uma sequência de Parâmetros do Usuário se uma tiver sido especificada usando a opção da linha de comandos *-u* quando runmqdnm foi iniciado. Observe que seu componente recebe o conteúdo de uma mensagem que chegou na fila monitorada em um objeto MQMessage. Ele não precisa se conectar ao gerenciador de filas, abrir a fila ou obter a mensagem em si. O componente então deve processar a mensagem conforme apropriado e retornar o controle ao .NET Monitor.

Se seu componente tiver sido gravado como um componente transacional, ele registrará para confirmar ou retroceder a transação usando os recursos fornecidos pelo System.EnterpriseServices.ServicedComponent.

Como o componente recebe os objetos MQQueueManager e MQQueue, bem como a mensagem, ele possui informações de contexto completas para essa mensagem e pode, por exemplo, abrir outra fila no mesmo gerenciador de filas sem precisar conectar-se separadamente ao IBM MQ.

Windows *Fragmentos de Código de Exemplo*

Este tópico contém dois exemplos de componentes que obtêm uma mensagem do .NET Monitorar e imprimem-na, um usando processamento transacional e o outro usando processamento não transacional. Um terceiro exemplo mostra rotinas do utilitário comuns, aplicáveis aos dois primeiros exemplos. Todos os exemplos estão em C#.

Exemplo 1: Processamento Transacional

```
/********************************************************************/
/* Licensed materials, property of IBM                              */
/* 63H9336                                                          */
/* (C) Copyright IBM Corp. 2005, 2024.                              */
/********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
```

```
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

Exemplo 2: Processamento Não-transacional

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

Exemplo 3: Rotinas Comuns

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;
```

```
namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ---------------------------------------------------------------- */
    /* Default prefix string of the namespace.                           */
    /* ---------------------------------------------------------------- */
    private string prefixText = "dnmsamp";

    /* ---------------------------------------------------------------- */
    /* Constructor that takes the replacement prefix string to use.     */
    /* ---------------------------------------------------------------- */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ---------------------------------------------------------------- */
    /* Display an arbitrary string to the console.                      */
    /* ---------------------------------------------------------------- */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }


 /* ---------------------------------------------------------------- */
    /* Display the content of the message passed to the console.        */
    /* ---------------------------------------------------------------- */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ---------------------------------------------------------------- */
    /* Convert the byte array into a hex string.                        */
    /* ---------------------------------------------------------------- */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

Compilando programas IBM MQ .NET

Comandos de amostra para compilar aplicativos .NET gravados em várias linguagens.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para construir um aplicativo C# usando o IBM MQ classes for .NET, use o seguinte comando:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

Para construir um aplicativo Visual Basic usando o IBM MQ classes for .NET, use o seguinte comando:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Para construir um aplicativo C++ gerenciado usando o IBM MQ classes for .NET, use o seguinte comando:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

Para outros idiomas, consulte a documentação fornecida pelo fornecedor do idioma.

Usando o cliente IBM MQ .NET independente

O cliente IBM MQ .NET oferece a capacidade de empacotar e implementar um conjunto do IBM MQ .NET sem precisar usar a instalação completa do cliente IBM MQ em sistemas de produção para executar seus aplicativos.

Antes de começar

V 9.4.0 Em IBM MQ 9.4.0, a biblioteca do cliente `amqmdnetstd.dll` instalada no local padrão é baseada em .NET 6.

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for .NET. Se você estiver usando um aplicativo .NET 6 , será possível executar esse aplicativo sem que qualquer recompilação seja necessária, fazendo uma pequena edição no arquivo `runtimeconfig` para configurar o `targetframeworkversion` como `"net8.0"`

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do IBM MQ .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto IBM MQ 9.4.0

Stabilized LTS A biblioteca `amqmdnet.dll` ainda é fornecida, mas esta biblioteca está estabilizada; isto é, nenhum novo recurso será introduzido nela. Para qualquer um dos recursos mais recentes deve-se migrar para a biblioteca `amqmdnetstd.dll`. No entanto, é possível continuar a usar a biblioteca `amqmdnet.dll` nas liberações do IBM MQ 9.1 Long Term Support ou do Continuous Delivery.

Sobre esta tarefa

É possível construir seus aplicativos do IBM MQ .NET em uma máquina na qual o cliente IBM MQ completo está instalado e posteriormente empacotar o conjunto IBM MQ .NET , ou seja, `amqmdnetstd.dll`, juntamente com seu aplicativo e implementá-lo em sistemas de produção.

Os aplicativos que você construir e implementar podem ser os aplicativos .NET tradicionais, Serviços ou aplicativos Microsoft Azure Web / Worker

Em tais implementações, o cliente IBM MQ .NET suporta apenas o modo gerenciado de conectividade com um gerenciador de filas. As ligações de servidor e a conectividade do modo de cliente não gerenciado não estão disponíveis, visto que esses dois modos requerem uma instalação completa do cliente IBM MQ. Qualquer tentativa de usar estes outros dois modos resulta em uma exceção do aplicativo.

Procedimento

Referenciando o IBM MQ .NET conjunto do cliente nos aplicativos

- Faça referência ao conjunto `amqmdnetstd.dll` em seu aplicativo da mesma forma que você fez para versões anteriores.

  Configure a propriedade **CopyLocal** do conjunto `amqmdnetstd.dll` como `True` para assegurar que o conjunto `amqmdnetstd.dll` seja copiado para o diretório `bin` do aplicativo. A configuração dessa propriedade também ajuda a ferramenta do pacote de aplicativos para compactar os arquivos binários necessários para implementação em sistemas de produção, bem como ambientes Microsoft da nuvem Azure PaaS.

Incluindo o suporte de transação global

- Certifique-se de que seu aplicativo implemente o aplicativo WMQDotnetXAMonitor na máquina juntamente com o próprio aplicativo.

  Se um aplicativo usar o recurso de transação global gerenciado do IBM MQ .NET, ele deverá também implementar o WMQDotnetXAMonitor na máquina juntamente com o próprio aplicativo. Este utilitário é necessário para recuperar quaisquer transações em dúvida.

Iniciando e parando o rastreio

- Somente para IBM MQ classes for .NET Framework , para iniciar e parar o rastreio usando o arquivo de configuração de aplicativo e um arquivo de configuração de rastreio específico do IBM MQ , consulte Rastreando um IBM MQ classes para o cliente .NET Framework usando um arquivo de configuração de aplicativo

  Deve-se usar o arquivo de configuração de aplicativo e um arquivo de configuração de rastreio específico do IBM MQ porque, como não há uma instalação completa do cliente do IBM MQ, as ferramentas padrão que são usadas para iniciar e parar o rastreio, **strmqtrc** e **endmqtrc**, não estão disponíveis.

  Notas:

  - Essa maneira de gerar rastreio se aplica ao cliente gerenciado redistribuível do .NET , bem como ao cliente independente do .NET . Consulte .NET runtime do aplicativo- Windows apenas.
  - O arquivo de configuração de aplicativo não é suportado nas bibliotecas IBM MQ classes for .NET (.NET Standard e .NET 6)... Para ativar o rastreio para IBM MQ classes for .NET (bibliotecas.NET Standard e .NET 6), use a variável de ambiente **MQDOTNET_TRACE_ON** . Consulte Rastreio IBM MQ .NET de aplicativos usando variáveis de ambiente

- V 9.4.0

  Inicie e pare o rastreio usando o arquivo `mqclient.ini` e configurando as propriedades apropriadas da sub-rotina de Rastreio

  Consulte Rastreio IBM MQ .NET de aplicativos com mqclient.ini

  Em IBM MQ 9.4.0, é possível configurar o rastreio usando o arquivo `mqclient.ini` e configurar as propriedades apropriadas da sub-rotina Trace. Também é possível ativar e desativar o rastreio dinamicamente com o arquivo `mqclient.ini` ..

Ativando o redirecionamento de ligação no arquivo de configuração de aplicativo

- Para ativar a referência de ligação de tempo de compilação do conjunto IBM MQ .NET para uma versão posterior do mesmo, inclua a propriedade <dependentAssembly> para o arquivo de configuração do aplicativo.

  O fragmento de exemplo a seguir no arquivo `app.config` redireciona um aplicativo que foi compilado usando a versão IBM MQ 8.0.0 Fix Pack 2 (`8.0.0.2`) do conjunto IBM MQ .NET , mas posteriormente um fix pack, IBM MQ 8.0.0 Fix Pack 3, foi então aplicado que atualizou o conjunto IBM MQ.NET para `8.0.0.3`.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
```

```
                          publicKeyToken="dd3cb1c9aae9ec97"
                          culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

Conceitos relacionados

“Instalando IBM MQ classes for .NET” na página 566
IBM MQ classes for .NET, incluindo amostras, são instalados com IBM MQ em Windows e Linux

Clientes redistribuíveis

Aplicativo de tempo de execução do.NET - Windows somente

Tarefas relacionadas

“Usando o aplicativo WMQDotnetXAMonitor” na página 585
O cliente IBM MQ .NET fornece um aplicativo XA Monitor, WmqDotnetXAMonitor, que você pode usar para recuperar qualquer transação distribuída incompleta. O aplicativo WmqDotnetXAMonitor estabelece uma conexão com o gerenciador de filas onde as transações estão em dúvida e, em seguida, resolve a transação com base nos parâmetros que você configura.

Rastreando aplicativos .NET do IBM MQ

Propriedade OutboundSNI

É possível configurar a propriedade **OutboundSNI** em um aplicativo usando uma propriedade ou uma variável de ambiente.

A partir de IBM MQ 9.3.0, é possível configurar MQC.OUTBOUND_SNI_PROPERTY no aplicativo, usando uma tabela de dispersão ao usar a classe `MQQueueManager` para se conectar ao gerenciador de filas.

O MQC.OUTBOUND_SNI_PROPERTY leva os valores a seguir:

- MQC.OUTBOUND_SNI_CHANNEL, que mapeia para "CHANNEL"
- MQC.OUTBOUND_SNI_HOSTNAME, que mapeia para "HOSTNAME"
- MQC.OUTBOUND_SNI_ASTERISK, que mapeia para "`*`"

Adicionalmente, é possível configurar a propriedade **OutboundSNI** usando a variável de ambiente MQOUTBOUND_SNI, que leva os valores a seguir:

- CHANNEL
- HOSTNAME
- *

e configure o valor **OutboundSNI** no arquivo App`.config` , como qualquer outra propriedade `mqclient.ini` .

Nota: A propriedade assume o padrão MQC.OUTBOUND_SNI_CHANNEL se nenhum valor específico for definido.

A ordem de precedência para configurar a propriedade **OutboundSNI** no nó gerenciado é:

1. Propriedade do nível do aplicativo
2. Variável de Ambiente

Para a propriedade **OutboundSNI** em nó não gerenciado, somente `mqclient.ini` é suportado.

As propriedades configuradas no arquivo App`.config` são aplicáveis apenas para aplicativos .NET Framework

Se você fornecer um valor que não seja válido no nível do aplicativo ou no arquivo App`.config`, o código de retorno MQRC_OUTBOUND_SNI_NOT_VALID é emitido.

Se você configurar uma variável de ambiente que não seja válida ou fornecer um valor que não seja válido no arquivo `mqclient.ini` , o valor padrão de `CHANNEL` será usado.

`OutboundSNI` e múltiplos certificados

O IBM MQ usa o cabeçalho SNI para fornecer a funcionalidade de vários certificados Se um aplicativo estiver se conectando a um canal do IBM MQ configurado para usar um certificado diferente por meio do campo CERTLABL, o aplicativo deverá se conectar a uma configuração **`OutboundSNI`** de CHANNEL.

Se um aplicativo com uma configuração **`OutboundSNI`** diferente de CHANNEL se conectar a um canal com um rótulo certificado configurado, o aplicativo será rejeitado com um MQRC_SSL_INITIALIZATION_ERROR e uma mensagem AMQ9673 será impressa nos logs de erro do gerenciador de filas.

Para obter mais informações sobre como o IBM MQ fornece a funcionalidade de diversos certificados, consulte Como IBM MQ fornece a capacidade de diversos certificados

Desenvolvendo aplicativos do XMS .NET

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) fornece uma interface de programação de aplicativos (API) chamada XMS que possui o mesmo conjunto de interfaces que a API Java Message Service (JMS). O IBM MQ Message Service Client (XMS) for .NET contém uma implementação totalmente gerenciada do XMS, que pode ser usada por qualquer idioma compatível com o .NET.

Antes de começar

Deprecated V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, em IBM MQ classes for XMS .NET, os métodos WriteObject(), ReadObject(), CreateObjectMessage () e as classes ObjectMessage e XmsObjectMessageImpl usados para serialização e desserialização de dados foram descontinuados.

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do XMS .NET construída usando.NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

Sobre esta tarefa

O XMS suporta:

- Sistema de mensagens ponto a ponto
- Sistema de Mensagens de Publicação/Assinatura
- Entrega de Mensagem Síncrona
- Entrega de Mensagem Assíncrona

Um aplicativo XMS pode trocar mensagens com os seguintes tipos de aplicativo:

- Um aplicativo do XMS
- Um aplicativo do IBM MQ classes for JMS
- Um aplicativo IBM MQ nativo
- Um aplicativo JMS que está usando o provedor de sistemas de mensagens padrão IBM MQ

Um aplicativo XMS pode se conectar a, e usar os recursos do, qualquer um dos seguintes servidores de sistema de mensagens:

Gerenciador de filas da IBM MQ
O aplicativo pode se conectar no modo de ligações ou de cliente.

WebSphere Application Server service integration bus
O aplicativo pode usar uma conexão TCP/IP direta, ou pode usar HTTP sobre TCP/IP.

IBM Integration Bus
As mensagens são transportadas entre o aplicativo e o broker usando WebSphere MQ Real-Time Transport. As mensagens podem ser entregues para o aplicativo usando o WebSphere MQ Multicast Transport.

Ao conectar-se com um gerenciador de filas do IBM MQ, um aplicativo XMS pode usar o WebSphere MQ Enterprise Transport para comunicar-se com o IBM Integration Bus. Alternativamente, um aplicativo XMS pode publicar e assinar conectando-se a IBM MQ.

V 9.4.0 IBM MQ 9.4.0 fornece uma biblioteca do cliente XMS .NET construída com relação a .NET 6 como a estrutura de destino. Para obter mais informações, consulte "Instalando IBM MQ classes for XMS .NET" na página 630.

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for XMS .NET. Para obter mais informações, consulte "Instalando IBM MQ classes for XMS .NET" na página 630.

Os aplicativos gerenciados do XMS .NET podem balancear automaticamente as conexões entre os gerenciadores de filas em cluster. As bibliotecas IBM MQ classes for XMS .NET e IBM MQ classes for XMS .NET Framework são suportadas. Para obter mais informações, consulte Sobre clusters uniformes e Balanceamento automático de aplicativo.

Para obter mais informações sobre as diferenças entre IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET, consulte "Instalando IBM MQ classes for XMS .NET" na página 630.

Tarefas relacionadas

Entrando em Contato com o IBM Support

Resolução de problemas do XMS .NETProblemas

Estilos de sistema de mensagens suportados por XMS

XMS suporta os estilos ponto a ponto e publicar / assinar do sistema de mensagens.

Estilos de sistema de mensagens também são chamados de domínios de mensagens.

Sistema de mensagens ponto a ponto

Uma forma comum de sistema de mensagens ponto a ponto usa o enfileiramento. No caso mais simples, um aplicativo envia uma mensagem para outro aplicativo identificando, implicitamente ou explicitamente, uma fila de destino. O sistema de mensagens subjacente e o sistema de enfileiramento recebem a mensagem do aplicativo de envio e roteia a mensagem para sua fila de destino. O aplicativo de recebimento pode então recuperar a mensagem a partir da fila.

Se o sistema de mensagens e o sistema de enfileiramento subjacentes contiverem o IBM Integration Bus, o IBM Integration Bus poderá replicar uma mensagem e rotear cópias da mensagem para filas diferentes. Como resultado, mais de um aplicativo pode receber a mensagem. IBM Integration Bus também pode transformar uma mensagem e incluir dados a ele.

Uma característica chave do sistema de mensagens ponto-a-ponto é que um aplicativo coloca uma mensagem em uma fila local quando ela envia uma mensagem. O sistema de mensagens subjacente e o sistema de enfileiramento determinam para qual fila de destino a mensagem é enviada. O aplicativo de recebimento recupera a mensagem da fila de destino.

Sistema de Mensagens de Publicação/Assinatura

No sistema de mensagens de publicação / assinatura, há dois tipos de aplicativo: publicador e assinante.

Um *publicador* fornece informações sobre a forma de mensagens de publicação. Quando um publicador publica uma mensagem, ele especifica um tópico, que identifica o assunto das informações dentro da mensagem.

Um *assinante* é um consumidor das informações que são publicadas. Um assinante especifica os tópicos em que está interessado pela criação de assinaturas.

O sistema de publicação / assinatura recebe publicações de publicadores e assinaturas de assinantes. Ele roteia publicações para assinantes. Um assinante recebe publicações sobre apenas os tópicos para os quais ele subscreveu.

Uma característica chave do sistema de mensagens de publicação / assinatura é que um publicador identifica um tópico quando ele publica uma mensagem. Ele não identifica os assinantes. Se uma mensagem for publicada em um tópico para o qual não há assinantes, nenhum aplicativo receberá a mensagem.

Um aplicativo pode ser um publicador e um assinante.

O modelo de objeto XMS

A API XMS é uma interface orientada a objetos. O modelo de objeto XMS é baseado no modelo de objeto JMS 1.1 .

Principais classes XMS

As classes principais XMS ou tipos de objeto são os seguintes:

ConnectionFactory
Um objeto `ConnectionFactory` é encapsulado um conjunto de parâmetros para uma conexão. Um aplicativo usa um `ConnectionFactory` para criar uma conexão. Um aplicativo pode fornecer os parâmetros no tempo de execução e criar um objeto `ConnectionFactory` . Como alternativa, os parâmetros de conexão podem ser armazenados em um repositório de objetos administrados. Um aplicativo pode recuperar um objeto a partir do repositório e criar um objeto `ConnectionFactory` a partir dele.

Conexão
Um objeto `Connection` encapsula uma conexão ativa de um aplicativo para um servidor de sistema de mensagens. Um aplicativo usa uma conexão para criar sessões.

Destino
Um aplicativo envia mensagens ou recebe mensagens usando um objeto `Destination`. No domínio de publicação / assinatura, um objeto `Destination` encapsula um tópico e, no domínio ponto-a-ponto, um objeto `Destination` encapsula uma fila. Um aplicativo pode fornecer os parâmetros para criar um objeto `Destination` no tempo de execução. Como alternativa, ele pode criar um objeto `Destination` a partir de uma definição de objeto que é armazenada no repositório de objetos administrados.

Sessão
Um objeto `Session` é um único contexto encadeado para enviar e receber mensagens. Um aplicativo usa um objeto `Session` para criar objetos `Message`, `MessageProducere` `MessageConsumer` .

Mensagem
Um objeto `Message` contém o objeto `Message` que um aplicativo envia usando um objeto `MessageProducer` ou recebe usando um objeto `MessageConsumer`.

MessageProducer
Um objeto `MessageProducer` é usado por um aplicativo para enviar mensagens para um destino.

MessageConsumer
Um objeto `MessageConsumer` é usado por um aplicativo para receber mensagens enviadas para um destino.

Os objetos do XMS e seus relacionamentos

Figura 52 na página 628 mostra os tipos principais do objeto XMS : ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message e Destination. Um aplicativo usa um connection factory para criar uma conexão e usa uma conexão para criar sessões. O aplicativo pode então usar uma sessão para criar mensagens, produtores de mensagens e consumidores de mensagens. O aplicativo usa um produtor de mensagem para enviar mensagens para um destino e usa um consumidor de mensagens para receber mensagens enviadas para um destino.

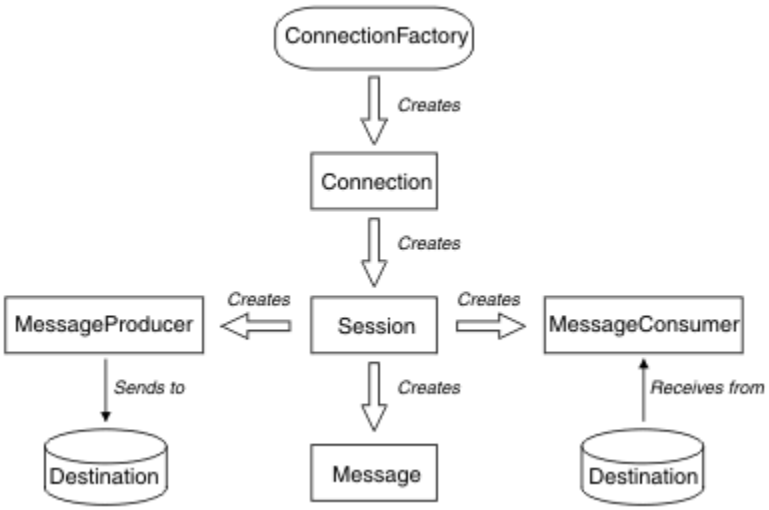


Figura 52. Os objetos do XMS e seus relacionamentos

Em XMS .NET, as classes XMS são definidas como um conjunto de interfaces .NET . Quando você está codificando aplicativos XMS .NET , é necessário apenas as interfaces declaradas.

O modelo de objeto XMS é baseado nas interfaces independentes de domínio que são descritas em Java Message Service Especificação, Versão 1.1. Classes específicas de domínio, como `Topic`, `TopicPublishere` `TopicSubscriber`, não são fornecidas.

Atributos e propriedades de objetos XMS

Um objeto XMS pode ter atributos e propriedades, que são características do objeto, que são implementados de maneiras diferentes:

Atributos
Uma característica de objeto que está sempre presente e ocupa o armazenamento, mesmo que o atributo não tenha um valor. Nesse aspecto, um atributo é semelhante a um campo em uma estrutura de dados de comprimento fixo. Um recurso de diferenciação de atributos é que cada atributo possui seus próprios métodos para configurar e obter seu valor.

Propriedades
Uma propriedade de um objeto está presente e ocupa o armazenamento apenas depois que seu valor for configurado. Uma propriedade não pode ser excluída ou sua memória recuperada após seu valor ser configurado. É possível alterar seu valor. XMS fornece um conjunto de métodos genéricos para a configuração e obtenção de valores de propriedade.

Objetos Administrados

Usando objetos administrados, é possível administrar as configurações de conexão usadas pelos aplicativos clientes a serem administradas a partir de um repositório central. Um aplicativo recupera definições de objeto do repositório central e as utiliza para criar objetos `ConnectionFactory` e `Destination` . Usando objetos administrados, é possível desacoplar aplicativos dos recursos usados no tempo de execução.

Por exemplo, os aplicativos XMS podem ser gravados e testados com objetos administrados que fazem referência a um conjunto de conexões e destinos em um ambiente de teste. Quando os aplicativos são implementados, os objetos administrados podem ser alterados para configurar os aplicativos para se referir a conexões e destinos no ambiente de produção.

XMS suporta dois tipos de objeto administrado:

- Um objeto `ConnectionFactory` , que é usado pelos aplicativos para fazer a conexão inicial com o servidor.

- Um objeto `Destination` , que é usado pelos aplicativos para especificar o destino para mensagens que estão sendo enviadas e a origem de mensagens que estão sendo recebidas. Um destino é um tópico ou uma fila no servidor para o qual um aplicativo se conecta.

A ferramenta de administração **JMSAdmin** é fornecida com IBM MQ. Ela é usada para criar e gerenciar objetos administrados em um armazenador central de objetos administrados.

Os objetos administrados no repositório podem ser usados pelos aplicativos IBM MQ classes for JMS e XMS . Aplicativos XMS podem usar os objetos `ConnectionFactory` e `Destination` para se conectar a um IBM MQ gerenciador de filas. Um administrador pode alterar as definições de objeto mantidas no repositório sem afetar o código do aplicativo.

O diagrama a seguir mostra como um aplicativo XMS geralmente usa objetos administrados. O lado esquerdo do diagrama mostra um repositório contendo as definições de objeto ConnectionFactory e Destination que são administradas usando um console de administração. O lado direito do diagrama mostra um aplicativo XMS que consulta as definições de objeto no repositório e, em seguida, usa essas definições de objeto ao se conectar a um servidor de sistema de mensagens.

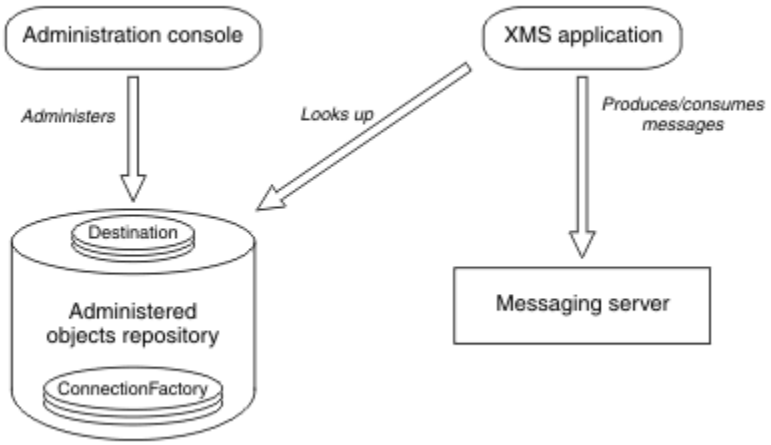


Figura 53. Uso Típico de Objetos Administrados por um Aplicativo XMS

O modelo de mensagem XMS

O modelo de mensagem XMS é o mesmo que o modelo de mensagem IBM MQ classes for JMS .

Em particular, XMS implementa os campos de cabeçalho da mesma mensagem e as propriedades de mensagem que o IBM MQ classes for JMS implementa:

- Campos de cabeçalho JMS . Esses campos possuem nomes que se iniciam com o prefixo JMS.
- Propriedades definidas pelo JMS. Esses campos possuem propriedades cujos nomes se iniciam com o prefixo JMSX.
- Propriedades definidas pelo IBM. Esses campos possuem propriedades cujos nomes começam com o prefixo JMS_IBM_.

Como resultado, os aplicativos XMS podem trocar mensagens com aplicativos IBM MQ classes for JMS . Em cada mensagem, alguns dos campos de cabeçalho e propriedades são configurados pelo aplicativo e outros são configurados por XMS ou IBM MQ classes for JMS. Alguns dos campos configurados por XMS ou IBM MQ classes for JMS são configurados quando a mensagem é enviada e outros quando ela é recebida. Os campos de cabeçalho e propriedades são propagados com uma mensagem por meio de um servidor de sistema de mensagens quando apropriado. Elas são disponibilizadas para qualquer aplicativo que receba a mensagem.

Conceitos relacionados

IBM MQ classes for JMS

Windows Linux Instalando IBM MQ classes for XMS .NET

O IBM MQ classes for XMS .NET, incluindo amostras, é instalado com IBM MQ em Windows e Linux

Instalação

V 9.4.0 IBM MQ 9.4.0 fornece uma biblioteca do cliente do XMS .NET construída em .NET 6 como a estrutura de destino. No IBM MQ 9.4.0, Microsoft .NET 6.0 é a versão mínima necessária para executar aplicativos usando bibliotecas do IBM MQ que são construídas usando o .NET 6 como a estrutura de destino. A biblioteca do cliente do XMS .NET construída usando .NET 6 como a estrutura de destino está disponível em *MQ_INSTALLATION_PATH*`/bin` em Windows e em *MQ_INSTALLATION_PATH*`/lib64` em Linux

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for XMS .NET. Se você estiver usando um aplicativo .NET 6 , será possível executar esse aplicativo sem que qualquer recompilação seja necessária, fazendo uma pequena edição no arquivo `runtimeconfig` para configurar o `targetframeworkversion` como `"net8.0"`

Deprecated V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, em IBM MQ classes for XMS .NET, os métodos WriteObject(), ReadObject(), CreateObjectMessage () e as classes ObjectMessage e XmsObjectMessageImpl usados para serialização e desserialização de dados foram descontinuados.

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do XMS .NET construída usando.NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

Biblioteca do `amqmxmsstd.dll`

V 9.4.0 A partir do IBM MQ 9.4.0, a biblioteca `amqmxmsstd.dll` construída usando .NET 6 como a estrutura de destino está disponível nos locais a seguir:

- Windows Em Windows: *MQ_INSTALLATION_PATH*`\bin`. Os aplicativos de amostra estão instalados em *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`
- Linux Em Linux: *MQ_INSTALLATION_PATH*`\lib64`. As amostras do .NET estão no *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do XMS .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

Atenção: V 9.4.0 Removed V 9.4.0 De IBM MQ 9.4.0, XMS .NET bibliotecas clientes construídas usando .NET Standard 2.0 como a estrutura de destino são removidas. Essas bibliotecas foram descontinuadas em IBM MQ 9.3.1.

Stabilized Todas as bibliotecas `IBM.XMS.*` ainda são fornecidas, mas essas bibliotecas estão estabilizadas; ou seja, nenhum recurso novo será introduzido nelas. Para qualquer um dos recursos mais recentes, deve-se migrar para a biblioteca `amqmxmsstd.dll`. No entanto, é possível continuar usando as bibliotecas existentes nas liberações do IBM MQ 9.1 Long Term Support ou Continuous Delivery.

V 9.4.0 V 9.4.0 A seguir estão dois cenários que podem ser encontrados após a remoção das bibliotecas `netstandard2.0` :

- Se você estiver usando um aplicativo IBM MQ classes for XMS .NET Framework construído usando as bibliotecas `netstandard2.0` , como `amqmdnetstd.dll`, será necessário reconstruir seu aplicativo com as bibliotecas Microsoft.NET Framework 4.7.2 , como `amqmdnet.dll`, para que seu aplicativo seja executado com êxito. Se você não reconstruir seu aplicativo, poderá obter um System.IO.Unexceptionable não excepcional:

```
Exceção capturada: System.IO.FileLoadException: Não foi possível carregar o arquivo ou conjunto
'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' ou uma de suas
dependências. A definição de manifest do conjunto localizado não corresponde à referência de
conjunto (Exceção de HRESULT: 0x80131040)
Nome do arquivo: 'amqmdnetstd, Version=9.3.5.0, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
   em SimplePut.SimplePut.PutMessages()
   em SimplePut.SimplePut.Principal (String [] args) em C:\SampleCode\Program.cs:line 132
```

- Se você estiver usando um aplicativo .NET 6 que é construído usando bibliotecas `netstandard2.0`, será necessário apenas substituir essas bibliotecas pelas mesmas bibliotecas .NET 6 na pasta `bin` do diretório de tempo de execução do aplicativo. Nenhuma reconstrução é necessária.

  Nota: A biblioteca .NET 6 de substituição deve sempre ser do mesmo nível ou superior que a biblioteca `netstandard2.0` substituída

Os IBM MQ classes for XMS .NET Standard estão disponíveis para download no repositório do NuGet . O pacote do NuGet contém as bibliotecas `amqmxmsstd.dll` e `amqmdnetstd.dll`. O `amqmxmsstd.dll` é dependente do `amqmdnetstd.dll` e, ao empacotar o aplicativo XMS .NET Core, ambos `amqmxmsstd.dll` e `amqmdnetstd.dll` devem ser empacotados juntamente com o aplicativo XMS .NET Core. Para obter mais informações, consulte “Fazendo download do IBM MQ classes for XMS .NET do repositório do NuGet” na página 633.

Comando dspmqver

É possível usar o comando **dspmqver** para exibir as informações de versão e de construção para o componente .NET Core.

Comparação entre as bibliotecas IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET Bibliotecas .NET 6 e .NET 6)

A tabela a seguir lista os recursos para IBM MQ classes for XMS .NET Framework em comparação com os recursos para IBM MQ classes for XMS .NET e .NET 6).

Tabela 80. Diferenças entre IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET

Recurso	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET**
Nomes de classes (APIs)	Todas as classes permanecem as mesmas em cada rede.	Todas as classes permanecem as mesmas em cada rede.
Sistema Operacional	Windows	Windows Contêineres Docker Linux macOS
Arquivo `app.config` (arquivo de configuração para ativar o Rastreio no cliente redistribuível)	O arquivo `app.config` é usado para ativar o rastreio para o pacote redistribuível	`app.config` não será suportada. Use as variáveis de ambiente

Tabela 80. Diferenças entre IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET (continuação)

Recurso	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
Trace	Para rastrear o cliente XMS .NET , é possível usar as variáveis de ambiente existentes, como a variável de ambiente **XMS_TRACE_ON** usada para ativar o rastreio. Para obter mais informações, veja Configurando o rastreio do XMS .NET usando variáveis de ambiente do XMS. Para clientes redistribuíveis, o arquivo `app.config` pode ser usado para ativar o rastreamento V 9.4.0 Em IBM MQ 9.4.0, é possível ativar e desativar o rastreio usando o arquivo `mqclient.ini` e configurando as propriedades apropriadas da sub-rotina Trace. Também é possível ativar e desativar o rastreio dinamicamente com o arquivo `mqclient.ini` .. Para obter mais informações, consulte Rastreio IBM MQ .NET de aplicativos com mqclient.ini	Para rastrear o cliente XMS .NET , é possível usar as variáveis de ambiente existentes, como a variável de ambiente **XMS_TRACE_ON** usada para ativar o rastreio. Para obter mais informações, veja Configurando o rastreio do XMS .NET usando variáveis de ambiente do XMS. V 9.4.0 Em IBM MQ 9.4.0, é possível ativar e desativar o rastreio usando o arquivo `mqclient.ini` e configurando as propriedades apropriadas da sub-rotina Trace. Também é possível ativar e desativar o rastreio dinamicamente com o arquivo `mqclient.ini` .. Para obter mais informações, consulte Rastreio IBM MQ .NET de aplicativos com mqclient.ini
Modos de transporte	Gerenciado, Não gerenciado e Ligações	Gerenciado
TLS	O keystore Windows é usado para armazenar os certificados.	No Windows, o keystore deve ser usado para armazenar os certificados. Os valores permitidos são *USER ou *SYSTEM. Com base na entrada, o cliente do IBM MQ .NET consulta o keystore do Windows do usuário atual ou todo o sistema. No Linux, recomenda-se usar a classe X509Store para instalar certificados e o .NET Core instala certificados no local a seguir: "`.dotnet/corefx/cryptography/x509stores`".
CCDT	Suportado	Suportado, e as configurações do caminho CCDT são iguais às para as classes do .NET Framework.
Reconexão automática do cliente	Suportado	Suportado
Transações distribuídas	Suportado	Não Suportado

Tabela 80. Diferenças entre IBM MQ classes for XMS .NET Framework e IBM MQ classes for XMS .NET (continuação)

Recurso	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
Instalação de bibliotecas de vínculo dinâmico (dlls) no cache de montagem global (GAC)	As Dlls são instaladas no GAC como parte da instalação do IBM MQ.	As Dlls não são instaladas no GAC como parte da instalação do IBM MQ.
Suporte para os tipos de conexão WMQ, WPM e RTT	Suporta os tipos de conexão WMQ, WPM e RTT	Suporte apenas para WMQ
Objetos Administrados por JNDI	Suporta LDAP e FileSystem	Suporta FileSystem apenas

Em IBM MQ 9.3.0, para executar IBM MQ classes for XMS .NET Framework , deve-se instalar o Microsoft.NET Framework V4.7.2 ou mais recente.

Tarefas relacionadas

"Usando os aplicativos de amostra XMS" na página 640
Os aplicativos de amostra XMS.NET fornecem uma visão geral dos recursos comuns de cada API. É possível usá-los para verificar a sua instalação e o servidor de sistema de mensagens configurado e para ajudar a construir os seus próprios aplicativos.

Windows Linux Fazendo download do IBM MQ classes for XMS .NET do repositório do NuGet

Os IBM MQ classes for XMS .NET estão disponíveis para download no repositório do NuGet para que possam ser facilmente consumidos por desenvolvedores do .NET.

Sobre esta tarefa

NuGet é o gerenciador de pacotes para plataformas de desenvolvimento da Microsoft, incluindo o .NET. As ferramentas do cliente do NuGet fornecem a capacidade de produzir e consumir pacotes. Um pacote NuGet é um único arquivo compactado com a extensão `.nupkg` que contém código compilado (DLLs), outros arquivos relacionados a aquele código e um manifesto descritivo que inclui informações como o número da versão do pacote.

É possível fazer download do pacote `IBMXMSDotnetClient` NuGet , que contém a biblioteca `amqmdnetstd.dll` e a biblioteca `amqmxmsstd.dll` , a partir da Galeria NuGet , que é o repositório do pacote central usado por todos os autores e consumidores do pacote.

Nota: V 9.4.0 V 9.4.0 No IBM MQ 9.4.0, o pacote NuGet contém bibliotecas construídas usando .NET 6 como a estrutura de destino.

Removed A biblioteca do cliente XMS .NET construída usando .NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0..

V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, IBM MQ suporta .NET 8 aplicativos usando IBM MQ classes for XMS .NET. Se você estiver usando um aplicativo .NET 6 , será possível executar esse aplicativo sem que qualquer recompilação seja necessária, fazendo uma pequena edição no arquivo `runtimeconfig` para configurar o `targetframeworkversion` como `"net8.0"`

Existem três maneiras de baixar o pacote `IBMXMSDotnetClient`:

- Usando o Microsoft Visual Studio. O NuGet é distribuído como uma extensão do Microsoft Visual Studio. No Microsoft Visual Studio 2012, o NuGet é pré-instalado por padrão.

- Na linha de comandos, usando o NuGet Package Manager ou a CLI do .NET.
- Usando um navegador da web.

Quanto ao pacote redistribuível, você ativa o rastreio usando a variável de ambiente **XMS_TRACE_ON**.

Procedimento

- Para baixar o pacote `IBMXMSDotnetClient` usando a IU do Gerente de Pacote em Microsoft Visual Studio, conclua as seguintes etapas:
  a) Clique com o botão direito no projeto do .NET e, em seguida, clique em **Gerenciar pacotes do Nuget**.
  b) Clique na guia **Procurar** e procure por `"IBMXMSDotnetClient"`.
  c) Selecione o pacote e clique em **Instalar**.

  Durante a instalação, o Package Manager fornece informações de progresso na forma de instruções do console.
- Para baixar o pacote `IBMXMSDotnetClient` a partir da linha de comando, escolha uma das seguintes opções:
  - usando o NuGet Package Manager, insira o comando a seguir:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    Durante a instalação, o Package Manager fornece informações de progresso na forma de instruções do console. É possível redirecionar a saída para um arquivo de log.
  - usando a CLI do .NET, insira o comando a seguir:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- Usando um navegador da web, baixe o pacote `IBMXMSDotnetClient` de https://www.nuget.org/packages/IBMXMSDotnetClient.

Conceitos relacionados

“Instalando IBM MQ classes for .NET” na página 566
IBM MQ classes for .NET, incluindo amostras, são instalados com IBM MQ em Windows e Linux

Informações sobre licença do IBM MQ Client for .NET

Tarefas relacionadas

“Fazendo download de IBM MQ classes for .NET a partir do repositório NuGet” na página 571
O IBM MQ classes for .NET está disponível para download a partir do repositório do NuGet , para que eles possam ser facilmente consumidos pelos Desenvolvedores do .NET

Configurando o Ambiente do Servidor de Mensagens

Os tópicos nesta seção descrevem como configurar o ambiente do servidor de sistema de mensagens para permitir que os aplicativos XMS se conectem a um servidor.

Sobre esta tarefa

Para aplicativos que se conectam a um gerenciador de filas IBM MQ , o cliente IBM MQ (ou o gerenciador de filas para o modo de ligações) é necessário.

Atualmente, não há pré-requisitos para aplicativos que usam uma conexão em tempo real com um broker.

Você deve configurar o ambiente do servidor de sistema de mensagens antes de executar quaisquer aplicativos XMS , incluindo os aplicativos de amostra fornecidos com XMS.

Esta seção contém os seguintes tópicos:

- “Configurando o gerenciador de filas e o broker para um aplicativo que se conecta a um gerenciador de filas IBM MQ” na página 637

- “Instalando IBM MQ classes for XMS .NET” na página 630
- “Configurando um broker para um aplicativo que usa uma conexão em tempo real com um broker” na página 638
- “Configurando o barramento de integração de serviços para um aplicativo que se conecta ao WebSphere Application Server” na página 639

Listeners de mensagens no XMS .NET

Um listener de mensagem é usado para receber mensagens assincronamente. Ao contrário da chamada `MessageConsumer.receive()` , o listener de mensagem não bloqueia o encadeamento de chamada, em vez disso, ele entrega mensagens para um método de retorno de chamada especificado pelo aplicativo, geralmente o método onMessage

A entrega de mensagem é iniciada quando o método `Connection.Start()` é chamado. A entrega de mensagem pode ser interrompida a qualquer momento usando os métodos `Connection.Stop()` e `Connection.Start()` , respectivamente.

Quando o método `Connection.Start()` é chamado após a configuração de um listener de mensagem para pelo menos um consumidor em uma sessão, essa sessão se torna uma sessão assíncrona. Quando uma sessão se torna assíncrona, não é possível chamar nenhum método síncrono do XMS .NET . Por exemplo, `MessageProducer.Send()`. Fazer isso resulta em uma exceção com o código de razão IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500).

Chamadas síncronas em uma sessão assíncrona

`Session.Close` é a única chamada síncrona permitida em uma sessão assíncrona. Os aplicativos também podem fazer chamadas síncronas (exceto `Session.Close`) usando o método de retorno de chamada do listener de mensagens, ou seja, o método onMessage

Além dessas duas opções, você deve parar a conexão usando o método `Connection.Stop()` para um aplicativo fazer qualquer chamada síncrona. Após as chamadas serem feitas, você deve continuar a conexão novamente usando o método `Connection.Start()` . que reinicia a entrega de mensagens..

Quantos consumidores de mensagens assíncrona uma sessão pode ter?.

Uma sessão pode ter vários consumidores de mensagens assíncronas Mas a qualquer momento uma mensagem é entregue a apenas um consumidor. O que isso praticamente significa é que, quando uma segunda mensagem chega enquanto o XMS .NET chamou o método `onMessage()` de um consumidor para entregar a primeira mensagem, a segunda mensagem não será entregue a um consumidor na sessão até que o método `onMessage()` seja retornado

A segunda mensagem é entregue a um consumidor na sessão somente após o método `onMessage()` retornar. Isso ocorre porque uma sessão gerencia a entrega de mensagens para os consumidores usando apenas um encadeamento Isso significa que apenas uma mensagem pode ser entregue por vez e o consumidor pode ser qualquer um.

Se um aplicativo requerer entrega de mensagem simultânea, ou seja, todos os consumidores devem receber mensagens ao mesmo tempo, então o aplicativo deverá criar várias sessões e cada um deve ter um consumidor de mensagens assíncronas

Os exemplos a seguir mostram esse recurso mais claramente.

No primeiro exemplo, há vários consumidores de mensagens assíncronas em uma sessão Uma sessão S possui três consumidores de mensagens assíncronas: AMC1, AMC2 e AMC3 que recebem mensagens de três destinos diferentes Q1, Q2 e Q3.

Como há apenas uma sessão S, há apenas encadeamento de entrega de mensagens para entregar mensagens aos consumidores AMC1, AMC2e AMC3. Quando a sessão está entregando mensagem para o AMC1, os outros dois consumidores AMC2 e AMC3 esperam, mesmo se houver mensagens no Q2 e Q3 prontas para entrega.

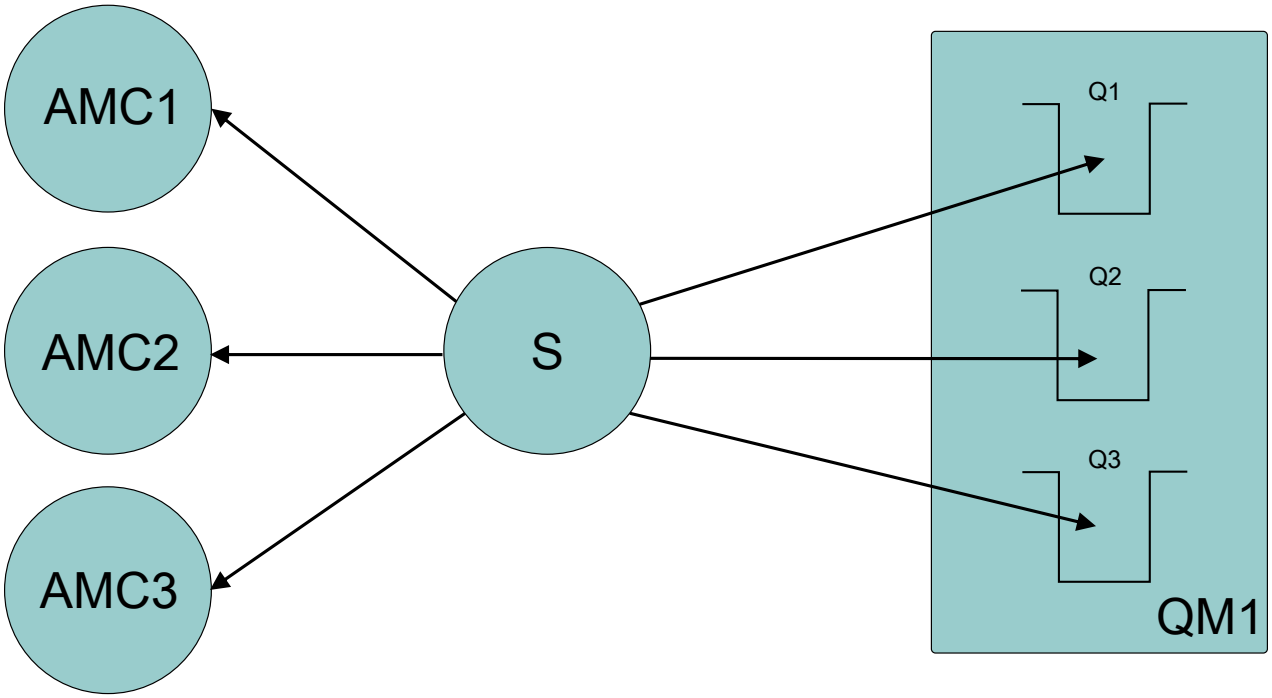


Figura 54. Uma sessão com três consumidores de mensagens assíncronas

No segundo caso, há várias sessões S1, S2e S3, cada uma tendo um consumidor de mensagens assíncronas `AMC1`, `AMC2`e `AMC3` , respectivamente, Como há um consumidor para cada sessão, as mensagens são entregues para os consumidores simultaneamente

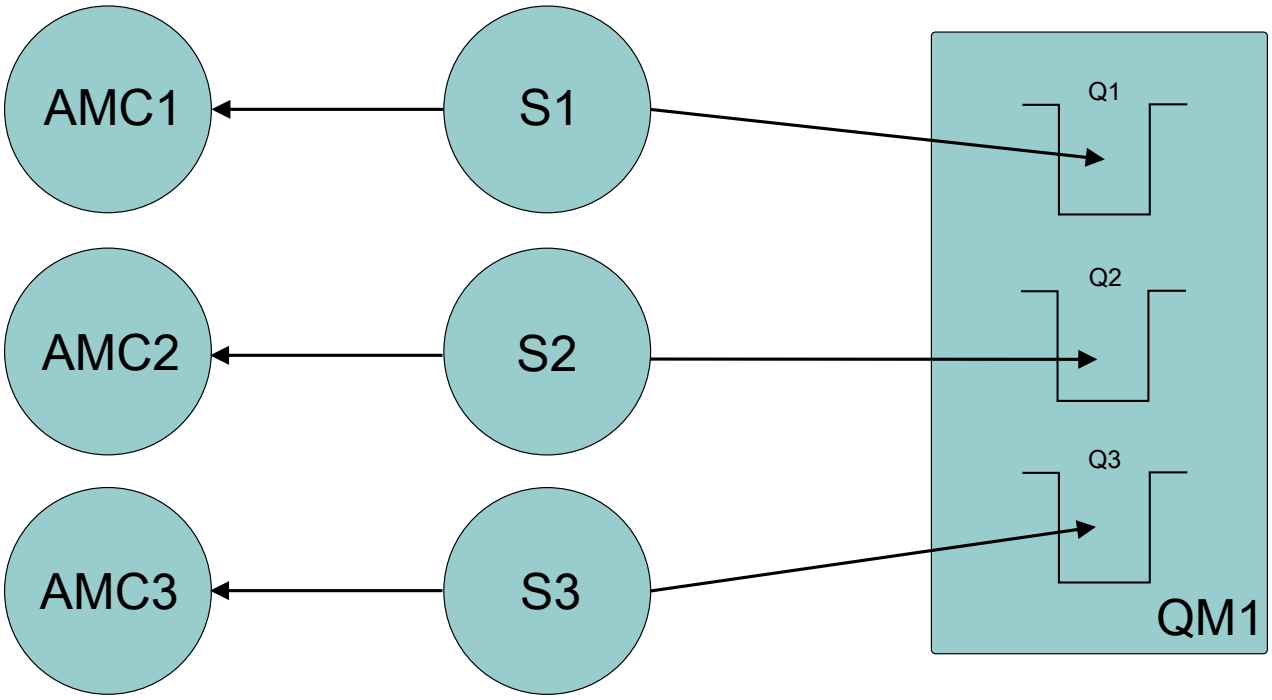


Figura 55. Várias sessões, cada uma com um consumidor de mensagens assíncronas

Isso mostra que, se você precisar de entrega de mensagem simultânea, precisará de várias sessões

Configurando o gerenciador de filas e o broker para um aplicativo que se conecta a um gerenciador de filas IBM MQ

Esta seção assume que você está usando IBM WebSphere MQ 7.0.1ou mais recente. Antes de poder executar um aplicativo que se conecta a um gerenciador de filas IBM MQ , você deve configurar o gerenciador de filas. Para um aplicativo de publicação / assinatura, alguma configuração adicional será necessária se você estiver usando a interface de publicação / assinatura enfileirada.

Antes de começar

O XMS funciona com o IBM Integration Bus ou o WebSphere Message Broker 6.1 ou mais recente

Antes de iniciar esta tarefa, execute as seguintes etapas:

- Certifique-se de que seu aplicativo tenha acesso a um gerenciador de filas que está em execução.
- Se o seu aplicativo for um aplicativo de publicação / assinatura e usar a interface de publicação / assinatura enfileirada, certifique-se de que o atributo **PSMODE** esteja configurado como ENABLED no gerenciador de filas.
- Certifique-se de que seu aplicativo usa um connection factory cujas propriedades são configuradas corretamente para se conectar ao gerenciador de filas. Se o seu aplicativo for um aplicativo de publicação / assinatura, certifique-se de que as propriedades apropriadas do connection factory estejam configuradas para o uso do broker. Para obter mais informações sobre as propriedades de um connection factory, consulte Propriedades do ConnectionFactory.

Sobre esta tarefa

É possível configurar o gerenciador de filas e o broker para executarem aplicativos XMS da mesma maneira que você configura o gerenciador de filas e a interface de publicação/assinatura enfileirada para executarem aplicativos IBM MQ JMS. As etapas a seguir resumem o que você precisa fazer.

Procedimento

1. No gerenciador de filas, crie as filas que seu aplicativo precisa.

   Para uma visão geral de como você cria filas, consulte Definindo filas.

   No caso de ser um aplicativo de publicação/assinatura e usar a interface de publicação/assinatura enfileirada que precisa de acesso às filas do sistema IBM MQ classes for JMS, aguarde até a Etapa 4a antes de criar as filas.
2. Conceda ao ID do usuário associado ao seu aplicativo a autoridade para se conectar ao gerenciador de filas e a autoridade apropriada para acessar as filas.

   Para uma visão geral sobre autorização, consulte Protegendo. Se o seu aplicativo se conecta ao gerenciador de filas no modo cliente, consulte também Clientes e servidores.
3. Se o seu aplicativo se conectar ao gerenciador de filas no modo cliente, certifique-se de que um canal de conexão do servidor esteja definido no gerenciador de filas e que um listener seja iniciado.

   Você não precisa executar esta etapa para cada aplicativo que se conecta ao gerenciador de filas. Uma definição de canal de conexão do servidor e um listener podem suportar todos os aplicativos que se conectam no modo cliente.
4. Se o seu aplicativo for um aplicativo de publicação / assinatura e usar a interface de publicação / assinatura enfileirada, execute as etapas a seguir.
   a) No gerenciador de filas, crie as filas do sistema IBM MQ classes for JMS executando o script de comandos MQSC que são fornecidos com IBM MQ. Certifique-se de que a ID do usuário associado com o IBM Integration Bus ou WebSphere Message Broker tenha autoridade para acessar as filas.

      Para obter informações sobre onde localizar o script e como executá-lo, consulte Usando IBM MQ classes for Java.

Execute esta etapa apenas uma vez para o gerenciador de filas. O mesmo conjunto de filas do sistema IBM MQ classes for JMS pode suportar todos os aplicativos XMS e IBM MQ classes for JMS que se conectam ao gerenciador de filas.

b) Conceda ao ID do usuário associado ao seu aplicativo a autoridade para acessar as filas do sistema IBM MQ classes for JMS .

Para obter informações sobre quais autoridades o ID do usuário precisa, consulte Usando IBM MQ classes for JMS.

c) Para um broker de IBM Integration Bus ou WebSphere Message Broker, crie e implemente um fluxo de mensagens para atender a fila na qual os aplicativos enviam mensagens que eles publicam.

O fluxo de mensagens básico inclui um nó de processamento de mensagens MQInput para ler as mensagens publicadas e um nó de processamento de mensagem de publicação para publicar as mensagens.

Para obter informações sobre como criar e implantar um fluxo de mensagens, consulte a documentação do produto IBM Integration Bus ou WebSphere Message Broker disponível a partir do Página da web da biblioteca de documentação do produto IBM Integration Bus.

Você não precisa executar esta etapa se um fluxo de mensagens adequado já estiver implementado no intermediário.

Resultados

Agora é possível iniciar seu aplicativo.

Configurando um broker para um aplicativo que usa uma conexão em tempo real com um broker

Antes de poder executar um aplicativo que usa uma conexão em tempo real com um broker, você deve configurar esse broker.

Antes de começar

Antes de iniciar esta tarefa, você executa as seguintes etapas:

- Certifique-se de que seu aplicativo tenha acesso a um broker que está em execução.
- Certifique-se de que seu aplicativo usa um connection factory cujas propriedades são configuradas apropriadamente para uma conexão em tempo real com um broker. Para obter mais informações sobre as propriedades de um connection factory, consulte Propriedades do ConnectionFactory.

Sobre esta tarefa

Você configura um broker para executar aplicativos XMS da mesma maneira que você configura um broker para executar aplicativos IBM MQ classes for JMS . As etapas a seguir resumem o que você precisa fazer:

Procedimento

1. Crie e implemente um fluxo de mensagens para ler mensagens a partir da porta TCP/IP na qual um broker está atendendo e publique as mensagens.

   Você pode fazer isso de uma das seguintes maneiras:

   - Crie um fluxo de mensagens que contenha um nó de processamento de mensagens **Real-timeOptimizedFlow** .
   - Crie um fluxo de mensagens que contenha um nó de processamento de mensagens **Real-timeInput** e um nó de processamento de mensagem de publicação.

Você deve configurar o nó **Real-timeOptimizedFlow** ou **Real-timeInput** para atender na porta usada para conexões em tempo real. Em XMS, o número da porta padrão para conexões em tempo real é 1506.

Você não precisa executar esta etapa se um fluxo de mensagens adequado já estiver implementado no intermediário.

2. Se precisar que as mensagens sejam entregues ao seu aplicativo usando o IBM MQ classes for JMS, configure o broker para ativar o multicast. Configure os tópicos que devem ser ativados multicast, especificando uma qualidade de serviço confiável para esses tópicos que requerem multicast confiável.
3. Se o seu aplicativo fornecer um ID de usuário e uma senha quando ele se conectar ao intermediário e você desejar que o intermediário autentique seu aplicativo utilizando essas informações, configure o servidor de nome de usuário e o intermediário para autenticação de senha simples de telnet.

Resultados

Agora é possível iniciar seu aplicativo.

Configurando o barramento de integração de serviços para um aplicativo que se conecta ao WebSphere Application Server

Antes de poder executar um aplicativo que se conecta a um barramento de integração de serviços WebSphere Application Server service integration technologies , você deve configurar a integração de serviço da mesma maneira que configurar o barramento de integração de serviços para executar aplicativos JMS que usam o provedor de sistemas de mensagens padrão.

Antes de começar

Antes de iniciar esta tarefa, você deve executar as seguintes etapas:

- Certifique-se de que um barramento do sistema de mensagens seja criado e que o servidor seja incluído no barramento como um membro do barramento.
- Certifique-se de que seu aplicativo tenha acesso a um barramento de integração de serviços que contém pelo menos um mecanismo de sistema de mensagens que esteja em execução.
- Se a operação HTTP, for necessária, um canal de transporte de entrada do mecanismo do sistema de mensagens HTTP deverá ser definido. Por padrão, os canais para SSL e TCP são definidos durante a instalação do servidor.
- Certifique-se de que seu aplicativo usa um connection factory cujas propriedades são configuradas corretamente para se conectar ao barramento de integração de serviços usando um servidor de autoinicialização. As informações mínimas necessárias são:
  - O terminal do provedor, que descreve o local e o protocolo a serem usados ao negociar uma conexão com o servidor de sistema de mensagens (ou seja, por meio do servidor de auto-inicialização). Em seu formato mais simples, para um servidor instalado com configurações padrão, o terminal de fornecimento pode ser configurado para o nome do host do servidor.
  - O nome do barramento por meio do qual as mensagens são enviadas.

  Para obter mais informações sobre as propriedades de um connection factory, consulte Propriedades do ConnectionFactory.

Sobre esta tarefa

Quaisquer espaços de fila ou de tópico que você precisa devem ser definidos. Por padrão, um espaço de tópico chamado Default.Topic.Space é definido durante a instalação do servidor, mas, se você precisar de espaços de tópico adicionais, deverá criar esses espaços de tópico você mesmo. Você não precisa predefinir tópicos individuais dentro de um espaço de tópico, uma vez que o servidor instancia esses tópicos individuais dinamicamente conforme necessário.

As etapas a seguir resumem o que você precisa fazer.

Procedimento

1. Crie as filas que seu aplicativo precisa para o sistema de mensagens ponto a ponto.
2. Crie quaisquer espaços de tópico adicionais que seu aplicativo precise para o sistema de mensagens de publicação / assinatura.

Resultados

Agora é possível iniciar seu aplicativo.

Usando os aplicativos de amostra XMS

Os aplicativos de amostra XMS.NET fornecem uma visão geral dos recursos comuns de cada API. É possível usá-los para verificar a sua instalação e o servidor de sistema de mensagens configurado e para ajudar a construir os seus próprios aplicativos.

Sobre esta tarefa

Se você precisar de ajuda para criar seus próprios aplicativos, será possível usar os aplicativos de amostra como um ponto de início. Tanto a origem quanto uma versão compilada são fornecidas para cada aplicativo. Revise o código-fonte de amostra e identifique as etapas principais para criar cada objeto necessário para seu aplicativo (ConnectionFactory, Conexão, Sessão, Destino, e um Produtor, ou um Consumidor, ou ambos) e para configurar quaisquer propriedades específicas que sejam necessárias para especificar como você deseja que seu aplicativo funcione. Para obter informações adicionais, consulte “Gravando aplicativos do XMS .NET” na página 643. As amostras estão sujeitas a mudanças em liberações futuras de XMS.

A tabela a seguir mostra os conjuntos de aplicativos de amostra (um para cada API) que são fornecidos com XMS.

Tabela 81. Aplicativos de amostra para o XMS .NET

Nome da amostra	Descrição
SampleConsumerCS	Um aplicativo consumidor de mensagens que obtém mensagens de uma fila ou assina um tópico.
SampleProducerCS	Um aplicativo de produtor de mensagem que produz mensagens para uma fila ou em um tópico.
SampleConfigCS	Um aplicativo de configuração que pode ser usado para criar um repositório de objeto administrado que se baseia em arquivo. O aplicativo contém um connection factory e um destino para suas configurações de conexão específicas. Esse repositório de objeto administrado pode, então, ser usado com cada um dos aplicativos de consumidor e de produtor de amostra.

As amostras que suportam as mesmas funções nas várias APIs possuem diferenças sintácticas.

- Os aplicativos de consumidor e de produtor de mensagens de amostra suportam as seguintes funções:
  - Conexões com IBM MQ, IBM Integration Bus (usando uma conexão em tempo real para um broker), e um WebSphere Application Server service integration bus
  - Consultas do Repositório de Objeto Administrado Usando a Interface de Contexto Inicial
  - Conexões com filas (IBM MQ e WebSphere Application Server service integration bus) e tópicos (IBM MQ, conexão em tempo real para um broker e WebSphere Application Server service integration bus)
  - Mensagens de base, de byte, de mapa, de objeto, de fluxo e de texto
- O aplicativo consumidor de mensagens de amostra suporta modos de recebimento síncronos e assíncronos e instruções SQL Selector.

- O aplicativo do produtor de mensagem de amostra suporta modos de entrega persistentes e não persistentes.

As amostras podem operar em um dos dois modos:

Modo Simples
É possível executar as amostras com a entrada mínima do usuário.

Modo Avançado
É possível customizar mais finamente a maneira na qual as amostras operam.

Todas as amostras são compatíveis e podem, portanto, operar por meio de linguagens.

Windows O IBM MQ suporta aplicativos .NET Core for XMS .NET em ambientes do Windows O IBM MQ classes for .NET Standard, incluindo amostras, são instalados por padrão como parte da instalação padrão do IBM MQ.

Linux O IBM MQ também suporta .NET Core para aplicativos em ambientes Linux .

Os aplicativos de amostra para XMS .NET são instalados em `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms`.

Para obter informações adicionais, consulte “Instalando IBM MQ classes for XMS .NET” na página 630.

Executando os aplicativos de amostra .NET

É possível executar os aplicativos de amostra .NET interativamente em modo simples ou avançado, ou não interativamente, usando arquivos de resposta gerados automaticamente ou gerados automaticamente.

Antes de começar

Antes de executar qualquer um dos aplicativos de amostra fornecidos, você deve primeiro configurar o ambiente do servidor de sistema de mensagens para que os aplicativos possam se conectar a um servidor. Consulte o “Configurando o Ambiente do Servidor de Mensagens” na página 634.

Procedimento

Para executar um aplicativo de amostra .NET, conclua as etapas a seguir:

Sugestão: Quando você estiver executando um aplicativo de amostra, digite ? em qualquer momento para obter ajuda sobre o que fazer em seguida.

1. Selecione o modo no qual você deseja executar o aplicativo de amostra.

   Digite `Advanced` ou `Simple`.
2. Responda às perguntas.

   Para selecionar o valor padrão, que é mostrado entre colchetes no final da pergunta, pressione Enter. Para selecionar um valor diferente, digite o valor apropriado e pressione Enter.

   Aqui está uma pergunta de exemplo:

   ```
   Enter connection type [wpm]:
   ```

   Nesse caso, o valor padrão é `wpm` (conexão com um WebSphere Application Server service integration bus).

Resultados

Quando você executa os aplicativos de amostra, os arquivos de resposta são gerados automaticamente no diretório de trabalho atual. Os nomes de arquivos de resposta estão no formato *`connection_type-sample_type`*`.rsp`; por exemplo, `wpm-producer.rsp`. Se necessário, é possível usar o arquivo de

resposta gerado para executar novamente o aplicativo de amostra com as mesmas opções, de modo que não tenha que inserir as opções novamente.

Tarefas relacionadas

Construindo os aplicativos de amostra .NET
Quando você constrói um aplicativo .NET de amostra, uma versão executável de sua amostra escolhida é criada.

Construindo seus próprios aplicativos
Você constrói seus próprios aplicativos, como você constrói os aplicativos de amostra.

Construindo os aplicativos de amostra .NET

Quando você constrói um aplicativo .NET de amostra, uma versão executável de sua amostra escolhida é criada.

Antes de começar

Instale o compilador apropriado. Essa tarefa supõe que o Microsoft Visual Studio 2012 esteja instalado e que você esteja familiarizado com sua utilização.

Procedimento

Para construir um aplicativo de amostra .NET , conclua as etapas a seguir:

1. Clique no arquivo de solução `Samples.sln` fornecido com as amostras .NET.
2. Clique com o botão direito do mouse na solução `Amostras` na janela Explorador de Solução e selecione **Construir Solução**.

Resultados

Um programa executável é criado na subpasta apropriada da amostra, seja `bin/Debug` ou `bin/Release`, dependendo da configuração que você escolheu. Este programa tem o mesmo nome da pasta, com um sufixo de CS. Por exemplo, se você está construindo a versão C# do aplicativo de amostra do produtor de mensagens, `SampleProducerCS.exe` é criado na pasta `SampleProducer` .

Tarefas relacionadas

Executando os aplicativos de amostra .NET
É possível executar os aplicativos de amostra .NET interativamente em modo simples ou avançado, ou não interativamente, usando arquivos de resposta gerados automaticamente ou gerados automaticamente.

Construindo seus próprios aplicativos
Você constrói seus próprios aplicativos, como você constrói os aplicativos de amostra.

“Construindo seus próprios aplicativos” na página 642
Você constrói seus próprios aplicativos, como você constrói os aplicativos de amostra.

Construindo seus próprios aplicativos

Você constrói seus próprios aplicativos, como você constrói os aplicativos de amostra.

Antes de começar

Instale o compilador apropriado. Essa tarefa supõe que o Microsoft Visual Studio 2012 esteja instalado e que você esteja familiarizado com sua utilização.

Procedimento

- Construa seu aplicativo .NET , conforme descrito em “Construindo os aplicativos de amostra .NET” na página 642.

  Para obter orientação adicional sobre como construir seus próprios aplicativos, use os makefiles fornecidos para cada aplicativo de amostra.

Sugestão: Para ajudar no diagnóstico de problemas no caso de uma falha, é possível achar útil compilar aplicativos com símbolos incluídos.

Tarefas relacionadas

Executando os aplicativos de amostra .NET
É possível executar os aplicativos de amostra .NET interativamente em modo simples ou avançado, ou não interativamente, usando arquivos de resposta gerados automaticamente ou gerados automaticamente.

Construindo os aplicativos de amostra .NET
Quando você constrói um aplicativo .NET de amostra, uma versão executável de sua amostra escolhida é criada.

Gravando aplicativos do XMS .NET

Esta seção fornece informações para ajudar a gravar aplicativos XMS .NET , incluindo informações sobre propriedades, tipos de dados e manipulação de erros.

Antes de começar

Deprecated V 9.4.0 V 9.4.0 Em IBM MQ 9.4.0, em IBM MQ classes for XMS .NET, os métodos WriteObject(), ReadObject(), CreateObjectMessage () e as classes ObjectMessage e XmsObjectMessageImpl usados para serialização e desserialização de dados foram descontinuados.

V 9.4.0 Removed V 9.4.0 A biblioteca do cliente do XMS .NET construída usando.NET Standard 2.0, que foi descontinuada em IBM MQ 9.3.1, foi removida do produto em IBM MQ 9.4.0

A partir de IBM MQ 9.2.0, o número de bibliotecas de link dinâmico do XMS .NET foi significativamente reduzido para um total de cinco. As cinco bibliotecas de vínculo dinâmico são:

- `IBM.XMS.dll` - inclui todas as mensagens de idioma nacional
- `IBM.XMS.Comms.RMM.dll`
- Três bibliotecas de vínculo dinâmico de política:
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

No XMS .NET, todas as sequências são transmitidas usando a sequência .NET nativa. Como isso tem uma codificação fixa, nenhuma informação adicional é necessária para interpretá-la. Portanto, a propriedade XMSC_CLIENT_CCSID não é necessária para aplicativos XMS .NET .

Sobre esta tarefa

Esta seção contém os seguintes tópicos:

- “Exclusão de objeto” na página 661
- “Tipos de dados para XMS.NET” na página 661
- “Tipos primitivos do XMS” na página 662
- “Conversão implícita de um valor de propriedade de um tipo de dados para outro” na página 663
- “Iteradores” na página 665
- “Manipulação de erros no XMS .NET” na página 665
- “Usando listeners de mensagem e de exceção em .NET” na página 666
- “IBM MQ Reconexão de cliente automática por meio de XMS” na página 667

Operações gerenciadas e não gerenciadas no .NET

O código gerenciado é executado exclusivamente dentro do ambiente Common Language Runtime do .NET e depende totalmente dos serviços fornecidos por esse tempo de execução. Um aplicativo é classificado como não gerenciado se qualquer parte do aplicativo executar ou chamar serviços fora do ambiente de tempo de execução de linguagem comum .NET .

Determinadas funcionalidades avançadas não podem ser suportadas atualmente dentro do ambiente gerenciado do .NET.

Se o seu aplicativo requer alguma funcionalidade que não é atualmente suportado no ambiente totalmente gerenciado, é possível alterar seu aplicativo para usar o ambiente não gerenciado sem requerer uma mudança substancial no seu aplicativo. No entanto, é necessário observar que a pilha XMS faz uso de código não gerenciado quando essa seleção é feita.

Conexões com um gerenciador de filas IBM MQ

Conexões gerenciadas com WMQ_CM_CLIENT não suportará comunicações não TCP e compactação de canal. No entanto, essas conexões podem ser suportadas pelo uso de uma conexão não gerenciada (WMQ_CM_CLIENT_UNMANAGED). Para obter informações adicionais, consulte “Desenvolvendo aplicativos do .NET” na página 565.

Se você criar um connection factory a partir de um objeto administrado em um ambiente não gerenciado, você deverá alterar manualmente o valor para o modo de conexão para XMSC_WMQ_CM_CLIENT_UNMANAGED.

Conexões com um mecanismo de mensagens de barramento de integração de serviços WebSphere Application Server

Conexões com um mecanismo de sistema de mensagens do barramento de integração de serviços WebSphere Application Server que requerem o uso do protocolo SSL (incluindo HTTPS) não são atualmente suportadas como código gerenciado.

Windows Usando o modelo de projeto IBM MQ XMS .NET

O cliente IBM MQ XMS .NET oferece a você a capacidade de usar um modelo de projeto para auxiliá-lo no desenvolvimento de seus aplicativos XMS .NET Core.

Antes de começar

Deve-se ter o Microsoft Visual Studio 2017, ou mais recente, e o .NET Core 2.1 instalado no seu sistema.

Deve-se copiar o modelo XMS .NET do diretório

```
&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip
```

para o diretório

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

em que:

- *&MQ_INSTALL_ROOT* é o diretório-raiz da sua instalação
- *& USER_HOME_DIRECTORY* é o seu diretório inicial.

Deve-se parar e reiniciar Microsoft Visual Studio para escolher o modelo.

Sobre esta tarefa

O modelo de projeto XMS .NET inclui algum código comum que pode ser usado para ajudar a desenvolver os seus aplicativos. Com o código integrado, é possível conectar-se ao gerenciador de filas IBM MQ e executar uma operação put ou get, simplesmente modificando as propriedades no código integrado.

Procedimento

1. Abra o Microsoft Visual Studio.
2. Clique em **Arquivo**, seguido por **Novo** e, em seguida, **Projeto**.
3. Na *janela Criar um novo projeto*, selecione `IBM XMS .NET Client App (.NET Core)` e clique em **Avançar**..
4. Na janela *Configurar seu novo projeto*, mude o *Nome do projeto* de seu projeto, se desejar, e clique em **Criar** para criar o projeto do XMS .NET.

   `XMSDotnetApp.cs` é o arquivo que é criado junto com o arquivo do projeto. Esse arquivo contém o código que se conecta ao gerenciador de filas e executa as operações de envio e recebimento.

   As propriedades de conexão são configuradas para valores padrão:

   - WMQ_CONNECTION_NAME_LIST é configurado como *localhost(1414)*
   - XMSC.WMQ_CHANNEL é configurado como *DOTNET.SVRCONN*

   A fila é configurada como *Q1* e é possível modificar essas propriedades adequadamente.
5. Compile e execute o aplicativo.

Conceitos relacionados

Componentes e recursos do IBM MQ

Aplicativo de tempo de execução do.NET - Windows somente

O modelo de encadeamento

As regras gerais controlam como um aplicativo multiencadeado pode usar objetos XMS .

- Apenas objetos dos tipos a seguir podem ser usados simultaneamente em encadeamentos diferentes:
  - ConnectionFactory
  - Conexão
  - ConnectionMetaData
  - Destino
- Um objeto de Sessão pode ser usado em apenas um único encadeamento em um determinado momento.

As exceções para essas regras são indicadas por entradas rotuladas "Contexto de Encadeamento" nas definições de interface dos métodos no Referência do IBM Message Service Client for .NET.

Propriedades em XMS.NET

Um aplicativo .NET usa os métodos na interface PropertyContext para obter e configurar as propriedades de objetos. O manuseio de propriedades inexistentes no XMS .NET é amplamente consistente com a especificação JMS, e também com as implementações C e C++ de XMS.

propriedades do XMS .NET e seus valores

A interface PropertyContext encapsula métodos que obtem e configuram propriedades. Esses métodos são herdados, direta ou indiretamente, pelas classes a seguir:

- BytesMessage
- Conexão
- ConnectionFactory
- ConnectionMetaData
- Destino
- MapMessage
- Mensagem
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Sessão
- StreamMessage
- TextMessage

Se um aplicativo configurar o valor de uma propriedade, o novo valor substituirá qualquer valor anterior que a propriedade possuiu. Para obter mais informações sobre propriedades do XMS, consulte Propriedades de objetos do XMS.

Para facilidade de uso, os nomes e valores da propriedade XMS em XMS são predefinidos como constantes públicas em uma estrutura chamada XMSC. Os nomes dessas constantes estão no formato XMSC.*constant*; por exemplo, XMSC.USERID (uma constante de nome de propriedade) e XMSC.DELIVERY_AS_APP (uma constante de valor).

Além disso, é possível acessar constantes IBM MQ usando o IBM.XMS.MQC struct. Se o namespace IBM.XMS já estiver importado, será possível acessar os valores para essas propriedades no formato MQC.*constant*. Por exemplo, MQC.MQRO_COA_WITH_FULL_DATA.

Se você tiver um aplicativo híbrido que usa as classes XMS .NET e IBM MQ para .NET e que importa ambos IBM.XMS e IBM.WMQ , em seguida, deve-se qualificar totalmente o namespace de estrutura do MQC para assegurar que cada ocorrência seja exclusiva

Nota: Algumas funcionalidades avançadas não são atualmente suportadas dentro do ambiente gerenciado .NET . Para obter mais informações, consulte “Operações gerenciadas e não gerenciadas no .NET” na página 644.

Manipulação de propriedades não existentes no XMS .NET

No JMS, o acesso a uma propriedade não existente pode resultar em uma exceção do sistema Java quando um método tenta converter o valor não existente (nulo) para o tipo necessário. Se uma propriedade não existir, ocorrerão as seguintes exceções:

- getStringProperty e getObjectProperty return null
- getBooleanProperty retorna false porque Boolean.valueOf (null) retorna false
- getIntProperty etc.throw java.lang.NumberFormatException porque Integer.valueOf (null) throws a exceção

Se uma propriedade não existir no XMS .NET, ocorrerão as seguintes exceções:

- GetStringProperty e GetObjectProperty (e GetBytesProperty) retornam nulo (que é o mesmo que Java)
- GetBooleanProperty throws System.NullReferenceException
- GetIntProperty etc. throws throws System.NullReferenceException

Essa implementação é diferente de Java, mas ela é amplamente consistente com a especificação JMS, e com as interfaces C e C++ XMS. Assim como a implementação Java, XMS .NET propaga quaisquer exceções da chamada System.Convert para o responsável pela chamada. Ao contrário de Java, no entanto, XMS lança explicitamente NullReferenceExceptions em vez de apenas usar o comportamento nativo da estrutura .NET através de passagem de nulo para rotinas de conversão do sistema. Se seu aplicativo configurar uma propriedade para uma Cadeia como "abc" e chamar GetIntProperty, o System.FormatException lançado pelo Convert.ToInt32 ("abc") será propagado para o responsável pela chamada, o que é consistente com Java. MessageFormatException será lançada apenas se os tipos usados para setProperty e getProperty forem incompatíveis. Esse comportamento também é consistente com Java.

ConnectionFactories e objetos de Conexão

Um objeto ConnectionFactory fornece um modelo que um aplicativo usa para criar um objeto Connection. O aplicativo usa o objeto Connection para criar um objeto Session.

Para .NET, o aplicativo XMS usa primeiro um objeto XMSFactoryFactory para obter uma referência a um objeto ConnectionFactory que seja adequado ao tipo de protocolo necessário. Esse objeto ConnectionFactory pode, então, produzir conexões somente para esse tipo de protocolo.

Um aplicativo XMS pode criar várias conexões, e um aplicativo multiencadeado pode usar um único objeto de Conexão simultaneamente em vários encadeamentos. Um objeto Connection encapsula uma conexão de comunicações entre um aplicativo e um servidor de sistema de mensagens.

Uma conexão atende a vários propósitos:

- Quando um aplicativo cria uma conexão, o aplicativo pode ser autenticado.
- Um aplicativo pode associar um identificador de cliente exclusivo a uma conexão. O identificador de cliente é usado para suportar assinaturas duráveis no domínio de publicação / assinatura. O identificador de cliente pode ser configurado de duas maneiras:

  A maneira preferida de designar um identificador de cliente de conexões é configurar em um objeto ConnectionFactory específico do cliente usando propriedades e designá-lo de forma transparente à conexão criada.

  Uma maneira alternativa de designar um identificador de cliente é usar um valor específico do provedor que é configurado no objeto Connection. Esse valor não substitui o identificador que foi configurado administrativamente. Ele é fornecido para o caso em que não existe nenhum identificador administrativamente especificado. Se um identificador especificado administrativamente existir, uma tentativa de substituí-la com um valor específico do provedor fará com que uma exceção seja lançada. Se um aplicativo configurar explicitamente um identificador, ele deverá fazer isso imediatamente após a criação da conexão e antes de qualquer outra ação na conexão ser tomada; caso contrário, uma exceção será lançada.

Um aplicativo XMS normalmente cria uma conexão, uma ou mais sessões e um número de produtores de mensagens e consumidores de mensagens.

A criação de uma conexão é relativamente cara em termos de recursos do sistema, porque envolve o estabelecimento de uma conexão de comunicação e pode também envolver a autenticação do aplicativo.

Modo iniciado e interrompido da conexão

Uma conexão pode operar no modo iniciado ou interrompido.

Quando um aplicativo cria uma conexão, a conexão fica no modo interrompido. Quando a conexão está no modo interrompido, o aplicativo pode inicializar as sessões e pode enviar mensagens, mas não pode recebê-las, de forma síncrona ou assíncrona.

Um aplicativo pode iniciar uma conexão chamando o método `Start Connection` . Quando a conexão está no modo iniciado, o aplicativo pode enviar e receber mensagens. O aplicativo pode, então, parar e reiniciar a conexão chamando os métodos Stop Connection e `Start Connection` .

Fechamento da conexão

Um aplicativo fecha uma conexão chamando o método Fechar Conexão. Quando um aplicativo fecha uma conexão, XMS executa as ações a seguir:

- Ele fecha todas as sessões associadas à conexão e exclui determinados objetos associados a essas sessões. Para obter mais informações sobre quais objetos são excluídos, consulte “Exclusão de objeto” na página 661. Ao mesmo tempo, XMS retrocede quaisquer transações atualmente em andamento dentro das sessões.
- Ele encerra a conexão de comunicações com o servidor de sistema de mensagens.
- Ele libera a memória e outros recursos internos usados pela conexão.

O XMS não reconhece o recebimento de nenhuma mensagem que ele tenha falhado em reconhecer durante uma sessão antes de fechar a conexão. Para obter mais informações sobre como reconhecer o recebimento de mensagens, consulte “Reconhecimento de mensagem” na página 650.

Tratamento de exceção

XMS .NET exceções são todas derivadas de System.Exception.. Para obter mais informações, consulte “Manipulação de erros no XMS .NET” na página 665.

Conexão com um barramento de integração de serviços

Um aplicativo XMS pode se conectar a um barramento de integração de serviços WebSphere Application Server usando uma conexão TCP/IP direta ou usando HTTP sobre TCP/IP.

O protocolo HTTP pode ser usado em situações em que uma conexão TCP/IP direta não é possível. Uma situação comum é quando se comunica através de um firewall, como quando duas empresas trocam mensagens. Usar HTTP para se comunicar por meio de um firewall é frequentemente referido como *tunelamento HTTP*. O tunelamento HTTP, no entanto, é inerentemente mais lento do que o uso de uma conexão TCP/IP direta, porque os cabeçalhos de HTTP incluem significativamente a quantidade de dados que são transferidos e porque o protocolo HTTP requer mais fluxos de comunicação do que o TCP/IP.

Para criar uma conexão TCP/IP, um aplicativo pode usar um connection factory cuja propriedade XMSC_WPM_TARGET_TRANSPORT_CHAIN esteja configurada como XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC. Esse é o valor padrão da propriedade. Se a conexão for criada com sucesso, a propriedade XMSC_WPM_CONNECTION_PROTOCOL da conexão será configurada como XMSC_WPM_CP_TCP.

Para criar uma conexão que utiliza HTTP, um aplicativo deve usar um connection factory cuja propriedade XMSC_WPM_TARGET_TRANSPORT_CHAIN esteja configurada para o nome de uma cadeia de transporte de entrada, que está configurada para usar um canal de transporte HTTP. Se a conexão for criada com sucesso, a propriedade XMSC_WPM_CONNECTION_PROTOCOL da conexão será configurada como XMSC_WPM_CP_HTTP. Para obter informações sobre como configurar cadeias de transporte, consulte Configurando cadeias de transporte na documentação do produto WebSphere Application Server .

Um aplicativo possui uma opção semelhante de protocolos de comunicação ao se conectar a um servidor de auto-inicialização. A propriedade XMSC_WPM_PROVIDER_ENDPOINTS de um connection factory é uma sequência de um ou mais endereços de terminal de servidores de autoinicialização. O componente de cadeia de transporte de auto-inicialização de cada endereço de terminal pode ser XMSC_WPM_BOOTSTRAP_TCP, para uma conexão TCP/IP com um servidor de auto-inicialização ou XMSC_WPM_BOOTSTRAP_HTTP, para uma conexão que usa HTTP.

Sessões

Uma sessão é um único contexto encadeado para enviar e receber mensagens.

Um aplicativo pode usar uma sessão para criar mensagens, produtores de mensagens, consumidores de mensagens, navegadores de filas e destinos temporários. Um aplicativo também pode usar uma sessão para executar transações locais.

Um aplicativo pode criar várias sessões, em que cada sessão produz e consome mensagens independentemente das outras sessões. Se dois consumidores de mensagem em sessões separadas (ou mesmo na mesma sessão) assinam o mesmo tópico, cada um recebe uma cópia de qualquer mensagem publicada nesse tópico.

Diferente de um objeto Connection, um objeto Session não pode ser usado simultaneamente em encadeamentos diferentes. Apenas o método Fechar Sessão de um objeto de Sessão pode ser chamado a partir de um encadeamento diferente daquele que o objeto Session está usando no momento. O método Fechar Sessão termina uma sessão e libera os recursos do sistema alocados para a sessão.

Se um aplicativo precisar processar mensagens simultaneamente em mais de um encadeamento, o aplicativo deverá criar uma sessão em cada encadeamento e, em seguida, usar essa sessão para qualquer operação de envio ou recebimento dentro desse encadeamento.

Sessões transacionadas

Os aplicativos XMS podem executar transações locais. Uma *transação local* é aquela que envolve mudanças apenas para os recursos do gerenciador de filas ou do barramento de integração de serviços ao qual o aplicativo está conectado.

As informações neste tópico são relevantes apenas se um aplicativo se conectar a um gerenciador de filas IBM MQ ou a um barramento de integração de serviços WebSphere Application Server . As informações não são relevantes para uma conexão em tempo real com um broker.

Para executar transações locais, um aplicativo deve primeiro criar uma sessão transacionada ao chamar o método Create Session de um objeto Connection, especificando como um parâmetro que a sessão é transacionada. Em seguida, todas as mensagens enviadas e recebidas dentro da sessão são agrupadas em uma sequência de transações. Uma transação terminará quando o aplicativo confirmar ou recuperar as mensagens que ele enviou e recebeu desde o início da transação.

Para confirmar uma transação, um aplicativo chama o método Commit do objeto Session. Quando uma transação for confirmada, todas as mensagens enviadas dentro da transação se tornarão disponíveis para entrega para outros aplicativos e todas as mensagens recebidas dentro da transação serão confirmadas para que o servidor de sistema de mensagens não tente entregá-las ao aplicativo novamente. No domínio ponto a ponto, o servidor de sistema de mensagens também remove as mensagens recebidas a partir de suas filas.

Para retroceder uma transação, um aplicativo chama o método Rollback do objeto Session. Quando uma transação for retrocedida, todas as mensagens enviadas dentro da transação serão descartadas pelo servidor de sistema de mensagens e todas as mensagens recebidas dentro da transação se tornarão disponíveis para entrega novamente. No domínio de ponto a ponto, as mensagens que foram recebidas são colocadas de volta em suas filas e se tornam visíveis a outros aplicativos novamente.

Uma nova transação é iniciada automaticamente quando um aplicativo cria uma sessão transacionada ou chama o método Commit ou Rollback. Portanto, uma sessão transacionada sempre possui uma transação ativa.

Quando um aplicativo fechar uma sessão transacionada, um retrocesso implícito ocorrerá. Quando um aplicativo fechar uma conexão, um retrocesso implícito ocorrerá para todas as sessões transacionadas da conexão.

Uma transação é completamente contida dentro de uma sessão transacionada. Uma transação não pode abranger as sessões. Isso significa que não é possível para um aplicativo enviar e receber mensagens em duas ou mais sessões transacionadas e, em seguida, confirmar ou retroceder todas estas ações como uma única transação.

Conceitos relacionados

Reconhecimento de mensagem

Cada sessão que não é transacionada possui um modo de confirmação que determina como as mensagens recebidas pelo aplicativo são confirmadas. Três modos de confirmação estão disponíveis e a opção de modo de confirmação afeta o design do aplicativo.

Entrega de mensagem

XMS suporta modos persistentes e não persistentes de entrega de mensagens e entrega assíncrona e síncrona de mensagens.

Transações XA IBM MQ gerenciadas através de XMS
As transações XA IBM MQ gerenciadas podem ser usadas por meio de XMS.

Reconhecimento de mensagem

Cada sessão que não é transacionada possui um modo de confirmação que determina como as mensagens recebidas pelo aplicativo são confirmadas. Três modos de confirmação estão disponíveis e a opção de modo de confirmação afeta o design do aplicativo.

Nota: Este tópico será relevante apenas se um aplicativo se conectar a um gerenciador de fila do IBM MQ ou a um barramento de integração de serviços do WebSphere Application Server As informações não são relevantes para uma conexão em tempo real com um broker.

XMS usa o mesmo mecanismo para reconhecer o recebimento de mensagens que o JMS utiliza.

Se uma sessão não for transacionada, a maneira que as mensagens recebidas pelo aplicativo são confirmadas será determinada pelo modo de confirmação da sessão. Há três modos de confirmação:

XMSC_AUTO_ACKNOWLEDGE
A sessão confirma automaticamente cada mensagem recebida pelo aplicativo.

Se as mensagens forem entregues de forma síncrona para o aplicativo, a sessão confirmará o recebimento de uma mensagem toda vez que uma chamada Receive for concluída com sucesso. Se o aplicativo receber uma mensagem com sucesso, mas uma falha evitar a ocorrência de confirmação, a mensagem se tornará disponível para a entrega novamente. O aplicativo deve, portanto, ser capaz de manipular uma mensagem que seja entregue novamente.

XMSC_DUPS_OK_ACKNOWLEDGE
A sessão confirma as mensagens recebidas pelo aplicativo em momentos que ele seleciona.

O uso desse modo de confirmação reduz a quantidade de trabalho que a sessão deve fazer, mas uma falha que evita a confirmação de mensagens pode fazer com que mais de uma mensagem se torne disponível para entrega novamente. O aplicativo deve, portanto, ser capaz de manipular mensagens que são entregues novamente.

XMSC_CLIENT_ACKNOWLEDGE
O aplicativo confirma as mensagens que ele recebe chamando o método Acknowledge da classe Message.

O aplicativo pode confirmar o recebimento de cada mensagem individualmente ou receber um lote de mensagens e chamar o método Acknowledge apenas para a última mensagem que ele recebe. Quando o método Acknowledge for chamado, todas as mensagens recebidas desde a última vez que o método foi chamado serão confirmadas.

Juntamente com qualquer um destes modos de confirmação, um aplicativo pode interromper e reiniciar a entrega de mensagens em uma sessão chamando o método Recover da classe Session. Mensagens cujo recebimento tenha sido anteriormente não reconhecido são entregues novamente. No entanto, elas não podem ser entregues na mesma sequência em que foram entregues anteriormente. No entretanto, mensagens de prioridade superior podem ter chegado e algumas das mensagens originais podem ter expirado. No domínio ponto a ponto, algumas das mensagens originais podem ter sido consumidas por outro aplicativo.

Um aplicativo pode determinar se uma mensagem está sendo entregue novamente examinando o conteúdo do campo de cabeçalho JMSRedelivered da mensagem. O aplicativo faz isso chamando o método Get JMSRedelivered da classe Message.

Conceitos relacionados
Sessões transacionadas
Os aplicativos XMS podem executar transações locais. Uma *transação local* é aquela que envolve mudanças apenas para os recursos do gerenciador de filas ou do barramento de integração de serviços ao qual o aplicativo está conectado.

Entrega de mensagem

XMS suporta modos persistentes e não persistentes de entrega de mensagens e entrega assíncrona e síncrona de mensagens.

Transações XA IBM MQ gerenciadas através de XMS
As transações XA IBM MQ gerenciadas podem ser usadas por meio de XMS.

Entrega de mensagem

XMS suporta modos persistentes e não persistentes de entrega de mensagens e entrega assíncrona e síncrona de mensagens.

Modos de entrega de mensagem

XMS suporta dois modos de entrega de mensagens:

- Persistente

  Mensagens persistentes são entregues uma vez. Um servidor de sistema de mensagens toma precauções especiais, como a criação de log das mensagens, para assegurar que as mensagens persistentes não sejam perdidas em trânsito, mesmo no caso de uma falha.

- Não persistente

  Mensagens não persistentes são entregues não mais de uma vez. Mensagens não persistentes são menos confiáveis do que as mensagens persistentes porque elas podem ser perdidas em trânsito no caso de uma falha.

A opção de modo de entrega é uma troca entre confiabilidade e desempenho. Mensagens não persistentes são geralmente transportadas mais rapidamente do que as mensagens persistentes.

Entrega de Mensagem Assíncrona

XMS usa um encadeamento para manipular todas as entregas de mensagens assíncronas para uma sessão. Isso significa que apenas uma função de listener de mensagem ou um método `onMessage()` pode ser executado de cada vez.

Se mais de um consumidor de mensagens em uma sessão estiver recebendo mensagens de forma assíncrona, e uma função de listener de mensagem ou um método `onMessage()` estiver entregando uma mensagem para um consumidor de mensagens, qualquer outro consumidor de mensagens que estiver aguardando a mesma mensagem deverá continuar aguardando. Outras mensagens que estão aguardando para serem entregues para a sessão também devem continuar aguardando.

Se um aplicativo requerer a entrega simultânea de mensagens, crie mais de uma sessão para que XMS use mais de um encadeamento para manipular a entrega de mensagens assíncronas. Dessa maneira, mais de uma função do listener de mensagem ou método `onMessage()` pode ser executado simultaneamente.

Uma sessão não é feita assíncrona, atribuindo um listener de mensagem a um consumidor. Uma sessão se torna assíncrona somente quando o método `Connection.Start` é chamado. Todas as chamadas síncronas são permitidas até que o método `Connection.Start` seja chamado. A entrega de mensagem para os consumidores é iniciada quando o `Connection.Start` é chamado

Se as chamadas síncronas, como a criação de um consumidor ou um produtor, devem ser feitas em uma sessão assíncrona, o `Connection.Stop` deve ser chamado. Uma sessão pode ser continuada chamando o método `Connection.Start` para iniciar a entrega de mensagens. A única exceção a isso é o encadeamento de entrega de mensagens da sessão, que é aquele que entrega mensagens para a função de retorno de chamada. Esse encadeamento pode fazer qualquer chamada na sessão (exceto uma chamada Fechar) na função de retorno de chamada de mensagem.

Nota: No modo Não gerenciado, a chamada MQDISC dentro de uma função de retorno de chamada não é suportada pelo cliente IBM MQ .NET. Portanto, o aplicativo cliente não pode Criar ou Fechar sessões dentro do retorno de chamada MessageListener no modo de recebimento Assíncrono. Crie e despose a sessão fora do método `MessageListener`.

Entrega de Mensagem Síncrona

As mensagens são entregues de forma síncrona para um aplicativo se o aplicativo usar os métodos Receive de MessageConsumer objetos.

Usando os métodos Receive, um aplicativo pode esperar um período de tempo especificado para uma mensagem ou pode esperar indefinidamente. Como alternativa, se um aplicativo não desejar aguardar por uma mensagem, ele poderá usar o método Receive with No Wait.

Conceitos relacionados

Sessões transacionadas
Os aplicativos XMS podem executar transações locais. Uma *transação local* é aquela que envolve mudanças apenas para os recursos do gerenciador de filas ou do barramento de integração de serviços ao qual o aplicativo está conectado.

Reconhecimento de mensagem
Cada sessão que não é transacionada possui um modo de confirmação que determina como as mensagens recebidas pelo aplicativo são confirmadas. Três modos de confirmação estão disponíveis e a opção de modo de confirmação afeta o design do aplicativo.

Transações XA IBM MQ gerenciadas através de XMS
As transações XA IBM MQ gerenciadas podem ser usadas por meio de XMS.

Transações XA IBM MQ gerenciadas através de XMS

As transações XA IBM MQ gerenciadas podem ser usadas por meio de XMS.

Para usar transações XA por meio de XMS, uma sessão transacionada precisa ser criada. Quando a transação XA está em uso, o controle de transação é por meio de transações globais do Distributed Transaction Coordinator (DTC) e não são as sessões XMS . Ao usar transações XA, `Session.commit` ou `Session.rollback` não podem ser emitidas na sessão XMS. Em vez disso, use os métodos DTC `Transscope.Commit` ou `Transscope.Rollback` consolidando ou retroceda as transações. Se uma sessão for usada para transação XA, o produtor ou o consumidor que são criados usando a sessão deve ser uma parte da transação XA. Eles não podem ser usados para nenhuma operação fora do escopo de transação XA. Eles não podem ser usados para operações como `Producer.send` ou `Consumer.receive` fora da transação XA.

Um objeto de exceção `IllegalStateException` é emitido se:

- A sessão transacionada por XA é usada para `Session.commit` ou `Session.rollback`.
- Os objetos do produtor ou do consumidor que são usados uma vez na sessão transacionada XA são usados fora do escopo de transação XA.

As transações XA não são suportadas em consumidores assíncronos.

Nota:

1. Um fechamento pode ser emitido no objeto `Producer`, `Consumer`, `Session`, ou `Connection` antes da confirmação da transação XA Em quais casos, as mensagens na transação são retrocedidas. Da mesma forma, se a conexão for interrompida antes da confirmação da transação XA, todas as mensagens na transação serão retrocedidas. Para um objeto `Producer` , um retrocesso significa que as mensagens não são colocadas na fila. Para um objeto `Consumer` , uma recuperação significa que as mensagens permanecem na fila.
2. Se um objeto `Producer` colocar uma mensagem com `TimeToLive` no `TransactionScope` e um `commit` for emitido após o tempo decorrido, a mensagem poderá expirar antes do `commit` ser emitido. Nesse caso, a mensagem não é disponibilizada para os objetos `Consumer`
3. Objetos `Session` não são suportados em encadeamentos. O uso de transações com objetos `Session` que são compartilhados entre encadeamentos não é suportado.

Conceitos relacionados

Sessões transacionadas

Os aplicativos XMS podem executar transações locais. Uma *transação local* é aquela que envolve mudanças apenas para os recursos do gerenciador de filas ou do barramento de integração de serviços ao qual o aplicativo está conectado.

Reconhecimento de mensagem
Cada sessão que não é transacionada possui um modo de confirmação que determina como as mensagens recebidas pelo aplicativo são confirmadas. Três modos de confirmação estão disponíveis e a opção de modo de confirmação afeta o design do aplicativo.

Entrega de mensagem
XMS suporta modos persistentes e não persistentes de entrega de mensagens e entrega assíncrona e síncrona de mensagens.

Destinos

Um aplicativo XMS usa um objeto de Destino para especificar o destino das mensagens que estão sendo enviadas e a origem de mensagens que estão sendo recebidas.

Um aplicativo XMS pode criar um objeto Destination no tempo de execução ou obter um destino predefinido a partir do repositório de objetos administrados.

Como com um ConnectionFactory, a maneira mais flexível para um aplicativo XMS para especificar um destino é defini-la como um objeto administrado. Usando essa abordagem, os aplicativos gravados em linguagens C, C + + e .NET e Javapodem compartilhar definições do destino. As propriedades de objetos de Destino administrados podem ser mudadas sem alterar qualquer código.

Destinos no .NET

Em .NET, os destinos são criados de acordo com o tipo de protocolo e podem ser usados apenas no tipo de protocolo para o qual eles são criados. Dois métodos são fornecidos para criar destinos, um para tópicos e um para filas:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

Esses métodos estão disponíveis nos dois objetos a seguir na API .NET :

- ISession
- XMSFactoryFactory

Em ambos os casos, esses métodos podem aceitar uma sequência de estilo de URI, que pode incluir parâmetros, no seguinte formato:

```
"topic://some/topic/name?priority=5"
```

Como alternativa, esses métodos podem aceitar um nome de destino apenas, ou seja, um nome sem um tópico: // ou queue: // prefix e sem parâmetros. Isso significa que a sequência de estilo de URI a seguir:

```
CreateTopic("topic://some/topic/name");
```

produziria o mesmo resultado que o nome de destino a seguir:

```
CreateTopic("some/topic/name");
```

Para obter mais informações, consulte IDestination.

Como para o WebSphere Application Server barramento de integração de serviços JMS, os tópicos também podem ser especificados em um formato abreviado, que inclui o *topicname* e o *topicspace* , mas não pode incluir parâmetros:

```
CreateTopic("topicspace:topicname");
```

Identificadores de recursos uniformes do tópico

O URI (Identificador Uniforme de Recursos (URI) do tópico especifica o nome do tópico; ele também pode especificar uma ou mais propriedades para ele.

O URI para um tópico inicia com o tópico de sequência: //, seguido pelo nome do tópico e (opcional) uma lista de pares nome-valor que configuram as propriedades do tópico restantes. Um nome de tópico não pode estar vazio.

Aqui está um exemplo em um fragmento de código .NET :

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

Para obter mais informações sobre as propriedades de um tópico, incluindo o nome e os valores válidos que podem ser usados em um URI, consulte Propriedades do Destino.

Ao especificar um URI de tópico para uso em uma assinatura, os curingas podem ser usados. A sintaxe para esses curinga depende do tipo de conexão e da versão do broker; a opção a seguir está disponível:

- Barramento de integração de serviços WebSphere Application Server

Barramento de integração de serviços WebSphere Application Server

O barramento de integração de serviços WebSphere Application Server usa os seguintes caracteres curinga:

* para corresponder a qualquer caractere em um nível na hierarquia
// para corresponder a 0 ou mais níveis
//. para combinar com 0 ou mais níveis (no final de uma expressão Tópico)

Tabela 82 na página 654 fornece alguns exemplos de como usar esse esquema curinga.

Tabela 82. URIs de exemplo usando esquema curinga para o barramento de integração de serviços WebSphere Application Server

Identificador Uniforme de Recursos	**Correspondentes**	**Exemplos**
"topic://Sport/ * ball/ Results"	Todos os tópicos com um nome de nível hierárquico único terminando em "ball" entre Esporte e Resultados	"topic://Sport/Football/Results" e "topic://Sport/Netball/Results"
"topic://Sport// Results"	Todos os tópicos iniciando com "Sport/" e terminando em "/Results"	"topic://Sport/Football/Results" e "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/ Football//."	Todos os tópicos iniciando com "Sport/ Football/"	"topic://Sport/Football/Results" e "topic://Sport/Football/TeamNews/ Signings/Managerial"
"topic://Sport/ * ball// Results//."	tópicos	"topic://Sport/Football/Results" e "topic://Sport/Netball/National/Div3/ Results/2002/November"

Conceitos relacionados

Identificadores uniformes de recursos da fila

O URI para uma fila especifica o nome da fila; ele também pode especificar uma ou mais propriedades da fila.

Destinos Temporários
Os aplicativos XMS podem criar e usar destinos temporários.

Identificadores uniformes de recursos da fila

O URI para uma fila especifica o nome da fila; ele também pode especificar uma ou mais propriedades da fila.

O URI para uma fila começa com a sequência `queue://`, seguido pelo nome da fila; ele também pode incluir uma lista de pares de valor de nome que configuram as propriedades restantes da fila.

Para as filas IBM MQ (mas não para filas do provedor de mensagens padrão WebSphere Application Server), o gerenciador de filas no qual a fila reside pode ser especificado antes da fila, com um / separando o nome do gerenciador de filas do nome da fila.

Se um gerenciador de filas for especificado, ele deverá ser aquele para o qual o XMS está diretamente conectado para a conexão usando essa fila ou deve ser acessível a partir desta fila. Os gerenciadores de filas remotas são suportados apenas para recuperar mensagens das filas, não para colocar mensagens nas filas. Para obter detalhes completos, consulte a documentação do gerenciador de filas IBM MQ .

Se nenhum gerenciador de filas for especificado, então, o extra / separador será opcional e sua presença ou ausência não fará diferença para a definição da fila.

As definições de fila a seguir são equivalentes para uma fila do IBM MQ chamada QB em um gerenciador de filas chamado QM_A, à qual o XMS é conectado diretamente:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

Conceitos relacionados

Identificadores de recursos uniformes do tópico
O URI (Identificador Uniforme de Recursos (URI) do tópico especifica o nome do tópico; ele também pode especificar uma ou mais propriedades para ele.

Destinos Temporários
Os aplicativos XMS podem criar e usar destinos temporários.

Destinos Temporários

Os aplicativos XMS podem criar e usar destinos temporários.

Um aplicativo geralmente usa um destino temporário para receber respostas para solicitar mensagens. Para especificar o destino no qual uma resposta a uma mensagem de solicitação deve ser enviada, um aplicativo chama o método Set JMSReplyTo do objeto de mensagem que representa a mensagem de solicitação. O destino especificado na chamada pode ser um destino temporário.

Embora uma sessão seja usada para criar um destino temporário, o escopo de um destino temporário é, na verdade, a conexão que foi usada para criar a sessão. Qualquer uma das sessões da conexão pode criar produtores de mensagens e consumidores de mensagens para o destino temporário. O destino temporário permanece até que seja explicitamente excluído ou que a conexão termine, o que ocorrer primeiro.

Quando um aplicativo cria uma fila temporária, uma fila é criada no servidor de sistema de mensagens para o qual o aplicativo está conectado. Se o aplicativo estiver conectado a um gerenciador de filas, uma fila dinâmica será criada a partir da fila modelo cujo nome é especificado pela propriedade XMSC_WMQ_TEMPORARY_MODEL e o prefixo usado para formar o nome da fila dinâmica será especificado pela propriedade XMSC_WMQ_TEMP_Q_PREFIX . Se o aplicativo estiver conectado a um barramento de integração de serviços, uma fila temporária será criada no barramento e o prefixo usado para formar o nome da fila temporária será especificado pela propriedade XMSC_WPM_TEMP_Q_PREFIX .

Quando um aplicativo que está conectado a um barramento de integração de serviços cria um tópico temporário, o prefixo usado para formar o nome do tópico temporário é especificado pela propriedade XMSC_WPM_TEMP_TOPIC_PREFIX .

Conceitos relacionados

Identificadores de recursos uniformes do tópico
O URI (Identificador Uniforme de Recursos (URI) do tópico especifica o nome do tópico; ele também pode especificar uma ou mais propriedades para ele.

Identificadores uniformes de recursos da fila
O URI para uma fila especifica o nome da fila; ele também pode especificar uma ou mais propriedades da fila.

Produtores de mensagens

Em XMS, um produtor de mensagens pode ser criado com um destino válido ou sem destino associado. Ao criar um produtor de mensagem com um destino nulo, um destino válido precisa ser especificado ao enviar uma mensagem.

Produtores de mensagens com destino associado

Nesse cenário, o produtor da mensagem é criado usando um destino válido. Durante a operação de envio, o destino não precisa ser especificado.

Produtores de mensagens sem destino associado

Em XMS.NET, um produtor de mensagens pode ser criado com um destino nulo.

Para criar um produtor de mensagem sem destino associado ao usar a API .NET , NULL deve ser transmitido como um parâmetro no método `CreateProducer()` do objeto ISession (por exemplo, `session.CreateProducer(null)`). No entanto, um destino válido deve ser especificado quando a mensagem for enviada

Consumidores de mensagens

Os consumidores de mensagens podem ser classificados como assinantes duráveis e não duráveis e consumidores de mensagens síncrona e assíncrona.

Assinantes duráveis

Um assinante durável é um consumidor de mensagens que recebe todas as mensagens publicadas em um tópico, incluindo mensagens publicadas enquanto o assinante está inativo.

Atenção: Essas informações serão relevantes apenas se um aplicativo se conectar a um gerenciador de filas do IBM MQ ou a um barramento de integração de serviço do WebSphere Application Server As informações não são relevantes para uma conexão em tempo real com um broker.

Para criar um assinante durável para um tópico, um aplicativo chama o método Criar Assinante Durável de um objeto de Sessão, especificando como parâmetros um nome que identifica a assinatura durável e um objeto de Destino que representa o tópico. O aplicativo pode criar um assinante durável com ou sem um seletor de mensagem, e pode especificar se o assinante durável deve receber mensagens publicadas por sua própria conexão.

A sessão usada para criar um assinante durável deve ter um identificador de cliente associado. O identificador de cliente é o mesmo que aquele associado à conexão que é usada para criar a sessão; ela é especificada conforme descrito em “ConnectionFactories e objetos de Conexão” na página 647.

O nome que identifica a assinatura durável deve ser exclusivo dentro do identificador de cliente e, portanto, o identificador de cliente faz parte do identificador completo e exclusivo da assinatura durável. O servidor de sistema de mensagens mantém um registro da assinatura durável e assegura que todas as

mensagens publicadas no tópico sejam retidas até que sejam reconhecidas pelo assinante durável ou que elas expirem.

O servidor de sistema de mensagens continua a manter o registro da assinatura durável mesmo depois que o assinante durável for fechado. Para reutilizar uma assinatura durável que foi criada anteriormente, um aplicativo deve criar um assinante durável especificando o mesmo nome de assinatura e usando uma sessão com o mesmo identificador de cliente, como aqueles associados à assinatura durável. Apenas uma sessão de cada vez pode ter um assinante permanente para uma assinatura durável específica.

O escopo de uma assinatura durável é o servidor de sistema de mensagens que está mantendo um registro da assinatura. Se dois aplicativos conectados a diferentes servidores de sistema de mensagens criarem um assinante durável usando o mesmo nome de assinatura e o identificador de cliente, duas assinaturas duráveis completamente independentes serão criadas.

Para excluir uma assinatura durável, um aplicativo chama o método Unsubscribe de um objeto Session, especificando como um parâmetro o nome que identifica a assinatura durável. O identificador de cliente associado à sessão deve ser o mesmo que aquele associado à assinatura durável. O servidor de mensagens exclui o registro da assinatura durável que ele está mantendo e não envia mais mensagens para o assinante durável.

Para alterar uma assinatura existente, um aplicativo pode criar um assinante durável usando o mesmo nome de assinatura e identificador de cliente, mas especificando um tópico diferente ou seletor de mensagem (ou ambos). Alterar uma assinatura durável é equivalente a excluir a assinatura e criar uma nova.

Para um aplicativo que se conecta a um gerenciador de filas do IBM MQ , o XMS gerencia as filas de assinantes. Portanto, o aplicativo não é necessário para especificar uma fila de assinantes. O XMS ignorará a fila de assinantes, se especificada.

Observe que não é possível alterar a fila de assinantes para uma assinatura durável. A única maneira de alterar a fila de assinantes é excluir a assinatura e criar uma nova.

Para um aplicativo que se conecta a um barramento de integração de serviços, cada assinante durável deve ter um home de assinatura durável designado. Para especificar o início da assinatura durável para todos os assinantes permanentes que usam a mesma conexão, configure a propriedade XMSC_WPM_DUR_SUB_HOME do objeto ConnectionFactory usado para criar a conexão. Para especificar o início da assinatura durável para um tópico individual, configure a propriedade XMSC_WPM_DUR_SUB_HOME do objeto Destination que representa o tópico. Um lar de assinaturas duráveis deve ser especificado para uma conexão antes que um aplicativo possa criar um assinante durável que use a conexão. Qualquer valor especificado para um destino substitui o valor especificado para a conexão.

Consumidores de mensagens síncrona

O consumidor de mensagens síncronas recebe as mensagens de uma fila de forma síncrona e recebe uma mensagem por vez. Quando o método `Receive(wait interval)` é usado; a chamada aguarda apenas um período de tempo especificado em milissegundos para uma mensagem ou até o consumidor de mensagens ser fechado.

Se o método ReceiveNoWait () for usado, o consumidor de mensagens síncrona receberá mensagens sem nenhum atraso; se a próxima mensagem estiver disponível, ela será recebida imediatamente, caso contrário, um ponteiro para um objeto de Mensagem nulo será retornado.

Consumidores de mensagens assíncronas

O consumidor de mensagens assíncronas recebe uma mensagem de uma fila de forma assíncrona O listener de mensagem registrado pelo aplicativo é chamado sempre que uma nova mensagem está disponível na fila.

Mensagens suspeitas no XMS

Uma mensagem suspeita é aquela que não pode ser processada por um aplicativo MDB de recebimento. Se uma mensagem suspeita for encontrada, o objeto XMS MessageConsumer poderá reenfileirá-lo de acordo com duas propriedades da fila, **BOQUEUE**, e **BOTHRESH**,

Em algumas circunstâncias, uma mensagem entregue a um MDB pode ser retrocedida em uma fila do IBM MQ. Isso pode acontecer, por exemplo, quando uma mensagem é entregue dentro de uma unidade de trabalho que é, subsequentemente, retrocedida. Uma mensagem que é retrocedida é geralmente entregue novamente, mas uma mensagem mal formatada pode repetidamente fazer com que um MDB falhe e, portanto, não pode ser entregue. Essa mensagem é chamada de uma mensagem suspeita. É possível configurar IBM MQ para que a mensagem suspeita seja automaticamente transferida para outra fila para investigação adicional ou seja descartada. Para obter informações sobre como configurar o IBM MQ dessa maneira, consulte “Manipulando mensagens suspeitas no ASF” na página 659.

Às vezes, uma mensagem mal formatada incorretamente chega em uma fila. Nesse contexto, mal formatada significa que o aplicativo de recebimento não pode processar a mensagem corretamente. Essa mensagem pode fazer com que o aplicativo de recebimento falhe e restaure essa mensagem mal formatada. A mensagem pode então ser entregue repetidamente à fila de entrada e recuperada repetidamente pelo aplicativo. Essas mensagens são conhecidas como mensagens suspeitas. O objeto XMS MessageConsumer detecta mensagens suspeitas e roteia-as para um destino alternativo.

O gerenciador de filas IBM MQ mantém um registro do número de vezes que cada mensagem foi restaurada. Quando esse número atinge um valor limite configurável, o consumidor de mensagem recoloca a mensagem em uma fila de restauração denominada. Se esse novo enfileiramento falhar por qualquer razão, a mensagem será removida da fila de entrada e um enfileirada novamente na fila de mensagens não entregues ou descartada.

Os objetos de ConnectionConsumer XMS manipulam mensagens suspeitas da mesma maneira e usando as mesmas propriedades de fila. Se diversos consumidores de conexão estiverem monitorando a mesma fila, é possível que a mensagem suspeita possa ser entregue a um aplicativo mais vezes do que o valor limite antes que o novo enfileiramento ocorra. Este comportamento ocorre devido à maneira como consumidores de conexões individuais monitoram filas e enfileiram mensagens suspeitas novamente.

O valor do limite e o nome da fila de restauração são atributos de uma fila do IBM MQ. Os nomes dos atributos são **BackoutThreshold** e **BackoutRequeueQName**.. A fila à qual eles se aplicam é a seguinte:

- Para o sistema de mensagens ponto a ponto, é a fila local subjacente. Isso é importante quando os consumidores de mensagens e os consumidores de conexão usam aliases de filas.
- Para o sistema de mensagens de publicação/assinatura no modo normal do provedor do sistemas de mensagens do IBM MQ, ela é a fila modelo por meio da qual a fila gerenciada do tópico é criada.
- Para o sistema de mensagens de publicar/assinar no modo de migração do provedor do sistema de mensagens do IBM MQ, é a fila CCSUB definida no objeto TopicConnectionFactory ou a fila CCDSUB definida no objeto Topic.

Para configurar os atributos **BackoutThreshold** e **BackoutRequeueQName** , emita o comando MQSC a seguir:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Para o sistema de mensagens de publicação / assinatura, se o seu sistema criar uma fila dinâmica para cada assinatura, esses valores de atributo serão obtidos da fila modelo IBM MQ classes for JMS , SYSTEM.JMS.MODEL.QUEUE. Para alterar essas configurações, use:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Se o valor do limite de restauração for zero, a manipulação de mensagens suspeitas será desativada e as mensagens suspeitas permanecerão na fila de entrada. Caso contrário, quando a contagem de restaurações atingir o valor do limite, a mensagem será enviada para a fila de restauração denominada.

Se a contagem de restauração atingir o valor do limite, mas a mensagem não puder ir para a fila de restauração, a mensagem será enviada para a fila de mensagens não entregues ou, se a mensagem for não persistente, ela será descartada.

Essa situação ocorre se a fila de restauração não estiver definida ou se o objeto MessageConsumer não puder enviar a mensagem para a fila de restauração.

Configurando o seu sistema para executar a manipulação de mensagens suspeitas

A fila que o XMS .NET usa ao consultar os atributos **BOTHRESH** e **BOQNAME** depende do estilo de sistema de mensagens que está sendo executado:

- Para o sistema de mensagens ponto a ponto, é a fila local subjacente. Isso é importante quando um aplicativo XMS .NET está consumindo mensagens de filas de alias ou de filas de clusters.
- Para o sistema de mensagens de publicação/assinatura, uma fila gerenciada é criada para conter as mensagens para um aplicativo. O XMS .NET consulta a fila gerenciada para determinar os valores para os atributos **BOTHRESH** e **BOQNAME**.

  A fila gerenciada é criada por meio de uma fila modelo associada ao objeto Tópico que o aplicativo assinou e herda os valores dos atributos **BOTHRESH** e **BOQNAME** da fila modelo. A fila modelo que é usada depende de o aplicativo de recebimento ter obtido uma assinatura durável ou não durável:

  – A fila modelo usada para assinaturas duráveis é especificada pelo atributo **MDURMDL** do Tópico. O valor padrão desse atributo é `SYSTEM.DURABLE.MODEL.QUEUE`.

  – Para assinaturas não duráveis, a fila modelo que é usada é especificada pelo atributo **MNDURMDL**. O valor padrão do atributo **MNDURMDL** é `SYSTEM.NDURABLE.MODEL.QUEUE`.

Ao consultar os atributos **BOTHRESH** e **BOQNAME**, o XMS .NET:

- Abre a fila local ou a fila de destino para uma fila de alias.
- Consulta os atributos **BOTHRESH** e **BOQNAME**.
- Fecha a fila local ou a fila de destino para uma fila de alias.

As opções de abertura usadas ao abrir uma fila local ou a fila de destino para uma fila de alias, dependem da versão do IBM MQ que está sendo usada:

- Para o IBM MQ 9.1.0 Fix Pack 4 Long Term Support e anteriores e o IBM MQ 9.1.4 Continuous Delivery e anteriores: se a fila local ou a fila de destino para uma fila de alias for uma fila de clusters, o XMS .NET a abrirá com as opções `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` e `MQOO_FAIL_IF_QUIESCING`. Isso significa que o usuário que estiver executando o aplicativo de recebimento deverá ter acesso de consulta e de obtenção à instância local da fila de clusters.

  O XMS .NET abre todos os outros tipos de fila local com as opções de abertura `MQOO_INQUIRE` e `MQOO_FAIL_IF_QUIESCING`. Para que o XMS .NET consulte os valores dos atributos, o usuário que executa o aplicativo de recebimento deve ter acesso de consulta na fila local.

Para mover mensagens suspeitas para uma fila de reenfileiramento de restauração ou para a fila de mensagens não entregues do gerenciador de filas, você deve conceder ao usuário que está executando as autoridades `put` e `passall` do aplicativo.

Manipulando mensagens suspeitas no ASF

Ao usar o Application Server Facilities (ASF), o ConnectionConsumer, em vez de o MessageConsumer, processa mensagens suspeitas. O ConnectionConsumer enfileirou novamente as mensagens de acordo com as propriedades **BackoutThreshold** e **BackoutRequeueQName** da fila

Quando um aplicativo usa ConnectionConsumers, as circunstâncias nas quais uma mensagem é restaurada dependem da sessão que o servidor de aplicativos fornece:

- Quando a sessão é não transacionada, com AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, uma mensagem é restaurada somente após um erro do sistema ou se o aplicativo for finalizado inesperadamente.

- Quando a sessão não é transacionada com CLIENT_RECONHEÇO, mensagens não reconhecidas podem ser restauradas pelo servidor de aplicativos que chama `Session.recover()`.

  Geralmente, a implementação do cliente de MessageListener ou as chamadas do servidor de aplicativos `Message.acknowledge()` `Message.acknowledge()` reconhece todas as mensagens entregues na sessão até agora.
- Quando a sessão é transacionada, mensagens não reconhecidas podem ser restauradas pelo servidor de aplicativos chamando `Session.rollback()`.

Navegadores de fila

Um aplicativo usa um navegador de filas para pesquisar mensagens em uma fila sem removê-las.

Para criar um navegador de filas, um aplicativo chama o método Criar Navegador de Filas de um objeto de ISession, especificando como um parâmetro um objeto de Destino que identifica a fila a ser procurada. O aplicativo pode criar um navegador de filas com ou sem um seletor de mensagem.

Depois de criar um navegador de filas, o aplicativo pode chamar o método GetEnumerator do objeto IQueueBrowser para obter uma lista de mensagens na fila. O método retorna um enumerador que encapsula uma lista de objetos de Mensagem. A ordem dos objetos de Mensagem na lista é a mesma que a ordem em que as mensagens seriam recuperadas da fila. O aplicativo pode então usar o enumerador para pesquisar cada mensagem por vez.

O enumerador é atualizado dinamicamente conforme as mensagens são colocadas na fila e removidas da fila. Cada vez que o aplicativo chama IEnumerator.MoveNext () para navegar na próxima mensagem na fila, a mensagem reflete os conteúdos atuais da fila.

Um aplicativo pode chamar o método GetEnumerator mais de uma vez para um navegador de filas fornecido. Cada chamada retorna um novo enumerador. O aplicativo pode, portanto, usar mais de um enumerador para procurar as mensagens em uma fila e manter várias posições dentro da fila.

Um aplicativo pode usar um navegador de filas para procurar por uma mensagem adequada para remover de uma fila e, em seguida, usar um consumidor de mensagens com um seletor de mensagem para remover a mensagem. O seletor de mensagem pode selecionar a mensagem de acordo com o valor do campo de cabeçalho JMSMessageID. Para obter informações sobre esse e outros campos de cabeçalho da mensagem JMS, consulte “Campos de cabeçalho em uma mensagem XMS” na página 679.

Solicitantes

Um aplicativo usa um solicitante para enviar uma mensagem de solicitação e, em seguida, esperar e receber a resposta.

Muitos aplicativos do sistema de mensagens são baseados em algoritmos que enviam uma mensagem de solicitação e, em seguida, aguardam uma resposta. XMS fornece uma classe chamada Requestor para ajudar com o desenvolvimento desse estilo de aplicativo.

Para criar um solicitante, um aplicativo chama o construtor Create Requestor da classe Requestor, especificando como parâmetros um objeto Session e um objeto de Destino que identifica onde as mensagens de solicitação devem ser enviadas. A sessão não deve ser transacionada nem ter um modo de confirmação de XMSC_CLIENT_ACKNOWLEDGE. O construtor cria automaticamente uma fila ou um tópico provisório para os quais as mensagens de resposta devem ser enviadas.

Depois de criar um solicitante, o aplicativo pode chamar o método Request do objeto Solicitante para enviar uma mensagem de solicitação e, em seguida, aguardar e receber uma resposta do aplicativo que recebe a mensagem de solicitação. A chamada aguarda até que a resposta seja recebida ou até que a sessão termine, o que ocorrer primeiro. Somente uma resposta é requerida pelo solicitante para cada mensagem de solicitação.

Quando o aplicativo fecha o solicitante, a fila temporária ou o tópico é excluído. A sessão associada, no entanto, não fecha.

Exclusão de objeto

Quando um aplicativo exclui um objeto XMS que ele criou, XMS libera os recursos internos que foram alocados para o objeto.

Quando um aplicativo cria um objeto do XMS, o XMS aloca memória e outros recursos internos para o objeto. XMS retém esses recursos internos até que o aplicativo exclua explicitamente o objeto chamando o método close ou delete do objeto, no ponto em que o XMS libera os recursos internos. Se um aplicativo tentar excluir um objeto que já está excluído, a chamada será ignorada.

Quando um aplicativo exclui um objeto Connection ou Session, o XMS exclui determinados objetos associados automaticamente e libera seus recursos internos. Esses são objetos que foram criados pelo objeto Connection ou Session e não têm função independente do objeto. Esses objetos são mostrados em Tabela 83 na página 661.

Nota: Se um aplicativo fechar uma conexão com sessões dependentes, todos os objetos dependentes dessas sessões também serão excluídos. Apenas um objeto Connection ou Session pode ter objetos dependentes.

Tabela 83. Objetos que são excluídos automaticamente

Objeto excluído	Método	Objetos Dependentes que São Excluídos Automaticamente
Conexão	Encerrar Conexão	Objetos ConnectionMetaData e Session
Sessão	Fechar Sessão	Objetos MessageConsumer, MessageProducer, QueueBrowser e Requestor

Tipos de dados para XMS.NET

O XMS.NET suporta System.Boolean, System.Byte, System.SByte, System.Char, System.String, System.Single, System.Double, System.Decimal, System.Int32, System.Int64, System.UInt16, System.UInt32, System.UInt64 e System.Object. Os tipos de dados para XMS .NET são diferentes dos tipos de dados para C/C++ XMS. Você pode utilizar este tópico para identificar os tipos de dados correspondentes.

A tabela a seguir mostra os tipos de dados XMS .NET e XMS C/C++ correspondentes e os descreve resumidamente.

Tabela 84. Tipos de dados para XMS .NET e C/C++ XMS

Tipo de XMS .NET	XMS tipo de C/C++	Descrição
System.SByte	xmsSBYTE xmsINT8	Valor de 8 bits sinalizado
System.Byte	xmsBYTE xmsUINT8	Valor de 8 bits não assinado
System.Int16	xmsINT16 xmsSHORT	Valor de 16 bits sinalizado
System.UInt16	xmsUINT16 xmsUSHORT	Valor de 16 bits não assinado
System.Int32	xmsINT32 xmsINT	Valor de 32 bits assinado

Tabela 84. Tipos de dados para XMS .NET e C/C++ XMS (continuação)		
Tipo de XMS .NET	**XMS tipo de C/C++**	**Descrição**
System.UInt32	xmsUINT32 xmsUINT	Valor de 32 bits não assinado
System.Int64	xmsLONG xmsINT64	Valor de 64 Bits Assinado
System.UInt64	xmsULONG xmsUINT64	Valor de 64 bits não assinado
System.Char	xmsCHAR16	Caractere de 16 bits não assinado (Unicode para .NET)
System.Single	xmsFLOAT	IEEE 32-bit float
System.Double	xmsDOUBLE	IEEE 64-bit float
System.Boolean	xmsBOOL	Um valor True / False
Não-aplicável	xmsCHAR	Valor de 8-bit assinado ou não assinado (depende da plataforma)
System.Decimal	Não-aplicável	Número inteiro sinalizado de 96 bits 10^0 a 10^{28}
System.Object	Não-aplicável	Base de todos os tipos
System.String	Não-aplicável	Tipo de sequência

Tipos primitivos do XMS

O XMS fornece equivalentes dos oito tipos primitivos Java (byte, short, int, long, float, double, char e boolean). Isso permite a troca de mensagens entre o XMS e o JMS sem que os dados se tornem perdidos ou corrompidos.

Tabela 85 na página 662 lista o tipo de dados equivalentes Java , o tamanho e o valor mínimo e máximo de cada tipo primitivo XMS .

Tabela 85. Tipos de Dados XMS e seus Equivalentes Java				
Tipo de Dados XMS	**Tipo de DadosJava Compatível**	**Tamanho**	**Valor Mínimo**	**Valor máximo**
System.Boolean	booleano	32 bits	false	true
System.SBYTE	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.BYTE	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.CHAR	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.Int16	short	16 bits	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 bits	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	long	64 bits	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	float	32 bits	-3.402823E-38 (para precisão de 7 dígitos)	3.402823E + 38 (para precisão de 7 dígitos)

Tabela 85. Tipos de Dados XMS e seus Equivalentes Java (continuação)				
Tipo de Dados XMS	**Tipo de DadosJava Compatível**	**Tamanho**	**Valor Mínimo**	**Valor máximo**
System.Double	double	64 bits	-1.79769313486231E-308 (para 15-dígitos de precisão)	1.79769313486231E + 308 (para precisão de 15 dígitos)

Conversão implícita de um valor de propriedade de um tipo de dados para outro

Quando um aplicativo obtém o valor de uma propriedade, o valor pode ser convertido por XMS em outro tipo de dados. Muitas regras controlam quais conversões são suportadas e como o XMS executa as conversões.

Uma propriedade de um objeto tem um nome e um valor; o valor tem um tipo de dados associado, em que o valor de uma propriedade também é chamado de *tipo de propriedade*.

Um aplicativo usa os métodos da classe PropertyContext para obter e configurar as propriedades de objetos. Para obter o valor de uma propriedade, um aplicativo chama o método que é apropriado para o tipo de propriedade. Por exemplo, para obter o valor de uma propriedade de número inteiro, um aplicativo geralmente chama o método GetIntProperty.

No entanto, quando um aplicativo obtém o valor de uma propriedade, o valor pode ser convertido por XMS em outro tipo de dados. Por exemplo, para obter o valor de uma propriedade de número inteiro, um aplicativo pode chamar o método GetStringProperty, que retorna o valor da propriedade como uma sequência. As conversões suportadas pelo XMS são mostradas em Tabela 86 na página 663.

Tabela 86. Conversões Suportadas de um Tipo de Propriedade para Outros Tipos de Dados	
Tipo de propriedade	**Tipos de dados de destino suportados**
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
Matriz System.SByte	System.String
System.Int16	System.String, System.Int32, System.Int64

As regras gerais a seguir controlam as conversões suportadas:

- Os valores de propriedade numérica podem ser convertidos de um tipo de dados para outro, desde que nenhum dado seja perdido durante a conversão. Por exemplo, o valor de uma propriedade com o tipo de dados System.Int32 pode ser convertido em um valor com o tipo de dados System.Int64, mas não pode ser convertido em um valor com tipo de dados System.Int16.
- Um valor de propriedade de qualquer tipo de dados pode ser convertido em uma sequência.
- Um valor de propriedade de sequência pode ser convertido para qualquer outro tipo de dados, desde que a sequência seja formatada corretamente para a conversão. Se um aplicativo tentar converter um valor de propriedade de sequência que não está formatado corretamente, XMS poderá retornar erros.

- Se um aplicativo tentar uma conversão não suportada, o XMS poderá retornar um erro.

As regras a seguir se aplicam quando um valor de propriedade é convertido de um tipo de dados para outro:

- Ao converter um valor de propriedade booleana para uma sequência, o valor true é convertido para a sequência "true"e o valor false é convertido para a sequência "false".
- Ao converter um valor de propriedade booleana para um tipo de dados numérico, incluindo System.SByte, o valor true será convertido para 1 e o valor false será convertido para 0.
- Ao converter um valor de propriedade de sequência para um valor booleano, a sequência "true" (não faz distinção entre maiúsculas e minúsculas) ou "1" é convertida para true e a sequência "false" (sem distinção entre maiúsculas e minúsculas) ou "0" é convertida para false. Todas as outras sequências não podem ser convertidas.
- Ao converter um valor de propriedade de sequência para um valor com tipo de dados System.Int32, System.Int64, System.SByte ou System.Int16, a sequência deve ter o formato a seguir:

  [*em branco*][*assinar*]*digitos*

  Os componentes de sequência são definidos da seguinte forma:

  espaços em branco
  Caracteres em branco à esquerda opcionais.

  sinal
  Um caractere de sinal de mais (+) ou de sinal de menos (-) opcional.

  dígitos
  Uma sequência contígua de caracteres de dígito (0-9). Pelo menos um caractere de dígito deve estar presente.

  Após a sequência de caracteres de dígito, a sequência pode conter outros caracteres que não são caracteres de dígito, mas a conversão para assim que o primeiro desses caracteres for atingido. A sequência é assumida para representar um número inteiro decimal.

  XMS pode retornar um erro se a sequência não estiver formatada corretamente.
- Ao converter um valor de propriedade de sequência para um valor com tipo de dados System.Double ou System.Float, a sequência deve ter o formato a seguir:

  [*blanks*] [*sign*] [*digits*] [*point*[*d_digits*]] [*e_char*[*e_sign*]*e_digits*]

  Os componentes de sequência são definidos da seguinte forma:

  espaços em branco
  (Opcional) Leitura de caracteres em branco.

  sinal
  (Opcional) Sinal de mais (+) ou sinal de menos (-).

  dígitos
  Uma sequência contígua de caracteres de dígito (0-9). Pelo menos um caractere de dígito deve estar presente em *digits* ou *d_digits*.

  ponto
  (Opcional) Separador decimal (.).

  d_digits
  Uma sequência contígua de caracteres de dígito (0-9). Pelo menos um caractere de dígito deve estar presente em *digits* ou *d_digits*.

  e_char
  Um caractere expoente, que é um *E* ou *e*.

  e_sign
  (Opcional) Sinal de mais (+) ou sinal de menos (-) para o expoente.

e_digits
Uma sequência contígua de caracteres de dígito (0-9) para o expoente. Pelo menos um caractere de dígito deve estar presente se a sequência contiver um caractere exponente.

Após a sequência de caracteres de dígito ou os caracteres opcionais que representam um expoente, a sequência pode conter outros caracteres que não são caracteres de dígito, mas a conversão para assim que o primeiro desses caracteres for atingido. Supõe-se que a sequência represente um número de vírgula flutuante decimal com um expoente que é uma potência de 10.

XMS pode retornar um erro se a sequência não estiver formatada corretamente.

- Ao converter um valor de propriedade numérico em uma sequência, incluindo um valor de propriedade com tipo de dados System.SByte, o valor é convertido para a representação de sequência do valor como um número decimal, não a sequência que contém o caractere ASCII para esse valor. Por exemplo, o número inteiro 65 será convertido para a sequência "65", não para a sequência "A".
- Ao converter um valor de propriedade matriz de bytes em uma sequência, cada byte é convertido nos 2 caracteres hexadecimais que representam o byte. Por exemplo, a matriz de bytes {0}xF1, 0x12, 0x00, 0xFF } é convertida para a sequência "F11200FF".

Conversões de um tipo de propriedade para outros tipos de dados são suportadas pelos métodos de ambas as classes Property e PropertyContext.

Iteradores

Um agente iterativo encapsula uma lista de objetos e um cursor que mantém a posição atual na lista. O conceito de um Agente Iterativo, como disponível em IBM MQ Message Service Client (XMS) for C/C++, é implementado usando a interface IEnumerator em IBM MQ Message Service Client (XMS) for .NET.

Quando um agente iterativo é criado, a posição do cursor é anterior ao primeiro objeto. Um aplicativo usa um agente iterativo para recuperar cada objeto por sua vez.

A classe Iterator de IBM MQ Message Service Client (XMS) for C/C++ é equivalente à classe Enumerator em Java. O IBM MQ Message Service Client (XMS) for .NET é semelhante ao Java e usa uma interface IEnumerator.

Um aplicativo pode usar um IEnumerator para executar as tarefas a seguir:

- Para obter as propriedades de uma mensagem
- Para obter os pares nome-valor no corpo de uma mensagem de mapa
- Para procurar as mensagens em uma fila
- Para obter os nomes das propriedades de mensagem definidas pelo JMS suportadas por uma conexão

Manipulação de erros no XMS .NET

XMS .NET exceções são todas derivadas de System.Exception..

XMS .NETExceções

Chamadas de método XMS podem emitir exceções XMS específicas, como MessageFormatException, General XMSExceptions ou exceções do sistema, como NullReferenceException.

Grave aplicativos para capturar qualquer um desses erros, seja em blocos de captura específicos ou em blocos de captura `System.Exception` em geral, conforme apropriado para os requisitos do aplicativo.

Códigos de erro e de exceção do XMS

XMS usa um intervalo de códigos de erro para indicar falhas. Esses códigos de erro não são explicitamente listados nesta documentação porque eles podem variar de liberação para liberação. Apenas códigos de exceção XMS (no formato XMS_X_...) são documentados porque permanecem os mesmos em diferentes versões de XMS.

Informações relacionadas

Classe MQException.NET
Códigos de erro SSL comuns lançados pelas bibliotecas do cliente XMS .NET
Configuração do FFDC para aplicativos XMS .NET
Rastreando aplicativos do XMS .NET

Usando listeners de mensagem e de exceção em .NET

Um aplicativo .NET usa um listener de mensagem para receber mensagens de forma assíncrona, e ele usa um listener de exceção para ser notificado de forma assíncrona de um problema com uma conexão.

Sobre esta tarefa

A funcionalidade de ambos os listeners de mensagem e exceção é a mesma para .NET e para C++. No entanto, existem algumas pequenas diferenças de implementação.

Procedimento

- Para configurar um listener de mensagem para receber mensagens de forma assíncrona, conclua as etapas a seguir:
  a) Defina um método que corresponda à assinatura do delegado de listener de mensagem.
  O método que você define pode ser um método estático ou de instância e pode ser definido em qualquer classe acessível. A assinatura do delegado é a seguinte:

```
public delegate void MessageListener(IMessage msg);
```

  e, portanto, você poderia definir o método como:

```
void SomeMethodName(IMessage msg);
```

  b) Instancie este método como um delegado usando algo semelhante ao exemplo a seguir:

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

  c) Registre o delegado com um ou mais consumidores, configurando-o para a propriedade MessageListener do consumidor:

```
consumer.MessageListener = OnMsgMethod;
```

  É possível remover o delegado, reconfigurando o MessageListener para nulo:

```
consumer.MessageListener = null;
```

- Para configurar um listener de exceção, conclua as etapas a seguir.
  O listener de exceção funciona da mesma maneira que o listener de mensagem, mas possui uma definição de delegação diferente e é designado para a conexão, em vez disso, o consumidor de mensagens. Isso é o mesmo que para C + +.
  a) Defina o método.
  A assinatura do delegado é a seguinte:

```
public delegate void ExceptionListener(Exception ex);
```

  e, portanto, o método definido poderia ser:

```
void SomeMethodName(Exception ex);
```

b) Instancie este método como um delegado usando algo semelhante ao exemplo a seguir:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

c) Registre o delegado com a conexão configurando sua propriedade ExceptionListener:

```
connection.ExceptionListener = OnExMethod ;
```

É possível remover o delegado, reconfigurando o ExceptionListener para:

```
null:  connection.ExceptionListener = null;
```

IBM MQ Reconexão de cliente automática por meio de XMS

Configure o cliente XMS para reconectar automaticamente após uma falha de rede, gerenciador de filas ou servidor ao usar o cliente IBM WebSphere MQ 7.1 e mais recente, como o provedor de mensagens.

Use as propriedades `WMQ_CONNECTION_NAME_LIST` e `WMQ_CLIENT_RECONNECT_OPTIONS` da classe `MQConnectionFactory` para configurar uma conexão do cliente para reconectar automaticamente. A reconexão de cliente automática reconecta um cliente após uma falha de conexão ou como uma opção após parar o gerenciador de filas. O design de alguns aplicativos clientes os torna inadequados para reconexão automática.

As conexões do cliente reconectáveis automaticamente tornam-se reconectáveis depois que a conexão for estabelecida.

Nota: As propriedades `Opções de Reconexão do Cliente`, `Tempo Limite de Reconexão do Cliente` e `Lista de Nomes de Conexão` também podem ser definidas por meio da Tabela de Definição de Canal do Cliente (CCDT), ou ativando a reconexão do cliente por meio do arquivo `mqclient.ini`.

Nota: Se as propriedades de reconexão forem configuradas no objeto `ConnectionFactory` e, assim como na CCDT, a regra de precedência será a seguinte. Se o valor padrão da propriedade da lista de nomes de conexão for configurado no objeto `ConnectionFactory` , então, a CCDT terá precedência. Se a lista de nomes de conexão não estiver configurada para seu valor padrão, os valores da propriedade definidos no objeto `ConnectionFactory` têm precedência. O valor padrão da lista de nomes de conexão é `localhost(1414)`.

Trabalhando com objetos administrados XMS .NET

Os tópicos nesta seção fornecem informações sobre objetos administrados. Os aplicativos XMS podem recuperar as definições de objeto de um repositório de objetos administrados centrais e usá-los para criar connection factories e destinos.

Sobre esta tarefa

Esta seção fornece informações para ajudar na criação e gerenciamento de objetos administrados, descrevendo os tipos de repositório de objeto administrado que o XMS suporta. A seção também explica como um aplicativo XMS faz uma conexão com um repositório de objetos administrados para recuperar os objetos administrados necessários.

Esta seção contém os seguintes tópicos:

- “XMS .NET tipos suportados de repositório de objetos administrados” na página 668
- “XMS .NET mapeamento de propriedade para objetos administrados” na página 668
- “XMS .NET propriedades necessárias para objetos ConnectionFactory administrados” na página 671

- "XMS .NET propriedades necessárias para objetos de destino administrados" na página 671
- "XMS .NET criando objetos administrados" na página 672
- "XMS .NET criando objetos InitialContext" na página 673
- "propriedades XMS .NET InitialContext" na página 673
- "Formato de URI para contextos iniciais XMS" na página 673
- "Serviço da web de consulta do JNDI para XMS .NET" na página 674
- "XMS .NET recuperação de objetos administrados" na página 675

XMS .NET tipos suportados de repositório de objetos administrados

Os objetos administrados do Sistema de Arquivos e do LDAP podem ser usados para se conectar ao IBM MQ e ao WebSphere Application Server, enquanto que o COS Naming pode ser usado para se conectar apenas ao WebSphere Application Server.

Diretórios de objetos do Sistema de Arquivos assumem a forma de objetos serializados Java Naming Directory Interface (JNDI). Diretórios de objetos LDAP são diretórios que contêm objetos JNDI . Os diretórios do sistema de arquivos e do objeto LDAP podem ser administrados pelo IBM MQ Explorer, que é fornecido com o IBM MQ e mais recente Ambos os diretórios do sistema de arquivos e do objeto LDAP podem ser usados para administrar conexões do cliente centralizando connection factories e destinos do IBM MQ . O administrador de rede pode implementar vários aplicativos que se referem ao mesmo repositório central e que são automaticamente atualizados para refletir as mudanças nas configurações de conexão feitas no repositório central.

Um CORBA Naming Directory contém fábricas de conexão WebSphere Application Server service integration bus e destinos e pode ser administrado usando o console administrativo WebSphere Application Server. Para que um aplicativo XMS recupere objetos do CORBA Naming Directory, um serviço da web de consulta JNDI deve ser implementado. Esse serviço da web não está disponível em todos os WebSphere Application Server service integration technologies. Consulte a documentação do produto para obter detalhes.

Nota: Reinicie as conexões do aplicativo para que as mudanças no diretório de objeto tenham efeito.

XMS .NET mapeamento de propriedade para objetos administrados

Para permitir que os aplicativos XMS .NET usem IBM MQ JMS e WebSphere Application Server connection factory e definições de objetos de destino, as propriedades recuperadas dessas definições devem ser mapeadas para as propriedades XMS correspondentes que podem ser configuradas em XMS connection factories e destinos.

Para criar, por exemplo, uma connection factory do XMS com propriedades recuperadas de uma connection factory JMS do IBM MQ, as propriedades devem ser mapeadas entre as duas.

Todos os mapeamentos de propriedade são executados automaticamente.

Tabela 87 na página 668 demonstra os mapeamentos entre algumas das propriedades mais comuns de connection factories e destinos. As propriedades mostradas nesta tabela são apenas um pequeno conjunto de exemplos e nem todas elas são relevantes para todos os tipos de conexão e servidores.

Tabela 87. Exemplos de mapeamento de nome para connection factory e propriedades de destino

Nome da propriedade IBM MQ JMS	**Nome da propriedade XMS**	**Nome da propriedade WebSphere Application Server service integration bus**
PERSISTÊNCIA (POR)	XMSC_DELIVERY_MODE	
EXPIRAÇÃO (EXP)	XMSC_TIME_TO_LIVE	
PRIORIDADE (PRI)	XMSC_PRIORITY	

Tabela 87. Exemplos de mapeamento de nome para connection factory e propriedades de destino (continuação)		
Nome da propriedade IBM MQ JMS	**Nome da propriedade XMS**	**Nome da propriedade WebSphere Application Server service integration bus**
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

Nota: As propriedades mostradas em Tabela 88 na página 669 são aplicáveis para JMS assim como XMS .NET.

Tabela 88. XMS .NET propriedades					
Propriedade	**Tipo de objeto**				
	CF	**QCF**	**TCF**	**Fila**	**Tópico**
APPLICATIONNAME	Y	Y	Y	N/D	N/D
ASYNCEXCEPTION	Y	Y	Y	N/D	N/D
CCDTURL	Y	Y	Y	N/D	N/D
CHANNEL	Y	Y	Y	N/D	N/D
CONNECTIONNAMELIST	Y	Y	Y	N/D	N/D
CLIENTRECONNECTOPTIONS	Y	Y	Y	N/D	N/D
CLIENTRECONNECTTIMEOUT	Y	Y	Y	N/D	N/D
CLIENTID	N/D	Y	N/D	N/D	N/D
COMPHDR "1" na página 670	Y	N/D	Y	N/D	N/D
COMPMSG "1" na página 670	Y	Y	Y	N/D	N/D
CONNOPT "1" na página 670	Y	Y	Y	N/D	N/D
CONNTAG "1" na página 670	Y	Y	Y	N/D	N/D
DESCRIPTION "1" na página 670	N/D	Y	N/D	Y	Y
Expiração "1" na página 670	N/D	N/D	N/D	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
HOSTNAME	N/D	Y	N/D	N/D	N/D
LOCALADDRESS	N/D	Y	N/D	N/D	N/D

Tabela 88. XMS .NET propriedades (continuação)					
Propriedade	**Tipo de objeto**				
	CF	**QCF**	**TCF**	**Fila**	**Tópico**
PERSISTENCE	N/D	N/D	N/D	Y	Y
PORT	N/D	Y	N/D	N/D	N/D
Prioridade "1" na página 670	N/D	N/D	N/D	Y	Y
PROVIDERVERSION "1" na página 670	N/D	Y	N/D	N/D	N/D
QMANAGER	Y	Y	Y	Y	N/D
Fila "1" na página 670	N/D	N/D	N/D	Y	N/D
SHARECONVALLOWED	Y	Y	Y	N/D	N/D
tópico"1" na página 670	N/D	N/D	N/D	N/D	Y
Transporte"1" na página 670	N/D	Y	N/D	N/D	N/D

Nota:

1. Essas propriedades não possuem propriedades de nível de aplicativo mas opcionalmente podem ser definidas usando propriedades administradas.

Propriedade OutboundSNI

Em IBM MQ 9.3.0, é possível configurar a propriedade XMSC_WMQ_OUTBOUND_SNI, que configura a propriedade **OutboundSNI** em um aplicativo.

O XMSC_WMQ_OUTBOUND_SNI_PROPERTY leva os seguintes valores:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL, que mapeia para "CHANNEL"
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME, que mapeia para "HOSTNAME"
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK, que mapeia para "*"

Além disso, é possível configurar a propriedade **OutboundSNI** usando a variável de ambiente MQOUTBOUND_SNI, que obtém os seguintes valores:

- CHANNEL
- HOSTNAME
- *

Nota: O padrão de propriedade para XMSC_WMQ_OUTBOUND_SNI_CHANNEL se nenhum valor específico for definido.

A ordem de precedência para configurar a propriedade **OutboundSNI** no nó gerenciado é:

1. Propriedade do nível do aplicativo
2. Variável de Ambiente

Para a propriedade **OutboundSNI** em nó não gerenciado, somente `mqclient.ini` é suportado.

XMS .NET propriedades necessárias para objetos ConnectionFactory administrados

Quando um aplicativo cria um connection factory, um número de propriedades deve ser definido para criar uma conexão com um servidor de sistema de mensagens.

As propriedades listadas nas tabelas a seguir são o mínimo necessário para que um aplicativo seja configurado para criar uma conexão com um servidor de sistema de mensagens. Se você desejar customizar a maneira que uma conexão é criada, seu aplicativo poderá configurar quaisquer propriedades adicionais do objeto ConnectionFactory, conforme necessário. Para obter mais informações, consulte Propriedades de ConnectionFactory. Uma lista completa de propriedades disponíveis está incluída.

Conexão com um gerenciador de filas do IBM MQ

Tabela 89. Configurações de propriedade para objetos ConnectionFactory administrados para conexões com um gerenciador de filas do IBM MQ

Propriedade XMS necessária	Propriedade equivalente do IBM MQ JMS necessária
XMSC_CONNECTION_TYPE	XMS trabalha isso a partir do nome da classe do connection factory e da propriedade TRANSPORT (TRAN).
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	PORT
XMSC_WMQ_QUEUE_MANAGER	Nome do Gerenciador de Filas

Conexão de tempo real com um broker

Tabela 90. Configurações de propriedade para objetos ConnectionFactory administrados para conexões em tempo real com um broker

Necessário XMS	Propriedade equivalente do IBM MQ JMS necessária
XMSC_CONNECTION_TYPE	XMS trabalha isso a partir do nome da classe do connection factory e da propriedade TRANSPORT (TRAN).
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	PORT

Conexão com um WebSphere Application Server service integration bus

Tabela 91. Configurações de propriedade para objetos ConnectionFactory administrados para conexões com um WebSphere Application Server service integration bus

Propriedade XMS	Descrição
XMSC_CONNECTION_TYPE	O tipo de servidor de mensagens para o qual um aplicativo se conecta.. Isso é determinado a partir do nome da classe do connection factory.
XMSC_WPM_BUS_NAME	Para um connection factory, o nome do barramento de integração de serviços ao qual o aplicativo se conecta ou, para um destino, o nome do barramento de integração de serviços no qual o destino existe.

XMS .NET propriedades necessárias para objetos de destino administrados

Um aplicativo que está criando um destino deve configurar várias propriedades que o aplicativo em um objeto Destination administrado.

Tabela 92. Configurações de propriedade para objetos de Destino administrados		
Tipo de conexão	**Propriedade**	**Descrição**
Gerenciador de filas da IBM MQ	FILA (QU) TÓPICO (SUPERIOR)	A fila à qual você deseja se conectar O tópico que o aplicativo usa como um destino
Conexão de tempo real com um broker	TÓPICO (SUPERIOR)	O tópico que o aplicativo usa como um destino
WebSphere Application Server service integration bus	topicName queueName	Se seu aplicativo estiver se conectando a um tópico Se seu aplicativo estiver se conectando a uma fila

XMS .NET criando objetos administrados

As definições de objeto ConnectionFactory e Destination que os aplicativos XMS requerem para fazer uma conexão com um servidor de sistema de mensagens devem ser criadas usando as ferramentas administrativas apropriadas.

Antes de começar

Para obter detalhes adicionais sobre os diferentes tipos de repositório de objeto administrado que o XMS suporta, consulte "XMS .NET tipos suportados de repositório de objetos administrados" na página 668.

Sobre esta tarefa

Para criar os objetos administrados do IBM MQ, use a ferramenta de administração JMS (JMSAdmin) do IBM MQ Explorer ou IBM MQ.

Para criar os objetos administrados para IBM MQ ou IBM Integration Bus, use a ferramenta IBM MQ de administração JMS (JMSAdmin).

Para criar objetos administrados para WebSphere Application Server service integration bus, use o console administrativo WebSphere Application Server .

Na ferramenta administrativa, a propriedade é conhecida como **APPLICATIONNAME** ou **APPNAME** para abreviação.

Nota: Não é possível usar JMSAdmin para configurar TRANSPORT (UNMANAGED). Portanto, para obter um cliente XMS não gerenciado utilizando um nome de aplicativo escolhido administrativamente, é necessário digitar o seguinte comando:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

As etapas a seguir resumem o que você faz para criar objetos administrados.

Procedimento

1. Crie um connection factory e defina as propriedades necessárias para criar uma conexão de seu aplicativo com o servidor escolhido.

   As propriedades mínimas que o XMS requer para fazer uma conexão são definidas em "XMS .NET propriedades necessárias para objetos ConnectionFactory administrados" na página 671.
2. Crie o destino necessário no servidor de sistema de mensagens, ao qual seu aplicativo se conecta:
   - Para uma conexão com um gerenciador de filas do IBM MQ, crie uma fila ou um tópico.
   - Para uma conexão em tempo real com um broker, crie um tópico.
   - Para uma conexão com um WebSphere Application Server service integration bus, crie uma fila ou um tópico.

As propriedades mínimas que o XMS requer para fazer uma conexão são definidas em "XMS .NET propriedades necessárias para objetos de destino administrados" na página 671.

XMS .NET criando objetos InitialContext

Um aplicativo deve criar um contexto inicial a ser usado para fazer uma conexão com o repositório de objetos administrados para recuperar os objetos administrados necessários.

Sobre esta tarefa

Um objeto InitialContext encapsula uma conexão com o repositório. A API XMS fornece métodos para executar as tarefas a seguir:

- Criar um objeto InitialContext
- Consultar um objeto administrado no repositório de objetos administrado.

Procedimento

- Para obter detalhes adicionais sobre a criação de um objeto InitialContext, consulte InitialContext para .NET e Propriedades de InitialContext.

propriedades XMS .NET InitialContext

Os parâmetros do construtor InitialContext incluem o local do repositório de objetos administrados, fornecido como um indicador de recurso uniforme (URI). Para que um aplicativo estabeleça uma conexão com o repositório, pode ser necessário fornecer mais informações do que as informações contidas no URI.

No JNDI e na implementação .NET de XMS, as informações adicionais são fornecidas em um ambiente Hashtable ao construtor.

O local do repositório de objetos administrado é definido na propriedade XMSC_IC_URL . Normalmente, essa propriedade é transmitida na chamada Criar, mas pode ser modificada para se conectar a um diretório de nomenclatura diferente antes da consulta. Para contextos FileSystem ou LDAP, essa propriedade define o endereço do diretório. Para nomenclatura COS, esse é o endereço do serviço da web que usa essas propriedades para se conectar ao diretório JNDI.

As propriedades a seguir são transmitidas sem modificação para o serviço da web que as usará para usar para se conectar ao diretório JNDI.

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

Formato de URI para contextos iniciais XMS

O local do repositório de objetos administrados é fornecido como um indicador de recurso uniforme (URI). O formato do URI depende do tipo de contexto.

Contexto de FileSystem

Para o contexto FileSystem, a URL fornece o local do diretório baseado no sistema de arquivos. A estrutura da URL é conforme definido pelo RFC 1738, *Localizadores Uniformes de Recursos (URL)*: a URL tem o prefixo `file://` e a sintaxe após esse prefixo é uma definição válida de um arquivo que pode ser aberto no sistema no qual o XMS está executando.

Esta sintaxe pode ser específica da plataforma e pode usar os separadores '/ separadores ou' \ '. Se você usar '\', cada separador precisará ser escapado usando um '\' adicional. Isso evita que a estrutura do .NET tente interpretar o separador como um caractere de escape para o que se segue.

Esses exemplos ilustram esta sintaxe:

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

Contexto LDAP

Para o contexto LDAP, a estrutura básica da URL é conforme definido pelo RFC 2255, *The LDAP URL Format*, com o prefixo sem distinção entre maiúsculas e minúsculas `ldap://`

A sintaxe exata é ilustrada no exemplo a seguir:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Essa sintaxe é definida no RFC, mas sem suporte para quaisquer atributos, escopo, filtros ou extensões.

Exemplos desta sintaxe incluem:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

Contexto de WSS

Para o contexto WSS, a URL está no formato de um terminal de serviços da Web, com o prefixo `http://`.

Como alternativa, é possível usar o prefixo `cosnaming://` ou `wsvc://`,

Essas duas prefixos são interpretadas como significando que você está usando um contexto WSS com a URL acessada através de http, o que permite que o tipo de contexto inicial seja derivado facilmente diretamente da URL.

Exemplos dessa sintaxe incluem o seguinte:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

Serviço da web de consulta do JNDI para XMS .NET

Para acessar um diretório de nomenclatura COS a partir de XMS, um serviço da web JNDI Lookup deve ser implementado em um servidor WebSphere Application Server service integration bus . Esse serviço da web converte as informações de Java do serviço de nomenclatura COS em um formulário que os aplicativos XMS podem ler.

O serviço da Web é fornecido no arquivo de archive corporativo SIBXJndiLookupEAR.ear, localizado dentro do diretório de instalação. Para a liberação atual de IBM MQ Message Service Client (XMS) for .NET, SIBXJndiLookupEAR.ear pode ser localizado no diretório *install_dir*`\java\lib` . Isso pode ser instalado em um servidor WebSphere Application Server service integration bus , usando o console administrativo ou a ferramenta de script wsaadmin. Consulte a documentação do produto para obter informações adicionais sobre como implementar aplicativos de serviço da web.

Para definir o serviço da web dentro de aplicativos XMS , você simplesmente precisa configurar a propriedade XMSC_IC_URL do objeto InitialContext para a URL do terminal de serviço da web. Por exemplo, se o serviço da web for implementado em um host do servidor chamado MyServer, um exemplo de uma URL de terminal de serviço da web:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

Configurar a propriedade XMSC_IC_URL permite que chamadas InitialContext Lookup chamem o serviço da web no terminal definido que, por sua vez, consulta o objeto administrado necessário por meio do serviço de nomenclatura COS.

Os aplicativos .NET podem usar o serviço da web. A implementação do lado do servidor é a mesma para XMS C, /C++ e, XMS.NET.XMS O .NET chama o serviço da web diretamente por meio do Microsoft .NET Framework

XMS .NET recuperação de objetos administrados

XMS recupera um objeto administrado do repositório usando o endereço fornecido quando o objeto InitialContext é criado, ou nas propriedades InitialContext.

Os objetos a serem recuperados podem ter os seguintes tipos de nomes:

- Um nome simples que descreve o objeto Destination, por exemplo, um destino de fila chamado SalesOrders
- Um nome composto, que pode ser composto de SubContextos, separados por '/' e deve terminar com o nome do objeto. Um exemplo de um nome composto é "Warehouse / PickLists/DispatchQueue2", em que Warehouse e Picklists são SubContextos no diretório de nomenclatura e DispatchQueue2 é o nome de um objeto de Destino.

Evitando que os aplicativos usem uma versão mais nova do XMS

Por padrão, quando uma versão mais recente do XMS é instalada, os aplicativos que usam a versão anterior alternam automaticamente para a versão mais recente sem precisar recompilar. No entanto, é possível evitar que os aplicativos usem a versão mais recente configurando um atributo no arquivo de configuração de aplicativo.

Sobre esta tarefa

O recurso de coexistência de várias versões assegura que a instalação de uma versão mais recente do XMS não sobrescreva a versão XMS anterior. Em vez disso, várias instâncias de montagens de XMS .NET semelhantes coexistem no Global Assembly Cache (GAC), mas possuem números de versão diferentes. Internamente, o GAC usa um arquivo de políticas para rotear as chamadas de aplicativo para a versão mais recente de XMS. Os aplicativos são executados sem uma necessidade de recompilação e podem usar novos recursos disponíveis na versão XMS .NET mais recente.

Procedimento

- Se um aplicativo for necessário para usar a versão do XMS .NET mais antiga, configure o atributo `publisherpolicy` para no no arquivo de configuração do aplicativo.

  Nota: Um arquivo de configuração de aplicativo é um arquivo com um nome que consiste no nome do programa executável para o qual o arquivo está relacionado, com o sufixo .config. Por exemplo, o arquivo de configuração de aplicativo para text.exe teria o nome text.exe.config.

  A qualquer momento, no entanto, todos os aplicativos de um sistema usam a mesma versão do XMS .NET.

Protegendo Comunicações para Aplicativos XMS

Esta seção fornece informações sobre a configuração de comunicações seguras para permitir que os aplicativos XMS se conectem por meio de Secure Sockets Layer (SSL) para um mecanismo de sistema de mensagens WebSphere Application Server service integration bus ou gerenciador de filas IBM MQ .

Sobre esta tarefa

A seção contém os seguintes tópicos:

- “Conexões seguras com um gerenciador de filas IBM MQ” na página 676
- “Mapeamentos de nomes de CipherSuite e CipherSpec para conexões XMS com um gerenciador de filas IBM MQ” na página 677
- “Conexões seguras com um mecanismo de sistema de mensagens WebSphere Application Server service integration bus” na página 677
- “Mapeamentos de Nome CipherSuite e CipherSpec para Conexões com um WebSphere Application Server service integration bus” na página 678

Conexões seguras com um gerenciador de filas IBM MQ

Para ativar um aplicativo XMS.NET para fazer conexões seguras com um gerenciador de filas IBM MQ , as propriedades relevantes devem ser definidas no objeto ConnectionFactory.

O protocolo usado na negociação de criptografia pode ser Secure Sockets Layer (SSL) ou Segurança da Camada de Transporte (TLS), dependendo de qual CipherSuite você especificar no objeto ConnectionFactory.

As propriedades ConnectionFactory para conexões usando SSL para um gerenciador de filas do IBM MQ , com uma descrição breve, são mostradas na tabela a seguir:

Tabela 93. Propriedades de ConnectionFactory para conexões com um gerenciador de filas IBM MQ via SSL

Nome da propriedade	**Descrição**
XMSC_WMQ_SSL_CERT_STORES	Os locais dos servidores que retêm as listas de revogação de certificado (CRLs) a serem usadas em uma conexão SSL com um gerenciador de filas.
XMSC_WMQ_SSL_CIPHER_SPEC	O nome da CipherSpec a ser usada em uma conexão segura com um gerenciador de filas.
XMSC_WMQ_SSL_CIPHER_SUITE	O nome do CipherSuite a ser usado em uma conexão TLS para um gerenciador de filas. O protocolo usado para negociar a conexão segura depende do CipherSuite especificado.
XMSC_WMQ_SSL_CRYPTO_HW	Detalhes de configuração para o hardware criptográfico conectado ao sistema do cliente.
XMSC_WMQ_SSL_FIPS_REQUIRED	O valor dessa propriedade determina se um aplicativo pode ou não usar conjuntos de cifras compatíveis não FIPS. Se essa propriedade for configurada como true, apenas algoritmos do FIPS serão usados para a conexão cliente-servidor.
XMSC_WMQ_SSL_KEY_REPOSITORY	O local do arquivo do banco de dados de chaves no qual chaves e certificados são armazenados.
XMSC_WMQ_SSL_KEY_RESETCOUNT	O KeyResetCount representa o número total de bytes não criptografados enviados e recebidos dentro de uma conversa SSL antes de a chave secreta ser renegociada.

Tabela 93. Propriedades de ConnectionFactory para conexões com um gerenciador de filas IBM MQ via SSL (continuação)	
Nome da propriedade	**Descrição**
XMSC_WMQ_SSL_PEER_NAME	O nome do peer a ser usado em uma conexão SSL com um gerenciador de filas.

Mapeamentos de nomes de CipherSuite e CipherSpec para conexões XMS com um gerenciador de filas IBM MQ

O InitialContext é convertido entre a propriedade SSLCIPHERSUITE da Connection Factory de JMSAdmin e o XMS próximo-equivalente XMSC_WMQ_SSL_CIPHER_SPEC. Uma conversão semelhante é necessária se você especificar um valor para XMSC_WMQ_SSL_CIPHER_SUITE mas omitir valor para XMSC_WMQ_SSL_CIPHER_SPEC.

Tabela 94 na página 677 lista os CipherSpecs disponíveis e seus equivalentes JSSE CipherSuite.

Tabela 94. CipherSpecs Disponíveis e seus Equivalentes JSSE CipherSuite	
CipherSpec	**Equivalente CipherSuite JSSE**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

Nota: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.

Conexões seguras com um mecanismo de sistema de mensagens WebSphere Application Server service integration bus

Para ativar um aplicativo XMS.NET para fazer conexões seguras com um mecanismo do sistema de mensagens WebSphere Application Server service integration bus , as propriedades relevantes devem ser definidas no objeto ConnectionFactory.

XMS fornece suporte SSL e HTTPS para conexões com um WebSphere Application Server service integration bus. SSL e HTTPS fornecem conexões seguras para autenticação e confidencialidade.

Assim como a segurança WebSphere , a segurança XMS é configurada com relação aos padrões de segurança JSSE e convenções de nomenclatura, que incluem o uso de CipherSuites para especificar os algoritmos que são usados ao negociar uma conexão segura. O protocolo usado na negociação de criptografia pode ser SSL ou TLS, dependendo de qual CipherSuite você especifica no objeto ConnectionFactory.

Tabela 95 na página 677 lista as propriedades que devem ser definidas no objeto ConnectionFactory.

Tabela 95. Propriedades de ConnectionFactory para Conexões Seguras com um Mecanismo de Sistema de Mensagens WebSphere Application Server service integration bus	
Nome da propriedade	**Descrição**
XMSC_WPM_SSL_CIPHER_SUITE	O nome doCipherSuite para ser usado em uma conexão TLS para umWebSphere Application Server service integration bus mecanismo de mensagens. O protocolo usado para negociar a conexão segura depende do CipherSuite especificado.

Tabela 95. Propriedades de ConnectionFactory para Conexões Seguras com um Mecanismo de Sistema de Mensagens WebSphere Application Server service integration bus (continuação)	
Nome da propriedade	**Descrição**
XMSC_WPM_SSL_KEYRING_LABEL	O certificado a ser usado ao autenticar-se com o servidor.

A seguir está um exemplo de propriedades ConnectionFactory para conexões seguras com um mecanismo de sistema de mensagens WebSphere Application Server service integration bus:

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

Em que chain_name deve ser configurado como BootstrapTunneledeledSecureMessaging ou BootstrapSecureMessaging e port_number é o número da porta na qual o servidor de autoinicialização atende a solicitações recebidas.

A seguir, está um exemplo de propriedades ConnectionFactory para conexões seguras com um mecanismo de sistema de mensagens WebSphere Application Server service integration bus com valores de amostra inseridos:

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

Mapeamentos de Nome CipherSuite e CipherSpec para Conexões com um WebSphere Application Server service integration bus

Como IBM Global Security Kit (GSKit) usa CipherSpecs em vez de CipherSuites, os nomes CipherSuite de estilo JSSE especificados na propriedade XMSC_WPM_SSL_CIPHER_SUITE devem ser mapeados para os nomes GSKit-style CipherSpec .

Tabela 96 na página 678 lista o CipherSpec equivalente para cada CipherSuite reconhecido.

Tabela 96. CipherSuites Disponíveis e seus CipherSpecs equivalentes	
CipherSuite	**CipherSpec equivalente**
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

Nota: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA foi descontinuado. No entanto, ele ainda pode ser usado para transferir até 32 GB de dados antes de a conexão ser finalizada com erro AMQ9288. Para evitar esse erro, você precisará evitar o uso de DES triplo ou ativar a reconfiguração de chave secreta ao usar esse CipherSpec.

Mensagens XMS

Esta seção descreve a estrutura e o conteúdo de mensagens XMS e explica como os aplicativos processam mensagens XMS .

Esta seção contém os seguintes tópicos:

Partes de uma mensagem XMS

Uma mensagem XMS consiste em um cabeçalho, um conjunto de propriedades e um corpo.

Cabeçalho
O cabeçalho de uma mensagem contém campos, e todas as mensagens contêm o mesmo conjunto de campos de cabeçalho. XMS e aplicativos usam os valores dos campos de cabeçalho para identificar e rotear mensagens. Para obter mais informações sobre campos de cabeçalho, consulte “Campos de cabeçalho em uma mensagem XMS” na página 679.

Conjunto de propriedades
As propriedades de uma mensagem especificam informações adicionais sobre a mensagem. Embora todas as mensagens tenham o mesmo conjunto de campos de cabeçalho, cada mensagem pode ter um conjunto diferente de propriedades. Para obter mais informações, consulte “Propriedades de uma mensagem XMS” na página 680.

Conteúdo
O corpo de uma mensagem contém dados do aplicativo. Para obter mais informações, consulte “O corpo de uma mensagem XMS” na página 683.

Um aplicativo pode selecionar quais mensagens ele deseja receber. Usando seletores de mensagens, que especificam os critérios de seleção. Os critérios podem ser baseados nos valores de determinados campos de cabeçalho e os valores de qualquer uma das propriedades de uma mensagem. Para obter mais informações sobre os seletores de mensagem, veja “Seletores de mensagens” na página 686.

Campos de cabeçalho em uma mensagem XMS

Para permitir que um aplicativo XMS troque mensagens com um aplicativo WebSphere JMS, o cabeçalho de uma mensagem XMS contém os campos de cabeçalho da mensagem JMS.

Os nomes desses campos de cabeçalho iniciam com o prefixo JMS. Para obter uma descrição dos campos de cabeçalho da mensagem JMS , consulte a *EspecificaçãoJava Message Service*.

XMS implementa os campos de cabeçalho da mensagem JMS como atributos de um objeto de Mensagem. Cada campo de cabeçalho possui seus próprios métodos para configurar e obter seu valor. Para obter uma descrição desses métodos, consulte IMessage. Um campo de cabeçalho é sempre legível e gravável.

Tabela 97 na página 679 lista os campos de cabeçalho da mensagem JMS e indica como o valor de cada campo é configurado para uma mensagem transmitida. Alguns dos campos são configurados automaticamente por XMS quando um aplicativo envia uma mensagem ou, no caso de JMSRedelivered, quando um aplicativo recebe uma mensagem.

Tabela 97. Campos de cabeçalho da mensagem JMS.]

Nome do campo de cabeçalho de mensagem do JMS	**Como o valor é configurado para uma mensagem transmitida (no formato *method* [*class*])**
JMSCorrelationID	Set JMSCorrelationID [Message]
JMSDeliveryMode	Send [MessageProducer]
JMSDestination	Send [MessageProducer]
JMSExpiration	Send [MessageProducer]

Tabela 97. Campos de cabeçalho da mensagem JMS.] (continuação)	
Nome do campo de cabeçalho de mensagem do JMS	**Como o valor é configurado para uma mensagem transmitida (no formato *method* [*class*])**
JMSMessageID	Send [MessageProducer]
JMSPriority	Send [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Set JMSReplyTo [Message]
JMSTimestamp	Send [MessageProducer]
JMSType	Set JMSType [Message]

Propriedades de uma mensagem XMS

XMS suporta três tipos de propriedade de mensagem: JMS propriedades definidas, propriedades definidas do IBM e propriedades definidas pelo aplicativo.

Um aplicativo XMS pode trocar mensagens com um aplicativo WebSphere JMS porque XMS suporta as seguintes propriedades predefinidas de um objeto Mensagem:

- As mesmas propriedades definidas por JMS que o WebSphere JMS suporta. Os nomes dessas propriedades começam com o prefixo JMSX.
- As mesmas propriedades definidas por IBM que o WebSphere JMS suporta. Os nomes dessas propriedades começam com o prefixo JMS_IBM_.

Cada propriedade predefinida possui dois nomes:

- Um nome JMS, para uma propriedade definida por JMS, ou um nome WebSphere JMS, para uma propriedade definida por IBM.

  Este é o nome pelo qual a propriedade é conhecida em JMS ou WebSphere JMS, e é também o nome que é transmitido com uma mensagem que tem esta propriedade. Um aplicativo XMS usa este nome para identificar a propriedade em uma expressão de seletor de mensagem.
- Um nome de XMS para identificar a propriedade em todas as situações, exceto em uma expressão de seletor de mensagem. Cada nome XMS é definido como uma constante nomeada na classe `IBM.XMS.XMSC` . O valor da constante nomeada é o nome JMS ou WebSphereJMS correspondente.

Além das propriedades predefinidas, um aplicativo XMS pode criar e usar seu próprio conjunto de propriedades de mensagem. Essas propriedades são chamadas de *propriedades definidas pelo aplicativo.*

Depois que um aplicativo cria uma mensagem, as propriedades da mensagem são legíveis e graváveis. As propriedades permanecem legíveis e graváveis depois que o aplicativo envia a mensagem. Quando um aplicativo recebe uma mensagem, as propriedades da mensagem são somente leitura. Se um aplicativo chamar o método `Clear Properties` da classe de Mensagem quando as propriedades de uma mensagem forem somente leitura, as propriedades se tornarão legíveis e graváveis O método também limpa as propriedades.

A mensagem recebida, quando encaminhada após a limpeza das propriedades da mensagem, se comportará de uma maneira consistente com o comportamento de encaminhamento de um BytesMessage XMS for .NET do WMQ padrão com propriedades de mensagem limpas.

Isso, no entanto, não é recomendado desde que as seguintes propriedades sejam perdidas:

- Valor da propriedade JMS_IBM_Encoding, implicando que os dados da mensagem não podem ser decodificados de forma significativa.
- Valor da propriedade JMS_IBM_Format, implicando que o encadeamento de cabeçalho entre o cabeçalho da mensagem (MQMD ou o novo MQRFH2) e os cabeçalhos existentes seriam quebrados.

Para determinar os valores de todas as propriedades de uma mensagem, um aplicativo pode chamar o método Get Properties da classe Message. O método cria um agente iterativo que encapsula uma lista de

objetos de Propriedade, em que cada objeto Propriedade representa uma propriedade da mensagem. O aplicativo pode então usar os métodos da classe Iterator para recuperar cada objeto de Propriedade por vez, e ele pode usar os métodos da classe de Propriedade para recuperar o nome, o tipo de dados e o valor de cada propriedade.

JMS-propriedades definidas de uma mensagem

Várias propriedades definidas por JMS de uma mensagem são suportadas por ambos XMS e WebSphere JMS.

Tabela 98 na página 681 lista as propriedades definidas por JMS de uma mensagem que são suportadas por ambos XMS e WebSphere JMS. Para obter uma descrição das propriedades definidas pelo JMS, consulte *Java Message Service Especificação*.. As propriedades JMS-definidas não são válidas para uma conexão em tempo real com um broker.

A tabela especifica o tipo de dados de cada propriedade e indica como o valor da propriedade é configurado para uma mensagem transmitida. Algumas das propriedades são configuradas automaticamente por XMS quando um aplicativo envia uma mensagem ou, no caso de JMSXDeliveryCount, quando um aplicativo recebe uma mensagem.

Tabela 98. JMS-propriedades definidas de uma mensagem

Nome XMS da propriedade JMS definida	**Nome de JMS**	**Tipo de Dados**	**Como o valor é configurado para uma mensagem transmitida (no formato *method* [*class*])**
JMSX_APPID	JMSXAppID	System.String	Send [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Configurar propriedade de sequência [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Send [MessageProducer]

IBM-propriedades definidas de uma mensagem

Várias propriedades definidas pela IBM de uma mensagem são suportadas por XMS e WebSphere JMS.

Tabela 99 na página 682 lista as propriedades definidas IBM de uma mensagem que são suportadas por XMS e WebSphere JMS. Para obter mais informações sobre as propriedades definidas por IBM, consulte IBM MQ ou a documentação do produto WebSphere Application Server.

A tabela especifica o tipo de dados de cada propriedade e indica como o valor da propriedade é configurado para uma mensagem transmitida. Algumas das propriedades são configuradas automaticamente por XMS quando um aplicativo envia uma mensagem.

Tabela 99. Propriedades Definidas pela IBM de uma Mensagem

O nome do XMS da propriedade definida da IBM	**WebSphere JMS nome**	**Tipo de Dados**	**Como o valor é configurado para uma mensagem transmitida (no formato *method* [*class*])**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Configurar Propriedade Integer [PropertyContext]
CODIFICAÇÃO DE JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEMA DESTINO:	JMS_IBM_ExceptionProblemDestination	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Configurar Propriedade Integer [PropertyContext]
FORMATO JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Configurar propriedade de sequência [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Configurar Propriedade Integer [PropertyContext]

Tabela 99. Propriedades Definidas pela IBM de uma Mensagem (continuação)			
O nome do XMS da propriedade definida da IBM	**WebSphere JMS nome**	**Tipo de Dados**	**Como o valor é configurado para uma mensagem transmitida (no formato *method* [*class*])**
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ ID	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Configurar Propriedade Integer [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Send [MessageProducer]

Propriedades Definidas pelo Aplicativo de uma Mensagem

Um aplicativo XMS pode criar e usar seu próprio conjunto de propriedades de mensagem. Quando um aplicativo envia uma mensagem, essas propriedades também são transmitidas com a mensagem. Um aplicativo de recebimento, usando seletores de mensagens, pode, então, selecionar quais mensagens ele deseja receber com base nos valores dessas propriedades.

Para permitir que um aplicativo WebSphereJMS para selecionar e processar mensagens enviadas por um aplicativo XMS , o nome de uma propriedade definida pelo aplicativo deve estar em conformidade com as regras para formar identificadores nas expressões do seletor de mensagem. Para obter informações adicionais, consulte “Seletores de mensagens no JMS” na página 146. O valor de uma propriedade de aplicativo definido deve ter um dos seguintes tipos de dados: System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Double ou System.String.

O corpo de uma mensagem XMS

O corpo de uma mensagem contém dados do aplicativo. No entanto, uma mensagem não pode ter nenhum corpo e engloba apenas os campos de cabeçalho e propriedades.

XMS suporta cinco tipos de corpo da mensagem:

Bytes
O corpo contém um fluxo de bytes. Uma mensagem com esse tipo de corpo é chamada de *mensagem de bytes*. A interface IBytesMessage contém os métodos para processar o corpo de uma mensagem de bytes.

Mapa
O corpo contém um conjunto de pares nome-valor, em que cada valor tem um tipo de dados associado. Uma mensagem com esse tipo de corpo é chamada de *mensagem de mapa*. A interface IMapMessage contém os métodos para processar o corpo de uma mensagem de mapa.

Object
O corpo contém um objeto serializado Java ou .NET . Uma mensagem com esse tipo de corpo é chamada de *mensagem de objeto*. A interface IObjectMessage contém os métodos para processar o corpo de uma mensagem de objeto.

Fluxo
O corpo contém um fluxo de valores, em que cada valor possui um tipo de dados associado. Uma mensagem com esse tipo de corpo é chamada de *mensagem de fluxo*. A interface IStreamMessage contém os métodos para processar o corpo de uma mensagem de fluxo.

Texto
O corpo contém uma sequência. Uma mensagem com esse tipo de corpo é chamada de *mensagem de texto*. A interface ITextMessage contém os métodos para processar o corpo de uma mensagem de texto.

A interface IMessage é o pai de todos os objetos de mensagem e pode ser usada em funções do sistema de mensagens para representar qualquer um dos tipos de mensagens do XMS.

Para obter informações sobre o tamanho e o máximo e o mínimo de valores de cada um desses tipos de dados, consulte Tabela 85 na página 662.

Mensagens de Bytes

O corpo de uma mensagem de bytes contém um fluxo de bytes. O corpo contém apenas os dados reais, e é responsabilidade dos aplicativos de envio e recebimento interpretarem esses dados.

As mensagens de bytes são úteis se um aplicativo XMS precisa trocar mensagens com aplicativos que não estão usando a interface de programação de aplicativos XMS ou JMS.

Depois que um aplicativo cria uma mensagem de byte, o corpo da mensagem é somente gravação. O aplicativo monta os dados do aplicativo para o corpo, chamando os métodos de gravação apropriados da interface IBytesMessage para .NET. Toda vez que o aplicativo grava um valor para o fluxo de mensagens de bytes, o valor é montado imediatamente após o valor anterior gravado pelo aplicativo. XMS mantém um cursor interno para lembrar a posição do último byte que foi montado.

Quando o aplicativo envia a mensagem, o corpo da mensagem se torna somente leitura. Nesse modo, o aplicativo pode enviar a mensagem repetidamente.

Quando um aplicativo recebe uma mensagem de bytes, o corpo da mensagem é de leitura. O aplicativo pode usar os métodos de leitura apropriados da interface IBytesMessage para ler o conteúdo do fluxo de mensagens de bytes. O aplicativo lê os bytes na sequência e XMS mantém um cursor interno para lembrar a posição do último byte que foi lido.

Se um aplicativo chamar o método Reconfigurar da interface IBytesMessage quando o corpo de uma mensagem de bytes for gravável, o corpo se tornará somente leitura. O método também reposiciona o cursor no início do fluxo de mensagens de bytes.

Se um aplicativo chamar o método `Clear Body` da interface IMessage para o .NET quando o corpo de uma mensagem de bytes for somente leitura, o corpo se tornará gravável. O método também limpa o corpo.

Mensagens de Mapa

O corpo de uma mensagem de mapa contém um conjunto de pares nome-valor, em que cada valor tem um tipo de dados associado.

Em cada par de nome-valor, o nome é uma sequência que identifica o valor e o valor é um elemento de dados do aplicativo que possui um dos tipos de dados XMS listados em Tabela 100 na página 686. A

ordem dos pares nome-valor não está definida. A classe MapMessage contém os métodos para configurar e obter pares nome-valor.

Um aplicativo pode acessar um par nome-valor aleatoriamente, especificando seu nome.

Um aplicativo .NET pode usar a propriedade MapNames para obter uma enumeração dos nomes no corpo da mensagem de mapa.

Quando um aplicativo obtém o valor de um par nome-valor, o valor pode ser convertido por XMS em outro tipo de dados. Por exemplo, para obter um número inteiro a partir do corpo de uma mensagem de mapa, um aplicativo pode chamar o método GetString da classe MapMessage, que retorna o número inteiro como uma sequência. As conversões suportadas são as mesmas que aquelas suportadas quando XMS converte um valor de propriedade de um tipo de dados para outro. Para obter mais informações sobre as conversões suportadas, consulte “Conversão implícita de um valor de propriedade de um tipo de dados para outro” na página 663.

Depois que um aplicativo cria uma mensagem de mapa, o corpo da mensagem é legível e gravável. O corpo permanece legível e gravável depois que o aplicativo envia a mensagem. Quando um aplicativo recebe uma mensagem de mapa, o corpo da mensagem é de leitura. Se um aplicativo chamar o método Clear Body da classe de mensagem quando o corpo de uma mensagem de mapa for de leitura, o corpo se tornará legível e gravável. O método também limpa o corpo.

Mensagens de objeto

O corpo de uma mensagem de objeto contém um objeto serializadoJava ou .NET .

Um aplicativo XMS pode receber uma mensagem de objeto, alterar seus campos de cabeçalho e propriedades e, em seguida, enviá-lo para outro destino. Um aplicativo também pode copiar o corpo de uma mensagem de objeto e usá-lo para formar outra mensagem de objeto. XMS trata o corpo de uma mensagem de objeto como uma matriz de bytes.

Depois que um aplicativo cria uma mensagem de objeto, o corpo da mensagem é legível e gravável. O corpo permanece legível e gravável depois que o aplicativo envia a mensagem. Quando um aplicativo recebe uma mensagem de objeto, o corpo da mensagem é de leitura. Se um aplicativo chamar o método `Clear Body` da interface de IMessage para .NET quando o corpo de uma mensagem de objeto for de leitura, o corpo se tornará legível e gravável. O método também limpa o corpo.

Mensagens de Fluxo

O corpo de uma mensagem de fluxo contém um fluxo de valores, em que cada valor possui um tipo de dados associado.

O tipo de dados de um valor é um dos tipos de dados XMS listados em Tabela 100 na página 686.

Depois que um aplicativo cria uma mensagem de fluxo, o corpo da mensagem é gravável. O aplicativo monta os dados do aplicativo para o corpo, chamando os métodos de gravação apropriados da interface IStreamMessage para .NET. Toda vez que o aplicativo grava um valor para o fluxo de mensagens, o valor e seu tipo de dados são montados imediatamente após o valor anterior gravado pelo aplicativo. XMS mantém um cursor interno para lembrar a posição do último valor que foi montado.

Quando o aplicativo envia a mensagem, o corpo da mensagem se torna somente leitura. Nesse modo, o aplicativo pode enviar a mensagem várias vezes.

Quando um aplicativo recebe uma mensagem de fluxo, o corpo da mensagem é de leitura. O aplicativo pode usar os métodos de leitura apropriados da interface IStreamMessage para .NET para ler o conteúdo do fluxo de mensagens. O aplicativo lê os valores na sequência, e XMS mantém um cursor interno para lembrar a posição do último valor que foi lido.

Quando um aplicativo lê um valor do fluxo de mensagens, o valor pode ser convertido pelo XMS em outro tipo de dados. Por exemplo, para ler um número inteiro a partir do fluxo de mensagens, um aplicativo pode chamar o método ReadString, que retorna o número inteiro como uma sequência. As conversões suportadas são as mesmas que aquelas suportadas quando XMS converte um valor de propriedade de

um tipo de dados para outro. Para obter mais informações sobre as conversões suportadas, consulte “Conversão implícita de um valor de propriedade de um tipo de dados para outro” na página 663.

Se um erro ocorrer enquanto um aplicativo está tentando ler um valor do fluxo de mensagens, o cursor não está avançado. O aplicativo pode se recuperar do erro ao tentar ler o valor como outro tipo de dados.

Se um aplicativo chamar o método Reconfigurar da interface IStreamMessage para XMS quando o corpo de uma mensagem de fluxo for de gravação somente, o corpo se tornará somente leitura. O método também reposiciona o cursor no início do fluxo de mensagens.

Se um aplicativo chamar o método `Clear Body` da interface de IMessage para XMS quando o corpo de uma mensagem de fluxo for de leitura, o corpo se tornará somente gravação. O método também limpa o corpo.

Mensagens de texto

O corpo de uma mensagem de texto contém uma sequência.

Depois que um aplicativo cria uma mensagem de texto, o corpo da mensagem é legível e gravável. O corpo permanece legível e gravável depois que o aplicativo envia a mensagem. Quando um aplicativo recebe uma mensagem de texto, o corpo da mensagem é somente leitura. Se um aplicativo chamar o método Clear Body da interface IMessage para .NET quando o corpo de uma mensagem de texto for de leitura, o corpo se tornará legível e gravável. O método também limpa o corpo.

Tipos de Dados para Elementos de Dados do Aplicativo

Para assegurar que um aplicativo XMS possa trocar mensagens com um aplicativo IBM MQ classes for JMS, ambos os aplicativos devem ser capazes de interpretar os dados do aplicativo no corpo de uma mensagem da mesma maneira.

Por esse motivo, cada elemento de dados do aplicativo gravado no corpo de uma mensagem por um aplicativo XMS deve ter um dos tipos de dados listados em Tabela 100 na página 686. Para cada tipo de dados, a tabela mostra o tipo de dados Java compatível. XMS fornece os métodos para gravar elementos de dados do aplicativo apenas com esses tipos de dados.

Tabela 100. Tipos de dados XMS que são compatíveis com os tipos de dados Java

XMS Tipo de dados	**Representa**	**Tipo de DadosJava Compatível**
System.Boolean	O valor booleano true ou false	booleano
System.Char16	Caractere de byte duplo	char
System.SByte	Inteiro de 8 bits sinalizado	byte
System.Int16	Número inteiro de 16 bits assinado	short
System.Int32	Número inteiro de 32 bits assinado	int
System.Int64	Número inteiro de 64 bits sinalizado	long
System.Float	Número de vírgula flutuante assinado	float
System.Double	Número de vírgula flutuante de precisão dupla assinado	double
System.String	Sequência de caracteres	Sequência

Para obter informações sobre o tamanho, o valor máximo e o valor mínimo de cada um desses tipos de dados, consulte “Tipos primitivos do XMS” na página 662.

Seletores de mensagens

Um aplicativo XMS usa seletores de mensagens para selecionar as mensagens que ele deseja receber.

Quando um aplicativo cria um consumidor de mensagens, ele pode associar uma expressão de seletor de mensagem ao consumidor. A expressão do seletor de mensagem especifica os critérios de seleção.

Quando um aplicativo está se conectando ao gerenciador de filas IBM WebSphere MQ 7.0 , a seleção de mensagens é feita no lado do gerenciador de filas. XMS não faz nenhuma seleção e simplesmente entrega a mensagem que recebeu do gerenciador de filas, fornecendo, assim, melhor desempenho.

Um aplicativo pode criar mais de um consumidor de mensagens, cada um com sua própria expressão de seletor de mensagem. Se uma mensagem recebida atender aos critérios de seleção de mais de um consumidor de mensagens, o XMS entregará a mensagem para cada um desses consumidores.

Uma expressão de seletor de mensagem pode referenciar as seguintes propriedades de uma mensagem:

- Propriedades Definidas pelo JMS
- Propriedades Definidas pela IBM
- Propriedades Definidas pelo Aplicativo

Ele também pode referenciar os campos de cabeçalho da mensagem a seguir:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

Uma expressão de seletor de mensagem, no entanto, não pode referenciar dados no corpo de uma mensagem.

Aqui está um exemplo de uma expressão de seletor de mensagem:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

O XMS entrega uma mensagem para um consumidor de mensagens com essa expressão de seletor de mensagem apenas se a mensagem tiver uma prioridade maior que 3; uma propriedade definida pelo aplicativo, fabricante, com um valor de `Jaguar;` e outra propriedade definida pelo aplicativo, modelo, com um valor de `xj6` ou `xj12.`

As regras de sintaxe para formar uma expressão de seletor de mensagem em XMS são as mesmas que aquelas em IBM MQ classes for JMS. Para obter informações sobre como construir uma expressão do seletor de mensagens, consulte a Nota de documentação do produto IBM MQ que, em uma expressão do seletor de mensagens, os nomes de propriedades definidas de JMS devem ser os nomes de JMS, e os de propriedades definidas de IBM devem ser os nomes IBM MQ classes for JMS. Não é possível usar os nomes XMS em uma expressão de seletor de mensagem.

Mapeando mensagens do XMS para mensagens do IBM MQ

Os campos de cabeçalho JMS e propriedades de uma mensagem XMS são mapeados em campos nas estruturas de cabeçalho de uma mensagem IBM MQ.

Quando um aplicativo XMS está conectado a um gerenciador de filas IBM MQ , as mensagens enviadas para o gerenciador de filas são mapeadas para as mensagens IBM MQ da mesma maneira que as mensagens IBM MQ classes for JMS são mapeadas para mensagens IBM MQ em circunstâncias semelhantes.

Se a propriedade XMSC_WMQ_TARGET_CLIENT de um objeto Destination estiver configurada como XMSC_WMQ_TARGET_DEST_JMS, os campos de cabeçalho JMS e as propriedades de uma mensagem enviada para o destino serão mapeados para os campos nas estruturas de cabeçalho MQMD e MQRFH2 da mensagem IBM MQ . Configurar a propriedade XMSC_WMQ_TARGET_CLIENT dessa maneira assume que o aplicativo que recebe a mensagem pode manipular um cabeçalho MQRFH2. O aplicativo de

recebimento pode, portanto, ser outro aplicativo XMS , um aplicativo IBM MQ classes for JMS ou um aplicativo IBM MQ nativo que tenha sido projetado para manipular um cabeçalho MQRFH2.

Se a propriedade XMSC_WMQ_TARGET_CLIENT de um objeto de Destino for configurada como XMSC_WMQ_TARGET_DEST_MQ em vez disso, os campos de cabeçalho JMS e as propriedades de uma mensagem enviada para o destino serão mapeados para os campos na estrutura de cabeçalho MQMD da mensagem IBM MQ . A mensagem não contém um cabeçalho MQRFH2 e quaisquer campos de cabeçalho JMS e propriedades que não podem ser mapeados para campos na estrutura de cabeçalho MQMD são ignorados. O aplicativo que recebe a mensagem pode, portanto, ser um IBM MQ nativo que não foi projetado para manipular um cabeçalho MQRFH2.

As mensagens do IBM MQ recebidas de um gerenciador de filas são mapeadas para mensagens do XMS da mesma maneira que mensagens do IBM MQ são mapeadas para mensagens do IBM MQ classes for JMS em circunstâncias semelhantes.

Se uma mensagem IBM MQ recebida tiver um cabeçalho MQRFH2, a mensagem XMS resultante terá um corpo cujo tipo seja determinado pelo valor da propriedade **Msd** contida na pasta `mcd` do cabeçalho MQRFH2. Se a propriedade **Msd** não estiver presente no cabeçalho MQRFH2, ou se a mensagem IBM MQ não tiver cabeçalho MQRFH2, a mensagem XMS resultante terá um corpo cujo tipo seja determinado pelo valor do campo *`Format`* no cabeçalho MQMD. Se o campo *`Format`* estiver configurado como MQFMT_STRING, a mensagem XMS será uma mensagem de texto. Caso contrário, a mensagem XMS é uma mensagem de bytes. Se a mensagem IBM MQ não tiver cabeçalho MQRFH2, apenas os campos de cabeçalho JMS e propriedades que podem ser derivados de campos no cabeçalho MQMD são configurados.

Para obter mais informações sobre o mapeamento de mensagens IBM MQ classes for JMS para mensagens IBM MQ , consulte “Mapeando mensagens do JMS para mensagens do IBM MQ” na página 150.

Lendo e gravando o descritor de mensagens a partir de um aplicativo IBM MQ Message Service Client (XMS) for .NET

É possível acessar todos os campos do descritor de mensagens (MQMD) de uma mensagem IBM MQ , exceto StrucId e Versão; BackoutCount pode ser lido, mas não gravado.

Os atributos da mensagem fornecidos pelo IBM MQ Message Service Client (XMS) for .NET facilitam os aplicativos XMS para configurar campos MQMD e também para a unidade de aplicativos IBM WebSphere MQ .

Algumas restrições se aplicam ao usar o sistema de mensagens de publicação / assinatura. Por exemplo, os campos MQMD como MsgID e CorrelId, se configurados, são ignorados.

A função também estará indisponível quando a propriedade **PROVIDERVERSION** for configurada como 6.

Acessando IBM MQDados da mensagem a partir de um aplicativo IBM MQ Message Service Client (XMS) for .NET

Você pode acessar os dados da mensagem completos IBM MQ, incluindo o cabeçalho MQRFH2 (se presente) e quaisquer outros cabeçalhos IBM MQ (se presentes) dentro de um aplicativo IBM MQ Message Service Client (XMS) for .NET como o corpo de uma JMSBytesMessage.

A função descrita neste tópico está disponível apenas ao conectar-se com um gerenciador de filas do IBM WebSphere MQ 7.0 ou mais recente e quando o provedor de sistemas de mensagens do IBM MQ está no modo normal.

As propriedades do objeto de destino determinam como o aplicativo XMS acessa o conjunto de uma mensagem IBM MQ (incluindo o cabeçalho MQRFH2, se presente) como o corpo de uma JMSBytesMessage.

ALW Desenvolvendo aplicativos clientes AMQP

O suporte IBM MQ para APIs de AMQP, permite que um administrador IBM MQ crie um canal de AMQP. Quando ele é iniciado, este canal define um número de porta que aceita conexões de aplicativos clientes AMQP.

É possível instalar um canal AMQP em sistemas AIX, Linux, and Windows ; ele não está disponível no IBM i ou no z/OS.

Um aplicativo cliente do AMQP 1.0 pode se conectar ao gerenciador de filas com um canal de AMQP.

Desenvolvimento de aplicativos usando a biblioteca Apache Qpid JMS

Introdução

A biblioteca Apache Qpid JMS usa o protocolo AMQP 1.0 para fornecer uma implementação da especificação JMS 2.

Apache Qpid JMS usa alguns aspectos do protocolo AMQP 10 de uma maneira diferente das APIs de sistemas de mensagens do MQ Light. O IBM MQ 9.2 incluiu suporte para os canais AMQP do IBM MQ , para que os aplicativos do Apache Qpid JMS possam se conectar ao IBM MQ e ao sistema de mensagens de publicação / assinatura, incluindo o uso de assinaturas compartilhadas.

O IBM MQ 9.3 incluiu suporte adicional nos canais AMQP do IBM MQ , para que os aplicativos Apache Qpid JMS possam se conectar ao IBM MQ e executar o sistema de mensagens ponto a ponto. Consulte o “Suporte ponto a ponto em canais AMQP” na página 693 para obter informações adicionais.

IBM MQ 9.3.0 incluiu suporte de procura de fila adicional para canais AMQP do IBM MQ , para que Apache Qpid os aplicativos JMS possam se conectar ao IBM MQ e executar a navegação de mensagens de uma fila. Consulte “Suporte ponto a ponto em canais AMQP” na página 693 para obter mais informações.

IBM MQ 9.3.0 incluiu dois atributos de canal adicionais para canais AMQP, TMPMODEL e TMPQPRFX. Esses atributos são para a fila modelo e prefixo de fila temporário a ser usado ao criar uma fila temporária.

Intercomunicação com outros aplicativos IBM MQ

É possível enviar mensagens entre os aplicativos Apache Qpid JMS e outros aplicativos do IBM MQ. Por exemplo, um aplicativo Apache Qpid pode publicar mensagens em um tópico e os aplicativos MQ Light podem recebê-las criando uma assinatura.

Um aplicativo Apache Qpid JMS também pode publicar as mensagens que são consumidas pelos aplicativos tradicionais do IBM MQ, por exemplo, usando a chamada da API MQSUB para assinar o mesmo tópico.

Da mesma forma, os aplicativos Apache Qpid JMS podem assinar os tópicos do IBM MQ nos quais os aplicativos tradicionais do IBM MQ publicam mensagens.

Um aplicativo Apache Qpid JMS também pode compartilhar uma assinatura com um aplicativo do MQ Light, desde que ambos os clientes especifiquem o mesmo nome de compartilhamento e padrão de tópico.

Observe que, para fazer isso, o aplicativo Apache Qpid JMS não deve se conectar com um ID do cliente. Isso assegura que o nome de assinatura do IBM MQ usado por ambos os aplicativos seja o mesmo.

Atenção: Um aplicativo Apache Qpid JMS não pode compartilhar uma assinatura com um aplicativo do IBM MQ JMS.

Restrições do Apache Qpid JMS

Os recursos do JMS a seguir são suportados:

- Reconhecimento do cliente, reconhecimento automático e modo de reconhecimento dups OK (DUPS_OK_ACKNOWLEDGE)
  - Conexão com ou sem credenciais

  - Criação de um consumidor em um destino de tópico
  - Criação de um consumidor durável em um destino de tópico
  - Criação de um consumidor compartilhado em um destino de tópico
  - Criação de um consumidor durável compartilhado em um destino de tópico
  - Modos de reconhecimento do cliente e de reconhecimento automático
  - Confirmação de mensagem e confirmação de sessão
  - Cancelamento de uma assinatura durável
  - Criando uma fila temporária
  - Criando um consumidor em uma fila ou em um destino de fila temporária
  - MessageListeners JMS
  - Consumidor JMS para receber corpo; o método JMS 2.0 chamado `Consumer.receiveBody()`
  - Os tipos de mensagens JMS a seguir são suportados:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - Procurando mensagens em uma fila

Os recursos do JMS a seguir não são suportados por clientes AMQP:

- O uso de sessões transacionadas e de JMSContexts transacionados
  - O uso de seletores de mensagens
  - O uso do atributo **`nolocal`**
  - O uso de sessões transacionadas
  - O uso do atraso de entrega
  - No IBM MQ 9.3.0, procurando mensagens em uma fila.
  - Criando várias assinaturas ou clientes duráveis com o mesmo ID e tópico do cliente
  - Tópicos temporários de JMS
  - Os filtros do AMQP não são suportados.

Fazendo Download de Clientes AMQP de Amostra

IBM MQ não envia clientes AMQP, mas é possível fazer download de clientes MQ Light ou fazer download de clientes AMQP livres com base nas bibliotecas do Apache Qpid. Para obter mais informações, consulte IBM MQ Light e Apache Qpid.

Também é possível fazer download de outros clientes do AMQP de software livre com base nas bibliotecas Qpid do Apache. Para obter informações adicionais, consulte https://qpid.apache.org/index.html.

Atenção: IBM O suporte não pode fornecer suporte de configuração ou defeito para esses pacotes do cliente e quaisquer perguntas de uso ou relatórios de defeitos de código devem ser direcionados para os projetos respectivos

Implementando clientes AMQP para IBM MQ

Quando um aplicativo estiver pronto para ser implementado, ele irá requer todos os recursos de monitoramento, confiabilidade e segurança de outros aplicativos corporativos. Ele também pode trocar dados com outros aplicativos corporativos.

Ao implementar um cliente AMQP, é possível trocar mensagens com aplicativos IBM MQ. Por exemplo, se você usar o cliente AMQP para enviar uma mensagem de sequênciaJavaScript, o aplicativo IBM MQ receberá uma mensagem MQ, em que o campo de Formatar do MQMD é configurado como MQSTR.

Gerenciando o canal AMQP

O canal AMQP pode ser gerenciado da mesma forma que outros canais do MQ. É possível usar os comandos do MQSC, as mensagens de comando PCF ou IBM MQ Explorer para definir, iniciar, parar e gerenciar os canais. Em Criando e usando canais AMQP, comandos de exemplo são fornecidos para definir e iniciar a conexão de clientes a um gerenciador de filas.

Quando um canal AMQP é iniciado, é possível testá-lo conectando um cliente do AMQP 1.0. Por exemplo, MQ Light, Apache Qpid Proton ou Apache Qpid JMS.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW MQ Light, Apache Qpid JMSe AMQP (Advanced Message Queuing Protocol)

O cliente MQ Light , os clientes Qpid Apache como Apache Proton e Apache Qpid JMS APIs são baseados no protocolo de ligação OASIS Standard AMQP 1.0 . O AMQP especifica como as mensagens são enviadas entre emissores e receptores. Um aplicativo age como um emissor quando o aplicativo envia uma mensagem para o broker de mensagem, como IBM MQ. IBM MQ age como um remetente quando ele envia uma mensagem para um aplicativo AMQP.

Alguns dos benefícios do AMQP são os seguintes:

- Um protocolo padronizado aberto
- Compatibilidade com outros clientes AMQP 1.0 de software livre
- Muitas implementações de cliente de software livre disponíveis

Embora qualquer cliente AMQP 1,0 pode se conectar a um canal AMQP, alguns recursos AMQP não são suportados, por exemplo transações ou múltiplas sessões.

Para obter mais informações, consulte o website do AMQP.org e o PDF do OASIS Standard AMQP 1.0.

As APIs do MQ Light e do Apache Qpid JMS têm os recursos de sistema de mensagens a seguir:

- Entrega de mensagem no-máximo-uma
- Entrega de mensagem ao-menos-uma
- Endereçamento de destino da sequência de tópicos
- Mensagem e durabilidade do destino
- Destinos compartilhados para permitir que vários assinantes compartilhem a carga de trabalho
- Controle do cliente para fácil resolução de clientes interrompidos
- Leitura antecipada de mensagens configuráveis
- Confirmação de mensagens configuráveis

Para obter a documentação completa da API do Apache Qpid JMS, consulte Qpid JMS.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Suporte do AMQP 1.0

Os canais do AMQP fornecem um nível de suporte para aplicativos em conformidade com o AMQP 1.0.

Os canais AMQP suportam uma subrede do protocolo AMQP 1.0. É possível conectar clientes compatíveis com o AMQP 1.0 a um canal do IBM MQ do AMQP. Para usar todos os recursos do sistema de mensagens suportados pelos canais AMQP, deve-se configurar corretamente o valor de determinados campos do AMQP 1.0.

Essas informações esboçam a maneira que os campos do AMQP devem ser formatados e listam os recursos da especificação do AMQP 1.0 que não são suportados por canais AMQP.

Os seguintes recursos da especificação do AMQP 1.0 não são suportados ou tem um uso limitado:

Estrutura ATTACH

Os canais AMQP esperam que os recursos no quadro ANEXAR contenham um dos seguintes:

```
topic
temporary queue
queue
shared
```

Os recursos sugerem o tipo de objeto e, no caso de vários recursos, a ordem de prioridade de seleção é tópico, fila provisória, fila.

Se um recurso não contiver um valor esperado, o recurso padrão será tópico. Qualquer outro recurso será ignorado.

Nota: Alguns clientes AMQP não configuram esses recursos e obterão o comportamento padrão de publicação / assinatura do IBM MQ Por exemplo, o Quarkus Reactive Messaging AMQP 1.0 Connector apenas configura recursos da versão 2.8.0CR1 em diante.

Os canais do AMQP esperam que o `distribution-Mode` na estrutura ATTACH contenha um dos itens a seguir, para uma origem ou destino:

- `mover`
- `copiar`

em que `mover` implica uma obtenção destrutiva e `copiar` implica navegador.

Nota: Se o `distribution-Mode` não for configurado, ou configurado para qualquer outra coisa que não seja `cópia`, `mover` será assumido.

Nomes de links

Os canais AMQP esperam que o nome de um link AMQP siga um destes formatos:

- Um tópico simples (para publicação e assinatura)
  - Publicando mensagens: uma sequência de tópicos simples (por exemplo, um nome de link de "`/sports/football`") faz com que uma mensagem seja publicada no tópico `/sports/football`.
  - Assinando um tópico para receber mensagens: uma sequência de tópicos simples (por exemplo, um nome de link de "`/sports/football`" faz com que uma assinatura seja definida no tópico `/sports/football`.
- Um tópico detalhado privado (para assinatura)
  - Uma sequência de tópicos verbose que descreve uma assinatura privada no formato: "`private:`*`topic string`*" (por exemplo: "`private:/sports/football`"). O comportamento é idêntico a uma sequência de tópicos simples. A declaração `private` diferencia uma assinatura específica de um determinado cliente AMQP de uma assinatura compartilhada entre clientes.
- Um tópico detalhado compartilhado (para assinatura)
  - Uma sequência de tópicos verbose que descreve uma assinatura compartilhada no formato: "`share:`*`share name`*`:`*`topic string`*" (por exemplo: "`share:bbc:/sports/football`").
- Uma fila (para o sistema de mensagens ponto a ponto para produtor e consumidor)

  - Produtor para enviar mensagens; uma sequência de nomes de fila faz com que um produtor envie uma mensagem em uma fila.
  - Consumidor para receber mensagens; uma sequência de nomes de fila faz com que um consumidor receba mensagens de uma fila.
- Em branco (para o sistema de mensagens ponto a ponto em uma fila provisória)
  - Produtor para enviar mensagens em uma fila provisória; em branco faz com que um produtor envie uma mensagem em uma fila provisória.
  - Consumidor para receber mensagens em uma fila provisória; em branco faz com que um consumidor receba mensagens de uma fila provisória.

Para obter mais informações sobre como as mensagens AMQP são mapeadas para e a partir de IBM MQ mensagens consulte "Mapeando campos AMQP para os campos IBM MQ (mensagens recebidas)" na página 698.

Comprimentos máximos para sequência de tópicos, nomes de compartilhamento e identificadores de cliente

A sequência de tópicos, nome de compartilhamento e identificador de cliente devem ser contidos em 10237 bytes. Além disso, o comprimento máximo de um identificador de cliente é 256 caracteres.

Esses comprimentos máximos significam que você pode ter um dos seguintes:

- uma sequência de tópicos muito longa, desde que o nome de compartilhamento seja curto
- um nome de compartilhamento longo, mas uma sequência de tópicos curta

IDs do Contêiner

Os canais do AMQP esperam que o ID do contêiner de um AMQP Open performativo contenha um ID de cliente AMQP exclusivo. O comprimento máximo de um ID de cliente AMQP é de 256 caracteres e o ID pode conter caracteres alfanuméricos, sinal de percentual (%), barra (/), ponto (.) e sublinhado (_).

Sessões

Os canais AMQP suportam somente uma sessão única do AMQP. Um cliente do AMQP que tenta criar mais de uma sessão do AMQP recebe uma mensagem de erro e é desconectado do canal.

Transações

Os canais AMQP não suportam transações do AMQP. Uma estrutura de conexão do AMQP que tenta coordenar uma nova transação ou uma estrutura de transferência do AMQP que tenta declarar que uma nova transação foi rejeitada com uma mensagem de erro.

Estado de Entrega

Os canais AMQP suportam um estado de entrega apenas para os quadros de disposição Aceitos, Liberados ou Modificados. Observe que, quando um estado Modificado é usado, os canais AMQP não suportam a opção undeliverable-here.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Suporte ponto a ponto em canais AMQP

O canal AMQP do IBM MQ fornece suporte para o envio e o recebimento de mensagens das filas.

Os clientes do AMQP, como a biblioteca Apache Qpid™ JMS, solicitam uma capacidade de `queue` ou `temporary-queue` ao enviar a estrutura de anexo do AMQP. As capacidades permitem que o canal AMQP identifique o objeto como uma fila, como uma fila temporária ou como um tópico. Na ausência de

uma capacidade de fila ou de fila temporária ou até mesmo de qualquer uma das capacidades, supõe-se que a solicitação seja para um tópico.

Os canais AMQP do IBM MQ fornecem suporte de tipo de fila para o seguinte:

Envio e recebimento de fila
As mensagens podem ser enviadas para uma fila e consumidas por meio de uma fila. Para o consumo de mensagens, os modos síncrono e assíncrono são suportados.

Procura de mensagens em filas
Além de colocar e obter mensagens de uma fila, também é possível procurar mensagens em uma fila.

Suporte de fila temporária
As mensagens podem ser enviadas para uma fila temporária e consumidas por meio de uma fila temporária. Observe que a exclusão da fila temporária será suportada se o mesmo objeto de fila temporária usado para criar a fila temporária também for usado para excluir a fila temporária.

A SYSTEM.DEFAULT.MODEL.QUEUE é usada ao criar uma fila temporária e o prefixo para a fila temporária será `AMQP.*`.

A SYSTEM.DEFAULT.MODEL.QUEUE é, por padrão, uma fila dinâmica temporária, porém é possível usar a propriedade **`Definition type`** na fila SYSTEM.DEFAULT.MODEL.QUEUE para mudar a fila para que seja uma fila dinâmica permanente.

Fila dinâmica permanente
Uma fila dinâmica permanente é excluída quando um cliente AMQP, como uma biblioteca Apache Qpid JMS, envia uma solicitação com uma estrutura `detach` com o atributo **`closed`** configurado como *true*.

Importante:

Comportamento do Qpid JMS:
Deve-se chamar um comando de API do Qpid JMS, por exemplo, o método `javax.jms.TemporaryQueue.delete()`, para destruir a fila após o uso e esse processo também limpa as mensagens presentes na fila.

Se esse de comando não for emitido, a fila permanecerá com mensagens ainda presentes quando a conexão for encerrada.

-

Fila dinâmica temporária
Uma fila dinâmica temporária é excluída quando o cliente AMQP encerra a conexão.

Importante:

Comportamento do Qpid JMS:
Se você chamar um comando de API do Qpid JMS, por exemplo, o método `javax.jms.TemporaryQueue.delete()`, encerrar a conexão do JMS ou se a conexão for quebrada, a fila será excluída e todas as mensagens serão perdidas.

O fechamento de uma sessão do JMS em si não causa a exclusão da fila temporária, mesmo que a fila temporária possa ter sido criada usando o método `javax.jms.Session.createTemporaryQueue()`.

-

Tarefas relacionadas
Criando e utilizando canais AMQP
Protegendo clientes AMQP

ALW Mapeando campos de mensagens AMQP e IBM MQ

As mensagens AMQP são compostas de um cabeçalho, anotações de entrega, anotações de mensagens, propriedades, propriedades do aplicativo, corpo e rodapé.

As mensagens AMQP são compostas das seguintes partes:

Cabeçalho
O cabeçalho opcional contém cinco atributos fixos da mensagem:

- **duráveis** - especifica os requisitos de durabilidade
- **prioridade** - prioridade da mensagem relativa
- **ttl** - time to live em milissegundos
- **primeiro adquirente** - se isto for verdadeiro, a mensagem não foi adquirida por nenhum outro link
- **contagem de entrega** - o número de tentativas de entrega anteriores, malsucedidas.

Anotações de entrega
Opcional. Especifica os atributos de cabeçalho não padrão da mensagem para diferentes públicos pretendidos. As anotações de entrega transportam informações do ponto de envio para o ponto de recebimento.

Anotações de mensagem
Opcional. Especifica os atributos de cabeçalho não padrão da mensagem para diferentes públicos pretendidos. A seção de anotações de mensagem é usada para as propriedades da mensagem que visam a infraestrutura e devem ser propagadas em cada etapa da entrega.

Propriedades
Opcional. Esta parte é equivalente ao descritor de mensagens do MQ. Ela contém os seguintes campos fixos:

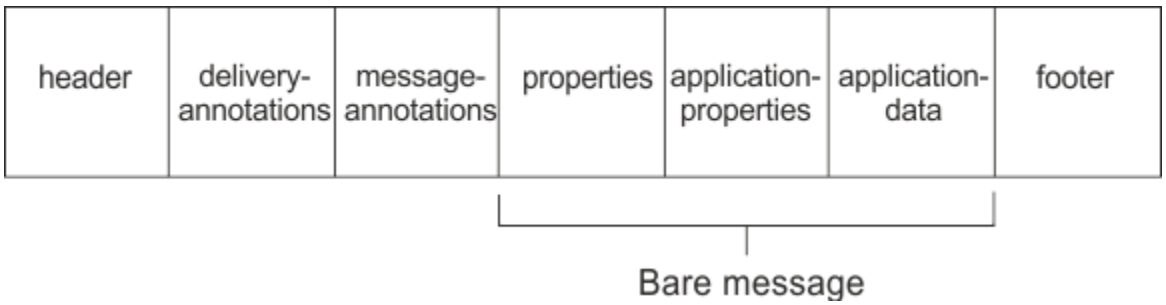
- **ID de mensagem** - identificador de mensagem do aplicativo
- **user-id** - ID do usuário da criação
- **para** - endereço do nó para o qual a mensagem é destinada
- **assunto** - o assunto da mensagem
- **responder para** - o nó para o qual a resposta é enviada
- **ID de correlação** - identificador de correlação do aplicativo
- **tipo de conteúdo** - tipo de conteúdo MIME
- **codificação de conteúdo** - tipo de conteúdo MIME. Usado como um modificador para o tipo de conteúdo.
- **tempo de expiração absoluto** - o tempo em que esta mensagem é considerada expirada
- **tempo de criação** - o momento em que esta mensagem foi criada
- **ID do grupo** - o grupo ao qual pertence esta mensagem
- **sequência do grupo** - o número de sequência desta mensagem dentro de seu grupo
- **ID do grupo de resposta** - o grupo ao qual pertence a mensagem de resposta

Propriedades de aplicativos
Equivalente a propriedades de mensagens do MQ.

Conteúdo
Equivalente à carga útil do usuário do MQ.

Rodapé
Opcional. O rodapé é usado para obter detalhes sobre a mensagem ou entrega que pode ser calculada ou avaliada apenas após a mensagem bare completa ter sido construída ou vista (por exemplo, hashes de mensagens, HMACs, assinaturas e detalhes de criptografia).

O formato da mensagem AMQP é ilustrado na figura a seguir:



As propriedades, propriedades do aplicativo e parte de dados do aplicativo são conhecidas como a mensagem "bare ". Esta é a mensagem conforme enviada pelo emissor e é imutável. O destinatário vê a mensagem inteira, incluindo o cabeçalho, o rodapé, as anotações de entrega e as anotações da mensagem.

Para obter uma descrição integral do formato da mensagem AMQP 1.0, veja o Padrão OASIS em https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Mapeando campos do IBM MQ para campos AMQP (mensagens não enviadas)

Quando uma mensagem IBM MQ é publicada e o IBM MQ a envia para um consumidor AMQP, ela propagará alguns dos atributos da mensagem IBM MQ em atributos equivalentes de mensagem AMQP.

cabeçalho

Um cabeçalho é incluído somente se um dos cinco campos do cabeçalho contiverem um valor não padrão. Apenas os campos com um valor não padrão serão incluídos no cabeçalho. Os cinco campos de cabeçalho serão inicialmente derivados da propriedade `mq_amqp.Hdr` equivalente, se ela estiver configurada e, em seguida, modificados conforme mostrado na tabela a seguir:

Tabela 101. Mapeamentos de campo de cabeçalho

Campo	Valor padrão	Value
durável	false	True se `MQMD.Persistence` estiver configurado para `MQPER_PERSISTENT`, caso contrário, false.
priority	4	Do `mq_amqp.Hdr.Pri` se estiver configurado ou, caso contrário, do `MQMD.Priority` se estiver configurado. Se nenhum estiver configurado, configure para 4.
ttl	n/a	`MQMD.Expiry` em milissegundos. Se o valor de `MQMD.Expiry` for MQEI_UNLIMITED, então configure o valor máximo para o campo ttl do AMQP
first-acquirer	false	Do `mq_amqp.Hdr.Fac`, se configurado, ou false caso contrário.
delivery-count	0	Do `mq_amqp.Hdr.Dct`, se configurado, ou 0 caso contrário.

delivery-annotation

Configure conforme necessário pelo canal AMQP.

message-annotation

Não incluído.

propriedades

As **propriedades** virão sem modificação das propriedades `mq_amqp.Prp` equivalentes se estas estiverem configuradas. Se a mensagem não era originalmente uma mensagem AMQP (ou seja, PutApplType não é MQAT_AMQP), então uma seção de propriedades será gerada conforme descrito na tabela a seguir:

Tabela 102. Mapeamentos de campo propriedades

Nome	**Value**
message-id	O `MQMD.MsgId` é configurado como binário.
id do usuário	O UTF-8 do `MQMD.UserIdentifier` é configurado como binário na rede na ordem do byte.
para	A fila da qual a mensagem foi obtida ou, para uma publicação, a sequência de tópicos.
subject	Não configurado.
reply-to	`MQMD.ReplyToQ` se não estiver em branco, caso contrário, não configurado.
id de correlação	O `MQMD.CorrelId` é configurado como binário se não estiver em branco, caso contrário, não configurado.
content-type	Não configurado.
content-encoding	Não configurado.
absolute-expiry-time	Não configurado.
creation-time	Os campos `MQMD.PutDate` e `MQMD.PutTime` são usados para gerar um registro de data e hora.
group-id	Não configurado.
group-sequence	Não configurado.
reply-to-id	Não configurado.

propriedades- da aplicaão

Todas as propriedades do IBM MQ no grupo "usr" são incluídas como **application-properties**.

corpo

O canal AMQP desempenha um get com conversão para converter a carga útil do IBM MQ UTF-8.

Se a carga útil do IBM MQ não contiver uma mensagem AMQP, a carga útil do IBM MQ será configurada no corpo como uma seção de dados de única sequência para formatar MQFMT_STRING (previsto que a conversão para UTF-8 foi bem-sucedida) ou como uma única seção de dados binários, caso contrário.

Se uma mensagem de formato AMQP estiver incluída, então é configurada como corpo. Quaisquer cabeçalhos IBM MQ (não incluindo as propriedades de mensagens que são retornadas em uma manipulação de mensagem) que precedam a mensagem AMQP são pré-anexados como um valor binário se o corpo for uma sequência AMQP. Caso contrário, os cabeçalhos do IBM MQ serão descartados.

rodapé

Nenhum rodapé está incluído.

Tarefas relacionadas
Criando e utilizando canais AMQP
Protegendo clientes AMQP
Referências relacionadas
MQMD - descritor de mensagem

ALW Mapeando campos AMQP para os campos IBM MQ (mensagens recebidas)

Quando o canal AMQP recebe uma mensagem e a coloca no IBM MQ, ele propaga alguns dos atributos da mensagem AMQP nos atributos de mensagens equivalentes do IBM MQ.

As restrições a seguir se aplicam ao mapear uma mensagem AMQP recebida:

- Se o campo `message-id` ou `correlation-id` na parte de propriedades for um uuid ou um ulong, a mensagem será rejeitada.
- Qualquer `message-annotations` faz com que a mensagem seja rejeitada.
- As seções `delivery-annotations` e `footer` são permitidas, mas não são propagadas para a mensagem IBM MQ

As subseções a seguir mostram a IBM MQ expressão de uma mensagem AMQP.

Descritor de Mensagens

Tabela 103. Descritor de mensagens para a mensagem AMQP

Campo	**Value**
StrucId	MQMD_STRUC_ID
Versão	MQMD_VERSION_1
Relatório	MQRO_NONE
MsgType	MQMT_DATAGRAM
Expiração	O valor obtido do campo `ttl` no cabeçalho da mensagem AMQP
Feedback	MQFB_NONE
Codificação	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
Formato	Consulte Carga útil
Priority	Valor obtido do campo `priority` no cabeçalho da mensagem AMQP. Se configurado, limitado a um máximo de 9. Se não for configurado, leva o valor padrão de 4.
Persistence	Se o campo `durable` no cabeçalho da mensagem AMQP for configurado como true, configure como MQPER_PERSISTENT Caso contrário, configure como MQPER_NOT_PERSISTENT.
MagId	O gerenciador de filas aloca um MsgId de 24 bytes exclusivo.
CorrelId	O valor obtido do campo `correlation-id` nas propriedades AMQP, se configurado. Configure como um valor binário de 24 bytes. Caso contrário, configure como MQCI_NONE/.

Tabela 103. Descritor de mensagens para a mensagem AMQP (continuação)	
Campo	**Value**
BackoutCount	0
ReplyToQ	O valor obtido do campo `reply-to` nas propriedades AMQP, se configurado. Caso contrário, configure como "".
ReplyToQMgr	""
Relatório	O valor derivado de qualquer propriedade de Relatório IBM de JMS configurado nas propriedades do aplicativo AMQP.
UserIdentifier	Configure como o identificador do usuário autenticado que conectou ao canal AMQP
AccountingToken	MQACT_NONE
ApplIdentityData	Sequência hexadecimal. Configure como os últimos 8 bytes do identificador de conexão MQ do canal AMQP.
PutApplType	MQAT_AMQP
PutApplName	
PutDate	Valor obtido do campo `creation-time` das propriedades AMQP, se configuradas. Caso contrário, configure como a data atual.
PutTime	Valor obtido do campo `creation-time` das propriedades AMQP, se configuradas. Caso contrário, configure como a hora atual.
ApplOriginData	""

Propriedades da Mensagem

Existem duas razões para configurar as propriedades de mensagens:

- Permitir que partes da mensagem AMQP fluam pelo gerenciador de filas sem afetar a carga útil da mensagem.
- Permitir a seleção de `application-properties`.

A tabela a seguir mostra as propriedades que são configuradas a partir da mensagem AMQP:

Tabela 104. Propriedades de mensagem AMQP			
Nome da Propriedade	**Nome do MQRFH2**	**tipo**	**Descrição**
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	Uma sequência de identificação para o canal AMQP. Ela é usada para gerar a mensagem, para que as partes interessadas possam informar em qual versão colocar a mensagem (por exemplo, a equipe de serviço ao diagnosticar problemas). O valor não é validado pelo gerenciador de filas e não deve ser documentado externamente.
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	A versão da mensagem AMQP. Se não estiver presente, "1.0" será assumido. O valor não é validado pelo gerenciador de filas.

Tabela 104. Propriedades de mensagem AMQP (continuação)

Nome da Propriedade	Nome do MQRFH2	tipo	Descrição
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	Uma sequência de identificação para a API. Ela é usada para enviar a mensagem AMQP para o canal, para que as partes interessadas possam informar em qual versão colocar a mensagem (por exemplo, a equipe de serviço ao diagnosticar problemas). O valor não é validado pelo gerenciador de filas e não deve ser documentado externamente.
AMQPDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	O valor do campo `durable` no cabeçalho da mensagem AMQP, se configurado..
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	O valor do campo `priority` no cabeçalho da mensagem AMQP, se configurado..
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	O valor do campo `ttl` no cabeçalho da mensagem AMQP, se configurado..
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLEAN	O valor do campo `first-acquirer` no cabeçalho da mensagem AMQP, se configurado..
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	O valor do campo `delivery-count` no cabeçalho da mensagem AMQP, se configurado..
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	O valor do campo `message-id` nas propriedades AMQP, se configuradas como uma sequência.
		MQTYPE_BYTE_STRING	O valor do campo `message-id` nas propriedades AMQP, se configuradas como uma sequência de bytes.
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	O valor do campo `user-id` nas propriedades AMQP, se configuradas.
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	O valor do campo `to` nas propriedades AMQP, se configuradas.
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	O valor do campo `subject` nas propriedades AMQP, se configuradas.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	O valor do campo `reply-to` nas propriedades AMQP, se configuradas.

Tabela 104. Propriedades de mensagem AMQP (continuação)

Nome da Propriedade	Nome do MQRFH2	tipo	Descrição
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	O valor do campo `correlation-id` nas propriedades AMQP, se configuradas como uma sequência.
		MQTYPE_BYTE_STRING	O valor do campo `correlation-id` nas propriedades AMQP, se configuradas como uma sequência de bytes.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	O valor do campo `content-type` nas propriedades AMQP, se configuradas.
AMQPContentEncoding	mq_amqp.Prp.Cneing	MQTYPE_STRING	O valor do campo `content-encoding` nas propriedades AMQP, se configuradas.
AMQPAbsoluteExpiryTime	mq_amqp.Prp.Aet	MQTYPE_STRING	O valor do campo `absolute-expiry-time` nas propriedades AMQP, se configuradas.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	O valor do campo `creation-time` nas propriedades AMQP, se configuradas.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	O valor do campo `group-id` nas propriedades AMQP, se configuradas.
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	O valor do campo `group-sequence` nas propriedades AMQP, se configuradas.
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	O valor do campo `reply-to-group-id` nas propriedades AMQP, se configuradas.

Cada uma das propriedades do aplicativo da mensagem AMQP é configurada como uma propriedade de mensagem do IBM MQ. A seção `application-properties` deve ser reconstituída de forma idêntica byte por byte e, então, as restrições a seguir serão aplicadas:

- Se uma propriedade do aplicativo for rejeitada pelo código de validação MQSETMP, a mensagem será rejeitada. Por exemplo:
  - O nome da propriedade é limitado em comprimento a MQ_MAX_PROPERTY_NAME_LENGTH.
  - O nome da propriedade deve seguir as regras definidas pelo Java Language Specification for Java Identifiers.
  - O nome da propriedade não deve iniciar com JMS ou `usr.`JMS, exceto para as propriedades JMS documentadas que podem ser configuradas.
  - O nome da propriedade não deve ser uma palavra-chave SQL.
- Uma propriedade do aplicativo que contém caractere Unicode U+002E (".") faz com que a mensagem seja rejeitada. A propriedade deve ser exprimível no grupo "usr" de propriedades usadas pelo JMS.
- Apenas as propriedades nula, booleana, byte, curta, int, longa, flutuante, dupla, binária e de sequência são suportadas. Uma propriedade do aplicativo com qualquer outro tipo fará com que a mensagem seja rejeitada.

É possível configurar as propriedades JMS a seguir usando `application-properties`:

- JMS_IBM_REPORT_EXCEPTION
- JMS_IBM_REPORT_EXPIRATION
- JMS_IBM_REPORT_COA
- JMS_IBM_REPORT_COD
- JMS_IBM_REPORT_PAN
- JMS_IBM_REPORT_NAN
- JMS_IBM_REPORT_PASS_MSG_ID
- JMS_IBM_Report_Pass_Correl_ID
- JMS_IBM_REPORT_DISCARD_MSG

Observe que os nomes e valores da propriedade são consistentes com os detalhes do "Mapeamento de campos específicos de provedor do JMS" na página 162 equivalente e que os valores que não são válidos são ignorados.

carga útil

- Para um AMQP body com uma seção de dados binários única, os dados binários (excluindo os bits AMQP) são colocados como a carga útil IBM MQ , com um Formato de MQFMT_NONE.
- Para um AMQP body com uma única seção de dados de sequência, os dados de sequência (excluindo os bits AMQP) são colocados como a carga útil IBM MQ , com um Formato de MQFMT_STRING.
- Caso contrário, o AMQP body formará a carga útil como está, com um Formato de MQFMT_AMQP.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Confiabilidade de entrega de mensagem

Esta seção compara os recursos de confiabilidade para MQ Light API e Apache Qpid JMS.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Confiabilidade da mensagem MQ Light

Existem quatro recursos da API MQ Light que permitem controlar a confiabilidade da entrega de mensagens para, e a partir de, aplicativos AMQP.

São elas:

- "Qualidade de serviço (QOS) da mensagem" na página 702
- "Confirmação automática do assinante" na página 703
- "Tempo de vida de assinatura" na página 703
- "Persistência de mensagem" na página 704

Qualidade de serviço (QOS) da mensagem

O MQ Light API oferece duas qualidades de serviço:

- No máximo uma vez
- Pelo menos uma vez

É possível escolher qual qualidade de serviço você deseja que os publicadores e assinantes usem.

Se você estiver usando um cliente MQ Light, configure a opção cliente ou assinar **qos** para *QOS_AT_MOST_ONCE* ou *QOS_AT_LEAST_ONCE*.

Se você estiver usando um cliente AMQP diferente, configure o atributo **settled** da estrutura de transferência (para publicadores), ou a estrutura de disposição (para assinantes) como *true* ou *falso*, dependendo da qualidade do serviço que você deseja atingir.

A qualidade de serviço determina quando uma mensagem é descartada do lado `sending` de uma conversação:

Publicação

- Se um editor escolher **QOS** *0* (no máximo uma vez) o publicador não aguardará uma confirmação do gerenciador de filas, antes de descartar sua cópia da mensagem. Se a conexão com o gerenciador de filas falhar antes da conclusão do envio, a mensagem poderá não ser recebida pelos assinantes.
- Se um publicador escolher **QOS** *1* (pelo menos uma vez) ele aguardará o gerenciador de filas reconhecer que a mensagem foi gravada em filas de assinantes antes de descartar sua cópia da mensagem. Se a conexão com o gerenciador de filas falhar durante o envio, o publicador reenviará a mensagem depois de ter se reconectado ao gerenciador de filas.

Assinando

- Se um assinante escolher **QOS** *0* o gerenciador de filas não aguardará um reconhecimento do assinante antes de descartar sua cópia da mensagem. Se a conexão com o assinante falhar antes de o assinante receber a mensagem, essa mensagem poderá ser perdida.
- Se um assinante escolher **QOS** *1* o gerenciador de filas aguardará um reconhecimento do assinante antes de descartar sua cópia da mensagem. V 9.4.0 De IBM MQ 9.3.3, as mensagens reconhecidas são removidas em lotes para melhorar o desempenho. Para obter mais informações, consulte “Removendo mensagens AMQP reconhecidas da fila em lotes” na página 705.

  Se a conexão com o assinante falhar antes de o assinante receber a mensagem, a mensagem será mantida pelo gerenciador de filas. O gerenciador de filas reenviará a mensagem para o assinante ao se reconectar, ou para outro assinante, se a assinatura for compartilhada.

Confirmação automática do assinante

Se um assinante escolher **QOS** *1* (pelo menos uma vez) ele deverá reconhecer o recebimento de cada mensagem antes que o gerenciador de filas descarte sua cópia. O assinante pode decidir quando reconhecer mensagens.

Com o **auto-confirm** configurado como *true*, o cliente MQ Light reconhece automaticamente a entrega de cada mensagem, uma vez que ele recebeu com sucesso a mensagem pela rede.

Isso assegura que, se houver uma falha de rede, a mensagem será entregue novamente para o aplicativo. No entanto, ainda é possível que o aplicativo perca a mensagem, no caso de ele falhar entre o reconhecimento da mensagem pelo cliente MQ Light e seu processamento pelo aplicativo.

Com o **auto-confirm** configurado como *false*, o cliente MQ Light não reconhece automaticamente a entrega da mensagem, mas deixa para o aplicativo decidir quando ela deve ser reconhecido.

Isso permite que um aplicativo faça uma atualização para um recurso externo, como um banco de dados ou um arquivo, antes de reconhecer para o gerenciador de filas que a mensagem agora foi processada e pode ser descartada.

Tempo de vida de assinatura

Quando um aplicativo assina, ele escolhe se a assinatura e o destino no qual as mensagens são armazenadas para essa assinatura continuarão a existir após a desconexão do aplicativo.

A MQ Lightopção de assinatura **ttl** é usada para especificar o tempo (em milissegundos) que uma assinatura continua existindo após o aplicativo se desconectar. Se o aplicativo se reconecta antes desse tempo, a assinatura continua e o aplicativo pode continuar a consumir mensagens dessa assinatura.

Se o período de tempo de vida passa sem a reconexão do aplicativo, a assinatura é removida e quaisquer mensagens armazenadas em seu destino são perdidas, mesmo se são mensagens persistentes.

Se é importante não perder mensagens, deve-se especificar um valor de tempo de vida para o aplicativo, que é alto o suficiente para assegurar que as mensagens não sejam perdidas durante uma indisponibilidade.

Persistência de mensagem

A persistência de mensagens é controlada pelos aplicativos de publicação e assinatura e pela configuração de objetos do tópico IBM MQ.

Se o assinante do AMQP usar **QOS** *0* (no máximo uma vez) e criar uma assinatura não durável, o canal do AMQP sempre colocará mensagens não persistentes na fila de assinantes, independentemente das outras opções descritas no texto a seguir.

Observe que, se o gerenciador de filas for interrompido, a assinatura e as mensagens serão perdidas.

Se um publicador do AMQP configurar o cabeçalho do AMQP **durable** para *true*, o canal do AMQP colocará mensagens persistentes nas filas de assinantes.

Se o gerenciador de filas for interrompido por qualquer motivo, as mensagens continuarão disponíveis aos assinantes quando ele for reiniciado.

Se o cabeçalho **durable** não for configurado, o canal do AMQP escolherá a persistência de mensagens publicadas com base no atributo **DEFPSIST** do objeto do tópico IBM MQ relevante.

Por padrão, esse é o SYSTEM.BASE.TOPIC, que usa um atributo **DEFPSIST** de *NÃO* (não persistente).

Atenção: As versões mais recentes do cliente MQ Light não suportam a configuração do cabeçalho durável do AMQP.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Confiabilidade de mensagem do Apache Qpid JMS

Há quatro recursos da biblioteca do Apache Qpid™ JMS que permitem controlar a confiabilidade da entrega de mensagens para e de aplicativos AMQP.

Eles são para:

- “Publicação” na página 704/Produtor para o sistema de mensagens ponto a ponto
  - Expiração da mensagem
  - Persistência de mensagem
- “Assinando” na página 705
  - Durabilidade da assinatura
  - Modo de confirmação de sessão (Aplicável também para o sistema de mensagens ponto a ponto do consumidor)

Publicação

Expiração da mensagem

A configuração do valor de tempo de vida do produtor de mensagem do JMS afeta o tempo de expiração dado às mensagens publicadas por ele.

Assegure-se de que o valor de tempo de vida para um produtor do JMS seja suficientemente grande para que as mensagens sejam consumidas antes de expirarem.

Como alternativa, deixar o valor de tempo de vida desconfigurado evita que a mensagem expire na fila de assinaturas.

Persistência de mensagem

A configuração do modo de entrega do produtor de mensagem do JMS configura a persistência da mensagem do IBM MQ publicada para o tópico especificado.

Assegure-se de usar **DeliveryMode**.PERSISTENT para mensagens que devem ser retidas quando um gerenciador de filas é finalizado ou tem uma indisponibilidade.

Assinando

Durabilidade da assinatura

Os canais AMQP suportam a criação de assinaturas duráveis usando as versões duráveis dos métodos de criação do consumidor do JMS:

- **`createDurableConsumer()`**
- **`createSharedDurableConsumer()`**

Modo de confirmação de sessão

Para garantir que uma mensagem consumida tenha sido totalmente processada antes de ser removida da fila de assinatura do IBM MQ , crie uma sessão do JMS usando o **Session**Modo CLIENT_RECONHEÇO e use o método **`message.acknowledge()`** para reconhecer esta mensagem e quaisquer outros recebidos anteriormente nesta sessão.

Conceitos relacionados

Desenvolvendo aplicativos clientes AMQP

O suporte IBM MQ para APIs de AMQP, permite que um administrador IBM MQ crie um canal de AMQP. Quando ele é iniciado, este canal define um número de porta que aceita conexões de aplicativos clientes AMQP.

V 9.4.0 Removendo mensagens AMQP reconhecidas da fila em lotes

Se um aplicativo AMQP estiver usando a entrega de mensagens QOS_AT_LEAST_ONCE (1), o serviço AMQP aguardará uma confirmação do aplicativo antes de descartar a cópia de uma mensagem que ele mantém após enviar essa mensagem para o aplicativo. A partir do IBM MQ 9.3.3, as mensagens que foram reconhecidas são removidas da fila em lotes, em vez de individualmente, resultando em um desempenho melhorado

Sobre esta tarefa

Antes de IBM MQ 9.4.0, cada mensagem é removida da fila individualmente

Em IBM MQ 9.4.0, é possível usar as duas propriedades de sistema **`com.ibm.mq.AMQP.BATCHSZ`** e **`com.ibm.mq.AMQP.BATCHINT`** para ajustar o processamento de confirmações em lotes para obter desempenho melhorado:

com.ibm.mq.AMQP.BATCHSZ

Esse atributo define o número máximo de reconhecimentos a serem recebidos antes que o serviço AMQP remova mensagens. O número pode estar no intervalo de 1 a 9999. Se um número inválido for configurado ou se o número especificado estiver fora do intervalo, o valor padrão 50 será usado.

O tamanho do lote não afeta a maneira com que as mensagens são transferidas As mensagens são sempre transferidas individualmente, mas são então removidas em um lote após o serviço AMQP receber as confirmações. O tamanho real de um lote pode ser menor que o valor especificado por **`com.ibm.mq.AMQP.BATCHINT`**. Por exemplo, um lote será concluído se o período configurado pelo atributo **`com.ibm.mq.AMQP.BATCHINT`** expirar

com.ibm.mq.AMQP.BATCHINT
Esse atributo define o período de tempo, em milissegundos para o qual o serviço AMQP mantém mensagens confirmadas na fila Se o lote não estiver cheio, ele será limpo após essa duração. É possível especificar qualquer número de milissegundos, de 1 a 999 999 999. O valor padrão é 50. Se você não especificar um valor para esse atributo, o valor padrão 50 será usado.

Notas:

1. Se o serviço AMQP aguarda uma confirmação antes de descartar uma mensagem depende de qual das duas qualidades de serviço a seguir um aplicativo está usando para entrega de mensagem:
   - QOS para QOS_AT_MOST_ONCE (0)

     Se um aplicativo AMQP estiver usando essa qualidade de serviço, ele não reconhecerá as mensagens, portanto, o serviço AMQP descarta as mensagens após enviá-las para o aplicativo sem aguardar uma confirmação.
   - QOS_AT_LEAST_ONCE (1)

     Se um aplicativo AMQP estiver usando essa qualidade de serviço, ele reconhecerá mensagens, portanto, o serviço AMQP mantém uma cópia de cada mensagem após enviá-la para o aplicativo até receber uma confirmação do aplicativo. Se o aplicativo se desconectar ou perder a conexão antes de confirmar a mensagem, a mensagem será disponibilizada para outros aplicativos O serviço AMQP não remove uma mensagem da fila até que ela tenha sido reconhecida
2. MQ Appliance As propriedades de sistema **com.ibm.mq.AMQP.BATCHSZ** e **com.ibm.mq.AMQP.BATCHINT** não são aplicáveis em IBM MQ Appliance Um valor padrão 50 é usado em IBM MQ Appliance.

.

Procedimento

Use as propriedades do sistema **com.ibm.mq.AMQP.BATCHSZ** e **com.ibm.mq.AMQP.BATCHINT** para ajustar o processamento de confirmações em lotes.

A partir do IBM MQ 9.3.3, quando o gerenciador de filas é criado, o arquivo `amqp_java.properties` contém os seguintes valores padrão para as propriedades do sistema:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

Dependendo da taxa de mensagens consumidas, é possível ajustar o processamento de reconhecimentos em lotes para melhorar o desempenho. Um gerenciador de filas migrado não possui essas propriedades no arquivo `amqp_java.properties` Portanto, para um gerenciador de filas migrado, ou se as propriedades não estiverem configuradas, os valores padrão serão usados É possível incluir essas propriedades para ajustar os valores para o desempenho otimizado.

As mensagens reconhecidas são removidas em lotes quando uma das seguintes condições é atendida:

- O número de mensagens reconhecidas atinge **com.ibm.mq.AMQP.BATCHSZ**.
- **com.ibm.mq.AMQP.BATCHINT** é excedido após o início do lote..
- O aplicativo desconecta ou fecha a fila ou tópico antes que as duas condições anteriores sejam satisfeitas.

ALW Topologias para clientes AMQP com o IBM MQ

Topologias de exemplo para ajudá-lo a desenvolver seus clientes AMQP para trabalhar com o IBM MQ.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Clientes AMQP comunicando-se por meio do IBM MQ

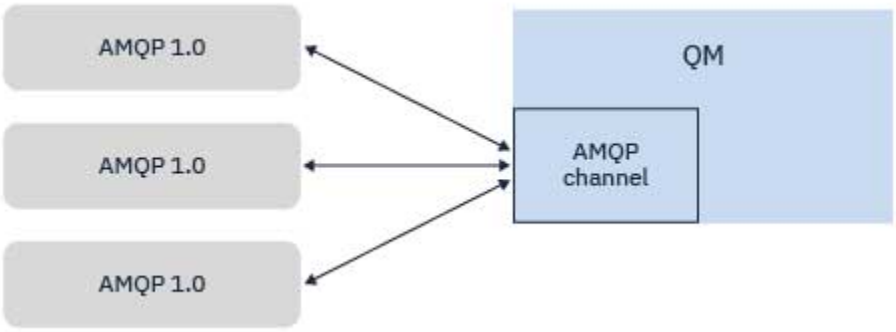
É possível usar IBM MQ como o provedor de mensagens para qualquer aplicativo que esteja em conformidade com o AMQP 1.0. Embora qualquer cliente AMQP 1,0 pode se conectar a um canal AMQP, alguns recursos AMQP não são suportados, por exemplo transações ou múltiplas sessões.

Ao definir um ou mais canais AMQP, os clientes AMQP 1.0 poderão se conectar ao gerenciador de filas e enviar mensagens para uma sequência de tópicos. Os clientes também podem assinar um padrão de tópico para receber mensagens que correspondem ao padrão.

No cenário a seguir, os únicos aplicativos que enviam e recebem mensagens são aplicativos do AMQP 1.0.

Os aplicativos podem escolher se os destinos criados pela assinatura de uma sequência de tópicos são persistentes para que as mensagens não sejam perdidas se o aplicativo perder temporariamente sua conexão com o gerenciador de filas.

Os aplicativos também podem escolher quanto tempo as mensagens serão mantidas antes de serem eliminadas do destino.



Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Clientes AMQP trocando mensagens com aplicativos IBM MQ

Ao definir e iniciar um canal AMQP, os aplicativos AMQP 1.0 podem publicar mensagens que são recebidas por aplicativos MQ existentes. As mensagens publicadas pelo canal AMQP são enviadas para os tópicos MQ e não para as filas MQ. Um aplicativo MQ que criou uma assinatura utilizando a chamada API MQSUB recebe mensagens publicadas por aplicativos AMQP 1.0, desde que a sequência de tópicos ou objeto do tópico usado pelo aplicativo MQ corresponde à sequência de tópicos publicada pelo cliente AMQP.

Dados da mensagem AMQP, atributos e propriedades são configurados na mensagem MQ recebida pelo aplicativo MQ. Para obter mais informações sobre AMQP para mapeamentos de mensagens do MQ , consulte “Mapeando campos AMQP para os campos IBM MQ (mensagens recebidas)” na página 698.

Se o aplicativo MQ tiver criado uma assinatura que é durável, as mensagens publicadas pelo aplicativo AMQP são armazenadas na fila que faz a assinatura. As mensagens são, então, recebidas pelo aplicativo MQ quando o aplicativo retomará sua assinatura. Se o aplicativo AMQP especifica um tempo de mensagem como ativo e o aplicativo MQ não reconectar dentro do tempo de vida, a mensagem será expirada da fila.

Os aplicativos AMQP 1.0 também podem consumir mensagens que são publicadas por aplicativos MQ existentes. As mensagens publicadas pelos aplicativos MQ para um tópico MQ ou sequência de tópicos são recebidas por um aplicativo AMQP 1.0 desde que o aplicativo foi subscrito com um padrão de tópico que corresponde à sequência de tópicos publicados.

Se o aplicativo AMQP 1.0 especifica um valor de tempo de vida para a assinatura e o aplicativo AMQP se desconecta para maior que o tempo de vida, a assinatura será expirada do gerenciador de filas e quaisquer mensagens armazenadas na fila de assinatura são perdidas.

Os campos MQMD, as propriedades da mensagem e os dados do aplicativo são configurados na mensagem AMQP recebida pelo aplicativo AMQP. Para obter mais informações sobre o MQ para mapeamentos de mensagens AMQP, consulte “Mapeando campos do IBM MQ para campos AMQP (mensagens não enviadas)” na página 696.

Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

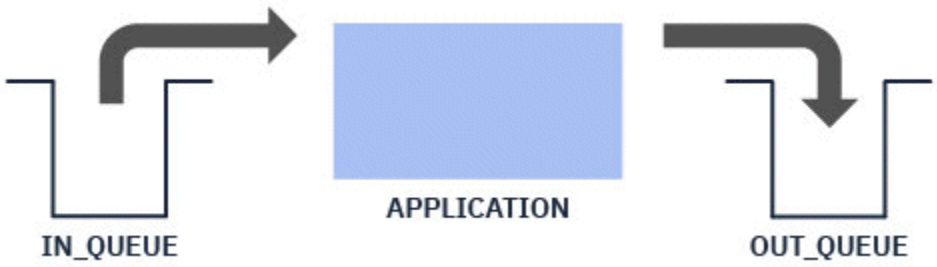
ALW Configurando clientes AMQP para interagir diretamente com aplicativos em filas IBM MQ

A implementação do AMQP IBM MQ suporta publicação / assinatura e ponto a ponto. Para qualquer cliente AMQP que não suporta ponto a ponto, use as etapas a seguir para enviar mensagens para uma fila ou receber mensagens de uma fila..

Visão Geral

Por exemplo, presuma que haja um aplicativo recebendo mensagens de uma fila de entrada `IN_QUEUE` e colocando essas mensagens em uma fila de saída `OUT_QUEUE`. É possível que os clientes AMQP coloquem mensagens em `IN_QUEUE`, e obtenham mensagens de `OUT_QUEUE`

Nota: Não há a necessidade de mudanças para o próprio aplicativo.



Para um publicador AMQP colocar uma mensagem em uma fila, é necessário criar uma assinatura administrativa para a sequência de tópicos na qual o cliente AMQP está publicando, com um destino da fila desejada. Consulte “Enviando mensagens para o aplicativo:” na página 708.

Para um assinante AMQP obter mensagens de uma fila, é necessário substituir a fila por uma fila de alias do mesmo nome, com um destino de um objeto do tópico representando a sequência de tópicos na qual o cliente AMQP está inscrito. Consulte “Obtendo mensagens do aplicativo:” na página 709

Enviando mensagens para o aplicativo:

O aplicativo já está coletando mensagens de `IN_QUEUE` e você quer que um cliente AMQP possa publicar mensagens, de modo que elas vão para essa fila para serem processadas pelo aplicativo.

Para fazer isso, você cria uma nova assinatura administrativa, na qual a sequência de tópicos da qual essa assinatura recebe mensagens é a sequência de tópicos na qual o cliente AMQP publica. A fila de destino dessa assinatura é a fila de entrada para o aplicativo, `IN_QUEUE`.

Quaisquer mensagens que sejam publicadas na sequência de tópicos definidos para essa assinatura administrativa são encaminhadas para o destino definido, neste caso `IN_QUEUE`.

Presumindo que o cliente AMQP publique em uma sequência de tópicos `/application/in`, é possível criar uma assinatura administrativa `APP_IN`, usando o seguinte comando MQSC:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

Quando você tiver definido esse objeto, todas as mensagens publicadas para `/application/in` serão encaminhadas para o destino `IN_QUEUE`, em que são coletadas pelo aplicativo da mesma forma que quaisquer outras mensagens colocadas nessa fila por outros aplicativos.

Obtendo mensagens do aplicativo:

O aplicativo está colocando mensagens em `OUT_QUEUE`, em que elas podem ser coletadas e processadas por outros clientes.

No entanto, nesse caso, você deseja que as mensagens sejam entregues a um cliente AMQP, como alternativa, mas os clientes AMQP usam apenas publicação/assinatura e não podem coletar mensagens diretamente de uma fila.

Para substituir os clientes que estavam recebendo mensagem anteriormente com o cliente AMQP de assinatura, é necessário criar um objeto de tópico para a sequência de tópicos na qual o cliente AMQP está inscrito e uma fila de alias.

Atenção: Se você definir a fila de alias e, em seguida, iniciar o aplicativo de produção antes de qualquer cliente AMQP ter tido uma chance de assinar, as mensagens que o aplicativo de produção enviará para a "fila" (agora um tópico) serão perdidas porque não há assinantes.

As mudanças descritas nesse texto substituem os clientes que estavam recebendo mensagens anteriormente apenas pelo cliente AMQP de assinatura. Para usar uma combinação de AMQP e de outros clientes para obter mensagens, são necessárias mudanças mais amplas.

Presumindo que o cliente AMQP assine uma sequência de tópicos `/application/out`, é possível definir o objeto do tópico `APP_OUT` usando o seguinte comando MQSC:

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

Todas as mensagens entregues a esse objeto do tópico são entregues para o cliente AMQP que assina a mesma sequência de tópicos.

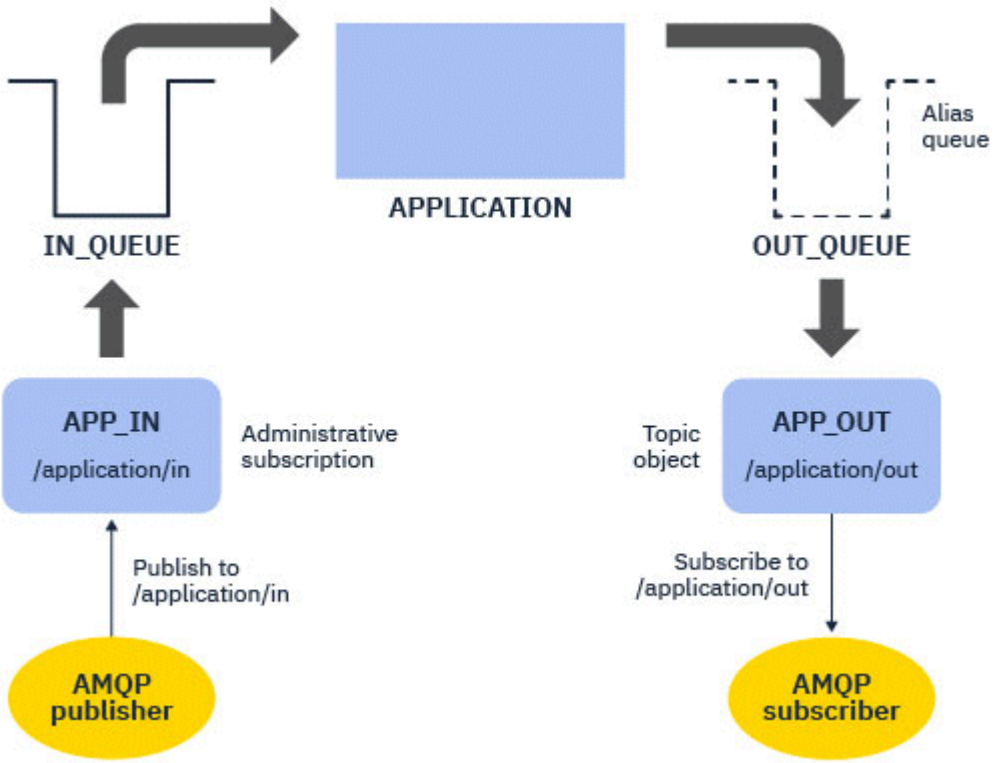
Você então precisa assegurar que as mensagens colocadas em `OUT_QUEUE` pelo aplicativo sejam entregues a esse novo objeto de tópico, para que sejam enviadas para o cliente cadastrado.

Para isso, substitua a fila existente `OUT_QUEUE` por uma fila de alias do mesmo nome, com um tipo de destino do objeto do tópico recém-criado, usando o seguinte comando MQSC:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Agora, as mensagens colocadas pelo aplicativo em `OUT_QUEUE` não esperam na fila para serem coletadas; em vez disso são entregues ao destino dessa fila de alias, ou seja, o novo objeto de tópico `APP_OUT`.

O cliente AMQP, que está inscrito na sequência de tópicos representados por esse objeto do tópico `/application/out`, recebe então quaisquer mensagens enviadas para esse objeto do tópico da fila de alias.



Tarefas relacionadas

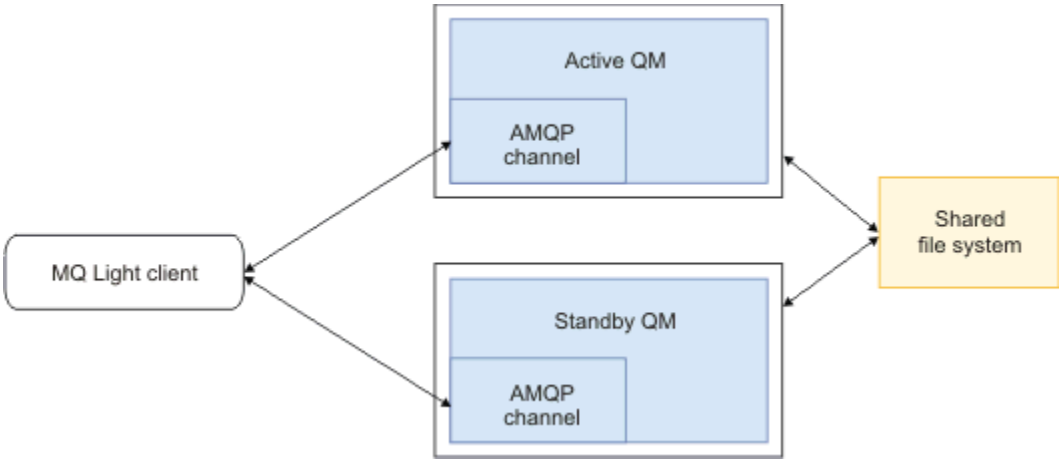
Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Configurando um cliente AMQP para alta disponibilidade

É possível configurar aplicativos AMQP 1.0 para se conectar à instância ativa de um gerenciador de filas de várias instâncias IBM MQ e realizar failover para a instância de espera do gerenciador de filas de várias instância em um par de alta disponibilidade (HA). Para fazer isso, você configura o aplicativo AMQP com dois endereços IP e dois pares de portas.

É possível configurar a API do cliente AMQP com uma função customizada, que será chamada se o cliente perder a conexão com o servidor. A função pode conectar-se a um endereço IP alternativo, por exemplo, um gerenciador de filas de espera do IBM MQ ou ao endereço IP original. Por outros clientes AMQP, se o cliente suportar a configuração de vários terminais de conexão, configure o aplicativo com dois pares de portas do host e use os recursos de reconexão fornecidos pela biblioteca AMQP para alternar para o gerenciador de filas de espera.



Tarefas relacionadas

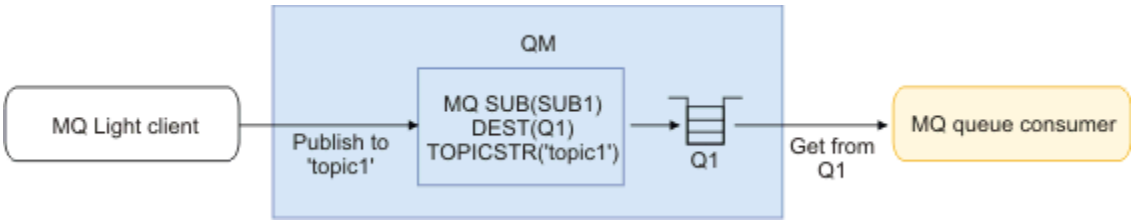
Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Configurando a Publicação / Assinatura para Clientes AMQP

Os clientes AMQP podem publicar em um tópico com uma assinatura do IBM MQ que roteia mensagens para uma fila do IBM MQ lida por um aplicativo existente. Se você desejar que um aplicativo do AMQP 1.0 envie mensagens para um aplicativo IBM MQ existente que é configurado para ler a partir de uma fila, deve definir uma assinatura IBM MQ administrada no gerenciador de filas.

Configure a assinatura para utilizar um padrão de tópico que corresponde à sequência de tópicos utilizada pelo aplicativo AMQP. Configure o destino da assinatura para o nome da fila da qual o aplicativo IBM MQ obtém ou procura mensagens.



Tarefas relacionadas

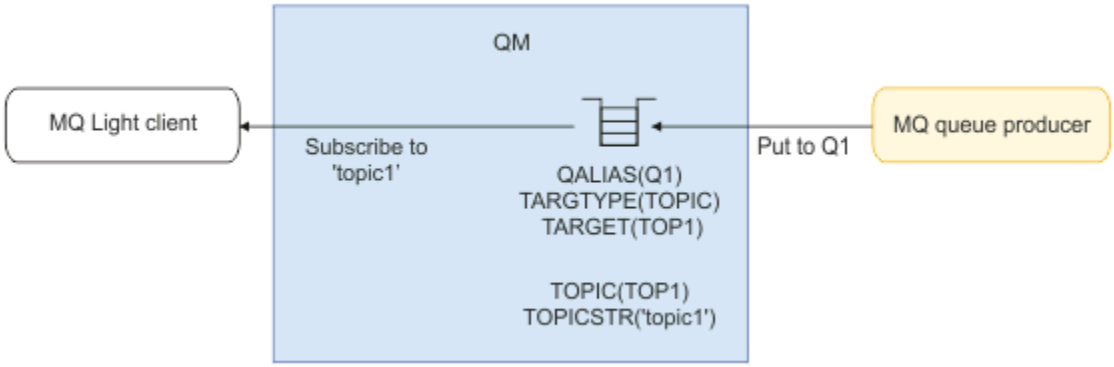
Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Cliente AMQP usando um alias de fila para receber mensagens de um aplicativo IBM MQ

Um cliente AMQP pode assinar um tópico e receber mensagens colocadas em uma fila de alias por um aplicativo IBM MQ. Se você desejar que um aplicativo AMQP 1.0 receba mensagens de um aplicativo IBM MQ existente que seja configurado para colocar mensagens em uma fila, deverá definir um alias de fila (QALIAS) no gerenciador de filas.

O alias da fila deve ter o mesmo nome que a fila que o aplicativo IBM MQ abre para colocação. O alias da fila deve especificar um tipo de base de TÓPICO e um objeto base de um objeto do tópico do IBM MQ que tem uma sequência de tópicos correspondentes ao padrão de tópico assinado pelo aplicativo AMQP.



Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Cliente AMQP submetendo solicitações para e consumindo respostas de um servidor de aplicativos

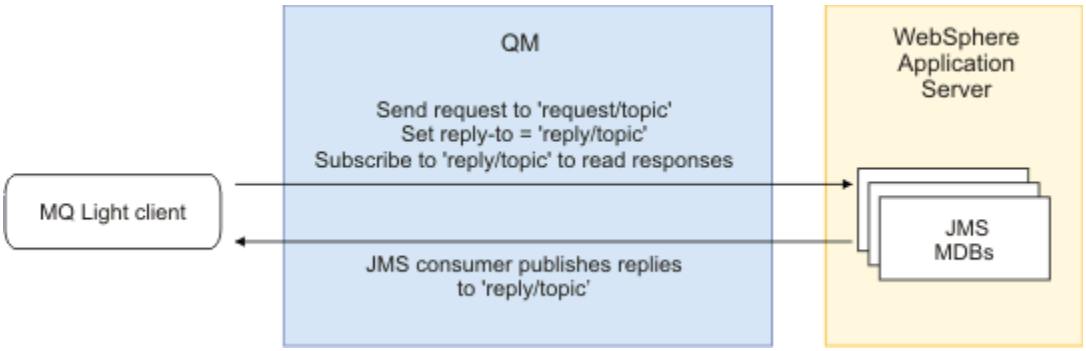
Um cliente AMQP pode enviar solicitações para um bean acionado por mensagens em execução em um servidor de aplicativos e consumir respostas de um tópico de resposta. O IBM MQ suporta aplicativos AMQP 1.0 configurando um tópico de respostas nas mensagens que o IBM MQ publica. Quando uma mensagem AMQP é publicada com o atributo de resposta configurado, o valor do campo de resposta é configurado como uma propriedade JMS para os consumidores JMS receberem. Essa configuração permite que os consumidores JMS leiam o tópico de resposta da mensagem e enviem uma mensagem de resposta de volta para o cliente AMQP.

A propriedade JMS é **JMSReplyTo**. A sequência de resposta do AMQP deve ser um dos tipos a seguir:

- Uma sequência de tópicos. Por exemplo, '`reply/topic`'
- Uma URL de endereço do AMQP no formato `amqp://host:port/[topic-string]`. Por exemplo, `amqp://localhost:5672/reply/topic`

Se você especificar uma URL de endereço do AMQP como o campo resposta, tudo, exceto a sequência de tópicos no final da URL será removido antes de configurar a propriedade **JMSReplyTo**.

Para obter mais informações sobre os mapeamentos de um endereço de resposta AMQP para uma propriedade **JMSReplyTo** , consulte “Mapeando campos AMQP para os campos IBM MQ (mensagens recebidas)” na página 698



Tarefas relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ALW Interoperabilidade entre os aplicativos MQ Light e Apache Qpid JMS

Os aplicativos MQ Light e Apache Qpid JMS funcionam de maneiras semelhantes e, ao se inscreverem a um tópico, criam assinaturas do IBM MQ que seguem a mesma convenção de nomenclatura.

Assinatura privada não compartilhada

O nome da assinatura do IBM MQ criada pelo aplicativo é `:private:<clientid>:<topicstring>`.

Um aplicativo que usa um ID do cliente diferente não consegue acessar as assinaturas criadas por outros aplicativos, porque o nome da assinatura é gerado automaticamente e inclui o ID do cliente AMQP.

Ambos os aplicativos Apache Qpid JMS e MQ Light usam essa convenção de nomenclatura para assinaturas privadas.

Assinaturas compartilhadas globalmente

O nome de uma assinatura do IBM MQ compartilhada globalmente criada por um cliente AMQP é `:share:<sharename>:<topicstring>`.

Se vários aplicativos com IDs de clientes AMQP diferentes especificarem o mesmo nome de compartilhamento e sequência de tópicos, eles compartilharão uma única assinatura e poderão trabalhar juntos para processar as mensagens para essa assinatura. Será possível usar esse padrão se você quiser escalar o número de aplicativos do trabalhador que drenam mensagens de uma assinatura.

Ambos os aplicativos Apache Qpid JMS e MQ Light usam essa convenção de nomenclatura para assinaturas globalmente compartilhadas. No caso do Apache Qpid JMS, isso requer que a conexão JMS não tenha um ID de cliente especificado nela.

A biblioteca do Apache Qpid JMS gera um ID de cliente AMQP automaticamente, mas este ID do cliente não é usado para os propósitos de nomenclatura de assinatura do IBM MQ.

Nota: As assinaturas compartilhadas globalmente ainda têm escopo definido para um gerenciador de filas individual.

Assinaturas compartilhadas privadas

O nome de uma assinatura do IBM MQ compartilhada de forma privada criada por um cliente AMQP é `:privateshare:<clientid>:<sharename>:<topicstring>`.

Se vários encadeamentos de um único aplicativo Apache Qpid JMS usarem o mesmo nome de compartilhamento e sequência de tópicos e um ID do cliente tiver sido configurado na conexão JMS, esses encadeamentos compartilharão o mesmo objeto de assinatura do IBM MQ.

No entanto, outras conexões do Apache Qpid JMS não são capazes de compartilhar a assinatura porque eles devem usar um ID de cliente diferente.

Os clientes MQ Light não suportam o conceito de assinaturas compartilhadas privadas e não podem consumir mensagens de uma assinatura compartilhada privada criada por um aplicativo Apache Qpid JMS.

Assinaturas do IBM MQ JMS

As assinaturas do IBM MQ JMS usam um esquema de nomenclatura diferente dos canais AMQP. Não é possível para os aplicativos MQ Light ou Apache Qpid JMS compartilharem assinaturas com os aplicativos IBM MQ JMS.

Conceitos relacionados

Desenvolvendo aplicativos clientes AMQP
O suporte IBM MQ para APIs de AMQP, permite que um administrador IBM MQ crie um canal de AMQP. Quando ele é iniciado, este canal define um número de porta que aceita conexões de aplicativos clientes AMQP.

ALW IBM MQ Propriedades de controle do listener AMQP

Para melhor desempenho em um aplicativo multiencadeado, é possível ajustar o número de encadeamentos do trabalhador que o serviço AMQP deve usar configurando uma propriedade no arquivo de propriedades AMQP.

É possível configurar as propriedades do serviço do listener AMQP nos seguintes arquivos de propriedades:

- Windows Em Windows sistemas: `amqp_win.properties` .
- Linux AIX Em AIX and Linux sistemas: `amqp_unix.properties` .

As propriedades que podem ser configuradas são as seguintes:

Tabela 105. Propriedades do serviço do listener AMQP	
Propriedade	**Descrição**
com.ibm.mq.MQXR.Workers	O número de encadeamentos do trabalhador do servidor que o serviço do listener do AMQP cria Se esse valor não for especificado, ele será padronizado como igual ao número de processadores lógicos no sistema.
MQIBindType	O tipo de ligação para o serviço AMQP: FASTPATH, SHAREDou ISOLADO. O padrão é FASTPATH.

O serviço do listener AMQP equilibra a carga de trabalho de conexão do cliente em vários encadeamentos do trabalhador. O número de encadeamentos do trabalhador que o serviço AMQP deve usar pode ser especificado usando a propriedade **com.ibm.mq.MQXR.Workers**

O administrador do gerenciador de fila do IBM MQ pode ajustar o número de encadeamentos do trabalhador para um melhor desempenho em um aplicativo multiencadeado Geralmente, o melhor desempenho é obtido quando o número de encadeamentos do trabalhador corresponde ao número de processadores lógicos no sistema No entanto, esse pode não ser sempre o caso para determinadas configurações de máquina e características de carregamento do cliente, portanto, um elemento de ajuste pode ser necessário para localizar o valor ideal para o número de encadeamentos do trabalhador...

Antes de ajustar, certifique-se de entender completamente a natureza de seus aplicativos clientes e suas cargas de trabalho. Medir o desempenho de seu aplicativo com diferentes contagens de encadeamento e benchmarking deve ajudar a determinar o valor ideal para o número de encadeamentos do trabalhador.

Nota: MQ Appliance Essas propriedades não são aplicáveis no IBM MQ Appliance Os valores padrão são usados em IBM MQ Appliance

Desenvolvendo aplicativos REST com o IBM MQ

É possível desenvolver aplicativos REST para enviar e receber mensagens. O IBM MQ suporta diferentes APIs de REST, dependendo da plataforma e capacidade.

As opções a seguir são as opções suportadas pelo IBM MQ das quais você pode escolher para enviar e receber mensagens do IBM MQ:

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

IBM MQ messaging REST API

É possível usar o messaging REST API para enviar, receber e procurar mensagens do IBM MQ em formato de texto sem formatação. O messaging REST API é ativado por padrão.

O suporte é fornecido para vários cabeçalhos de HTTP diferentes que podem ser usados para configurar propriedades de mensagens comuns.

O messaging REST API é totalmente integrado com a segurança do IBM MQ. Para usar o messaging REST API, os usuários devem ser autenticados para o servidor do mqweb e devem ser um membro da função MQWebUser.

Veja informações adicionais na publicação “Sistema de mensagens usando a REST API” na página 716. Consulte também Tutorial: introdução à REST API do sistema de mensagens do IBM MQ no IBM Developer, que inclui exemplos de Go e Node.js para uso da API de REST do sistema de mensagens.

IBM z/OS Connect EE

IBM z/OS O Connect EE é um produto z/OS que permite construir APIs REST em cima de ativos z/OS existentes, como transações CICS ou IMS, e filas e tópicos IBM MQ. O ativo existente do z/OS é ocultado do usuário. Isso permite que o REST ative os ativos sem mudá-los ou qualquer um dos aplicativos existentes que os usam.

O IBM z/OS Connect EE fornece transformação automática de dados para converter entre os dados JSON usados pelas APIs de REST e as estruturas de linguagem mais tradicionais, por exemplo COBOL, esperados por muitos aplicativos de mainframe.

O kit de ferramentas de API do IBM z/OS Connect EE baseado no Eclipse pode ser usado para construir uma API RESTful abrangente que faz uso de parâmetros de consulta e de segmentos de caminho de URL, manipulando o formato JSON à medida que flui pelo tempo de execução do IBM z/OS Connect EE.

O IBM z/OS Connect EE pode ser usado para expor filas e tópicos do IBM MQ como APIs RESTful por meio do provedor de serviços do IBM MQ. Dois tipos de serviços diferentes são suportados:

- Serviços unidirecionais: eles fornecem uma API de REST que permite que uma única operação do IBM MQ seja executada em uma fila ou tópico. Dependendo da configuração exata, uma solicitação de HTTP pode resultar em uma mensagem sendo enviada para uma fila ou publicada em um tópico; ou uma solicitação de HTTP pode resultar em uma mensagem sendo recebida destrutivamente de uma fila
- Serviços bidirecionais: fornecem uma API de REST em um par de filas usadas por um aplicativo de estilo de solicitação-resposta de back-end. Os responsáveis pela chamada emitem uma solicitação de HTTP para o serviço bidirecional. A carga útil da solicitação de HTTP é transformada de JSON para uma estrutura de linguagem tradicional e colocada em uma fila de solicitações na qual é processada pelo aplicativo back-end e uma resposta é colocada na fila de resposta. Essa resposta é recuperada pelo serviço, convertida da estrutura de linguagem tradicional para JSON e enviada de volta para o responsável pela chamada como o corpo de resposta POST.

Para mais informações sobre o Connect EE IBM z/OS, consulte z/OS Connect EE.

Para obter informações adicionais sobre o provedor de serviços IBM MQ, consulte Usando o provedor de serviços IBM MQ.

IBM Integration Bus

IBM Integration Bus é a tecnologia de integração líder da IBMque pode ser usada para conectar aplicativos e sistemas, independentemente dos formatos de mensagens e protocolos que eles suportam.

O IBM Integration Bus sempre suportou o IBM MQ e fornece os nós *HTTPInput* e *HTTPRequest* que podem ser usados para construir uma interface RESTful em cima do IBM MQ e muitos outros sistemas, como bancos de dados.

O IBM Integration Bus pode ser usado para fazer muito mais do que fornecer uma interface REST simples em cima do IBM MQ. Seus recursos podem ser usados para fornecer manipulação de carga útil avançada, enriquecimento de carga útil e muitos outros aprimoramentos como parte de uma REST API.

Para obter informações adicionais, veja a amostra de tecnologia que expõe um JSON sobre interface REST em cima de um aplicativo IBM MQ que espera uma carga útil XML.

DataPower

O DataPower Gateway é um gateway único de diversos canais que ajuda a fornecer segurança, controle, integração e acesso otimizado para uma gama de sistemas, incluindo o IBM MQ. Ele vem nos fatores de forma de hardware e virtual.

Um dos serviços que o DataPower fornece é um gateway multiprotocolo que pode tomar a entrada em um protocolo e gerar a saída em um protocolo diferente. Em particular, o DataPower pode ser configurado para aceitar dados HTTP(S) e roteá-los para o IBM MQ sobre uma conexão do cliente, os quais podem ser usados para construir uma interface REST em cima do IBM MQ. Outros serviços do DataPower, como transformação, também podem ser usados para aprimorar a interface REST.

Para obter informações adicionais, veja Gateway multiprotocolo.

Sistema de mensagens usando a REST API

É possível usar o messaging REST API para executar o sistema de mensagens simples ponto a ponto e de publicação simples. É possível publicar mensagens para um tópico, enviar mensagens para uma fila, procurar mensagens em uma fila e obter mensagens destrutivamente de uma fila. As informações são enviadas e recebidas da messaging REST API no formato de texto sem formatação.

Antes de começar

Nota:

- O messaging REST API é ativado por padrão. É possível desativar a messaging REST API para evitar todo o sistema de mensagens. Para obter mais informações sobre como ativar ou desativar a messaging REST API, consulte Configurando a messaging REST API.
- A messaging REST API é integrada com a segurança do IBM MQ. Para usar o messaging REST API, os usuários devem ser autenticados para o servidor do mqweb e devem ser um membro da função MQWebUser. O usuário também deve estar autorizado para acessar a fila especificada ou o tópico especificado. Para obter mais informações sobre segurança para o REST API, consulte Segurança do IBM MQ Console e do REST API.
- Se você usar o Advanced Message Security (AMS) com o messaging REST API, observe que todas as mensagens são criptografadas usando o contexto do servidor mqweb, não o contexto do usuário que posta a mensagem.
- Ao receber ou procurar uma mensagem, apenas as mensagens formatadas IBM MQ `MQSTR` ou JMS `TextMessage` são suportadas Subsequentemente, todas as mensagens são recebidas destrutivamente sob o ponto de sincronização e quaisquer mensagens não manipuladas são deixadas na fila. A fila do IBM MQ pode ser configurada para mover essas mensagens suspeitas para um destino alternativo. Veja informações adicionais na publicação “Manipulando mensagens suspeitas no IBM MQ classes for JMS” na página 238.
- A messaging REST API não fornece uma entrega única de mensagens com suporte transacional. Se um HTTP POST for emitido e a conexão falhar antes que uma resposta HTTP seja recebida pelo cliente, o cliente não poderá informar imediatamente se a mensagem foi enviada para a fila especificada ou publicada para o tópico especificado Se um HTTP DELETE for emitido e a conexão falhar antes que uma resposta HTTP seja recebida pelo cliente, uma mensagem poderá ter sido obtida destrutivamente da fila e perdida, pois não haverá como reverter a obtenção destrutiva.
- Em IBM MQ 9.3.0, as novas linhas em sequências recebidas não são mais removidas pela operação HTTP POST. Os aplicativos REST que usam versões anteriores não devem usar novas linhas em mensagens enviadas ou publicadas usando a API REST, pois elas serão perdidas.

Procedimento

- “Introdução ao messaging REST API” na página 717
- “Usando o messaging REST API” na página 719
- REST API manipulação de erros
- DescobertaREST API
- Suporte ao Idioma NacionalREST API

Referências relacionadas

Referência do sistema de mensagens da REST API

Informações relacionadas

Tutorial: introdução à REST API do sistema de mensagens do IBM MQ

Introdução ao messaging REST API

Compre rapidamente com a messaging REST API e experimente alguns comandos de exemplo usando cURL.

Antes de começar

Para começar a usar a messaging REST API, os exemplos nesta tarefa possuem os requisitos a seguir:

- Os exemplos usam cURL para enviar solicitações REST para colocar e obter mensagens de uma fila. Portanto, para concluir esta tarefa, será necessário ter o cURL instalado no sistema.
- Os exemplos usam um gerenciador de filas `QM1`. Crie um gerenciador de filas com o mesmo nome ou substitua um gerenciador de filas existente no sistema. O gerenciador de filas deve estar na mesma máquina que o servidor mqweb.
- Para concluir essa tarefa, deve-se ser um usuário com determinados privilégios para que seja possível usar o comando **dspmqweb**:
  - z/OS No z/OS, deve-se ter autoridade para executar o comando **dspmqweb** e acesso de gravação ao arquivo `mqwebuser.xml`.
  - Multi Em todos os outros sistemas operacionais, deve-se ser um usuário privilegiado.

  IBM i Em IBM i, os comando devem estar em execução em QSHELL

Procedimento

1. Assegure-se de que o servidor mqweb esteja configurado para a messaging REST API:
   - Assegure-se de que você tenha configurado o servidor mqweb para uso pelo administrative REST API, o administrative REST API para MFT, o messaging REST APIou IBM MQ Console.Para obter mais informações sobre como configurar o servidor mqweb com um registro básico, consulte Configuração básica para o servidor mqweb
   - Se o servidor mqweb já estiver configurado, certifique-se de ter incluído os usuários apropriados para ativar o sistema de mensagens na etapa 5 de Configuração básica para o servidor mqweb.
     - Se **mqRestMessagingAdoptWebUserContext** for configurado como `true` na configuração do servidor mqweb, os usuários do messaging REST API deverão ser um membro da função `MQWebUser` As funções `MQWebAdmin` e `MQWebAdminRO` não são aplicáveis à messaging REST API. Os usuários também devem estar autorizados para acessar filas e tópicos que são usados para o sistema de mensagens por meio do OAM ou do RACF
     - Se **mqRestMessagingAdoptWebUserContext** for configurado como `false` na configuração do servidor mqweb, o ID do usuário que é usado para iniciar o servidor mqweb deverá ser autorizado a acessar filas usadas para o sistema de mensagens por meio do OAM ou do RACF.
2. z/OS

   Em z/OS, configure a variável de ambiente WLP_USER_DIR para que seja possível usar o comando **dspmqweb** . Configure a variável para apontar para a configuração do servidor mqweb inserindo o comando a seguir:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   em que *`WLP_user_directory`* é o nome do diretório transmitido para `crtmqweb`. Por exemplo:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   Para obter mais informações, consulte Criando o servidor mqweb.
3. Determinar oREST API URL digitando o seguinte comando:

   ```
   dspmqweb status
   ```

Os exemplos nas etapas a seguir supõem que sua URL REST API é a URL padrão `https://localhost:9443/ibmmq/rest/v2/`. Se a URL for diferente do padrão, substitua-a nas etapas a seguir.

4. Crie uma fila, MSGQ, no gerenciador de filas QM1. Esta fila é utilizada para o sistema de mensagens. Use um dos métodos a seguir:
   - Use uma solicitação POST no recurso `mqsc` do administrative REST API, autenticando como o usuário `mqadmin`:

     ```
     curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/
     json" --data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\":
     \"qlocal\",\"name\": \"MSGQ\"}"
     ```

   - Usar comandos do MQSC:

     z/OS No z/OS, use uma origem 2CR em vez do comando **runmqsc**. Para obter mais informações, consulte Origens das quais é possível emitir comandos MQSC e PCF em IBM MQ for z/OS.

     a. Inicie **runmqsc** para o gerenciador de filas inserindo o comando a seguir:

        ```
        runmqsc QM1
        ```

     b. Use o comando MQSC **DEFINE QLOCAL** para criar a fila:

        ```
        DEFINE QLOCAL(MSGQ)
        ```

     c. Saia do **runmqsc** inserindo o comando a seguir:

        ```
        end
        ```

5. Conceda autoridade para o usuário que você incluiu no `mqwebuser.xml`, na etapa 5 de Configuração básica para o servidor mqweb para acessar a fila MSGQ. Substitua seu usuário, em que o `myuser` é usado:
   - z/OS No z/OS:

     a. Conceda a seu usuário acesso à fila:

        ```
        RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
        PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
        ```

     b. Conceda ao ID de usuário da tarefa iniciada mqweb acesso para configurar todo o contexto na fila:

        ```
        RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
        PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
        ```

   - Multi Em todos os outros sistemas operacionais, se o seu usuário estiver no grupo mqm, a autoridade já está concedida. Caso contrário, insira os comandos a seguir:

     a. Inicie **runmqsc** para o gerenciador de filas inserindo o comando a seguir:

        ```
        runmqsc QM1
        ```

     b. Use o comando MQSC **SET AUTHREC** para fornecer a seu usuário as autoridades de procura, consulta, obtenção e colocação na fila:

        ```
        SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
        PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
        ```

c. Saia do **runmqsc** inserindo o comando a seguir:

```
end
```

6. Coloque uma mensagem com o conteúdo `Hello World!` na fila `MSGQ`, no gerenciador de filas `QM1`, ao usar uma solicitação POST no recurso `message`. Substitua seu ID de usuário e senha do `mqwebuser.xml` para `myuser` e `mypassword`:

   A autenticação básica é usada, e um cabeçalho HTTP `ibm-mq-rest-csrf-token` com um valor arbitrário é configurado na solicitação de REST cURL. Esse cabeçalho adicional é necessário para as solicitações POST, PATCH e DELETE.

   ```
   curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X
   POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
   plain;charset=utf-8" --data "Hello World!"
   ```

7. Destrutivamente, obtenha a mensagem de fila `Hello World!` na fila `MSGQ` no gerenciador de filas `QM1`, usando uma solicitação DELETE no recurso `message`. Substitua seu ID de usuário e senha do `mqwebuser.xml` para `myuser` e `mypassword`:

   ```
   curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
   -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
   ```

   A mensagem `Hello World!` é devolvida.

Como proceder a seguir

- Os exemplos usam autenticação básica para proteger a solicitação. Em vez disso, é possível usar autenticação baseada em token ou autenticação baseada em cliente. Para obter mais informações, consulte Usando a autenticação de certificado de cliente com o REST API e o IBM MQ Consolee Usando a autenticação baseada em token com o REST API.
- Saiba mais sobre como usar o messaging REST API e construir URLs com parâmetros de consulta: “Usando o messaging REST API” na página 719.
- Quando você usa o messaging REST API, as conexões com o gerenciador de filas são agrupadas para otimizar o desempenho. É possível configurar o tamanho máximo do conjunto e qual ação será executada quando todas as conexões no conjunto estiverem em uso: Configurando o messaging REST API.
- Procure, nas informações de referência, os recursos disponíveis da messaging REST API e todos os parâmetros de consulta disponíveis: Referência do messaging REST API.
- Descubra a administrative REST API, uma interface RESTful para administração do IBM MQ: Administração usando a REST API.
- Descubra o IBM MQ Console, uma GUI baseada no navegador: Administração usando o IBM MQ Console.

Usando o messaging REST API

Ao usar o messaging REST API, você chama métodos de HTTP em URLs para enviar e receber as mensagens do IBM MQ. O método de HTTP, por exemplo, POST, representa o tipo de ação a ser executado no objeto representado pela URL. Informações adicionais sobre a ação podem ser codificadas em parâmetros de consulta. Informações sobre o resultado da execução da ação podem ser retornadas como o corpo da resposta HTTP.

Antes de começar

Considere estas coisas antes de usar o messaging REST API:

- Você deve se autenticar no servidor mqweb para usar o messaging REST API. É possível autenticar usando a autenticação básica HTTP, a autenticação por certificado de cliente ou autenticação baseada em token. Para obter mais informações sobre como usar esses métodos, consulte IBM MQ Console e REST API security.

- A REST API faz distinção entre maiúsculas e minúsculas. Por exemplo, um HTTP POST na URL a seguir resultará em um erro se o gerenciador de filas for chamado `qmgr1`.

  ```
  /ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- V 9.4.0 Se você estiver se conectando a um gerenciador de filas remotas com o messaging REST API, deverá usar o nome exclusivo para a conexão do gerenciador de filas em vez do nome do gerenciador de filas.
- Nem todos os caracteres que podem ser usados em nomes de objetos IBM MQ podem ser codificados diretamente em uma URL. Para codificar esses caracteres corretamente, deve-se usar a codificação de URL apropriada:
  - Uma barra deve ser codificada como %2F.
  - Um sinal de percentual deve ser codificado como %25.
  - Um período deve ser codificado como %2E.
  - Um ponto de interrogação deve ser codificado como %3F.
- Ao receber ou procurar uma mensagem, apenas mensagens formatadas IBM MQ `MQSTR` e `JMS TextMessage` são suportadas. Subsequentemente, todas as mensagens são recebidas destrutivamente sob o ponto de sincronização e quaisquer mensagens não manipuladas são deixadas na fila. A fila do IBM MQ pode ser configurada para mover essas mensagens suspeitas para um destino alternativo. Veja informações adicionais na publicação “Manipulando mensagens suspeitas no IBM MQ classes for JMS” na página 238.

Sobre esta tarefa

Ao usar a REST API para executar uma ação do sistema de mensagens em um objeto de fila do IBM MQ, você primeiro precisa construir uma URL para representar esse objeto. Cada URL se inicia com um prefixo, que descreve o nome do host e a porta para os quais enviar a solicitação. O restante da URL descreve um objeto específico ou uma rota para esse objeto, conhecido como recurso.

A ação do sistema de mensagens a ser executada no recurso define se a URL precisa de parâmetros de consulta ou não. Ela também define o método de HTTP usado e se informações adicionais são enviadas para a URL ou retornadas dela. As informações adicionais podem formar parte da solicitação de HTTP ou serem retornadas como parte da resposta HTTP.

Depois de construir a URL, é possível enviar a solicitação de HTTP para o IBM MQ. É possível enviar a solicitação usando a implementação HTTP que é construída na linguagem de programação de sua escolha. Também é possível enviar a solicitação usando ferramentas de linha de comandos, como cURL, um navegador da web ou o complemento de navegador da web.

Importante: Deve-se, no mínimo, realizar as etapas “1.a” na página 720 e “1.b” na página 720.

Procedimento

1. Construa a URL:
   a) Determine a URL do prefixo inserindo o comando a seguir:

      ```
      dspmqweb status
      ```

      A URL que você deseja usar inclui a frase `/ibmmq/rest/`.

   b) Inclua a fila e os recursos do gerenciador de filas associados a serem usados para o sistema de mensagens no caminho da URL.

      Na referência de mensagens, os segmentos de variáveis podem ser identificados na URL pelas chaves que o cercam *{ }*. Para mais informações, consulte /messaging/qmgr/{qmgrName}/queue/{queueName}/mensage.

Por exemplo, para interagir com a fila *Q1* associada ao gerenciador de filas *QM1*, inclua `/qmgr` e `/queue` na URL de prefixo para criar a seguinte URL:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
```

Sugestão: V 9.4.0 Se o gerenciador de fila for um gerenciador de fila remoto, deve-se usar o nome exclusivo para o gerenciador de filas no lugar do nome do gerenciador de fila O gerenciador de filas remotas deve ser configurado antes que ele possa ser utilizado com o messaging REST API Para obter mais informações, consulte "Configurando um gerenciador de filas remotas para usar com o messaging REST API" na página 721.

c) Opcional: Inclua um parâmetro de consulta opcional na URL.

Inclua um ponto de interrogação, ?, parâmetro de consulta, sinal de igual =, e um valor para a URL.

Por exemplo, para aguardar por um máximo de 30 segundos para a próxima mensagem se tornar disponível, crie a URL a seguir:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) Opcional: Inclua parâmetros de consulta opcionais adicionais na URL.

Inclua um e comercial, &, na URL e, em seguida, repita a etapa 1c.

2. Chame o método de HTTP relevante na URL. Especifique qualquer carga útil da mensagem opcional e forneça as credenciais de segurança apropriadas para autenticar. Por exemplo:
   - Use a implementação HTTP/REST de sua linguagem de programação escolhida.
   - Use uma ferramenta como um complemento do navegador do cliente REST ou cURL.

V 9.4.0 Configurando um gerenciador de filas remotas para usar com o messaging REST API

É possível usar o messaging REST API para se conectar aos gerenciadores de filas remotas para o sistema de mensagens Antes de poder se conectar a um gerenciador de filas remotas, você deve configurar a configuração do gerenciador de filas remotas. Em seguida, é possível conectar-se ao gerenciador de filas remotas usando o nome exclusivo definido nas informações de configuração.

Antes de começar

- Assegure-se de que você tenha configurado o servidor mqweb para uso pelo administrative REST API, o administrative REST API para MFT, o messaging REST APIou IBM MQ Console.Para obter mais informações sobre como configurar o servidor mqweb com um registro básico, consulte Configuração básica para o servidor mqweb
- Se o servidor mqweb já estiver configurado, assegure-se de que tenha incluído os usuários apropriados para ativar o sistema de mensagens na etapa 5 de Configuração básica para o servidor mqweb.
  Os usuários do messaging REST API devem ser um membro da função `MQWebUser` As funções `MQWebAdmin` e `MQWebAdminRO` não são aplicáveis à messaging REST API.
  - Se **`mqRestMessagingAdoptWebUserContext`** for configurado como `true` na configuração do servidor mqweb, os usuários na função `MQWebUser` deverão estar autorizados a acessar filas e tópicos que são usados para o sistema de mensagens por meio do OAM ou do RACF
  - Se **`mqRestMessagingAdoptWebUserContext`** for configurado como `false` na configuração do servidor mqweb, o ID do usuário que é usado para iniciar o servidor mqweb deverá estar autorizado a acessar filas e tópicos que são usados para o sistema de mensagens por meio do OAM ou RACF.
- Assegure-se de que o messaging REST API esteja configurado para conectar a gerenciadores de filas remotas. Para obter mais informações, consulte Configurando o modo de conexão para messaging REST API.

Sobre esta tarefa

É possível conectar a gerenciadores de filas remotas usando o messaging REST API. Um gerenciador de filas remotas pode ser um gerenciador de filas em outro sistema, um gerenciador de filas em outra instalação, ou um gerenciador de filas na mesma instalação que o servidor mqweb

Para se conectar a um gerenciador de filas remotas, deve-se concluir as etapas de configuração a seguir:

- Configure um canal de conexão do servidor e um listener.
- Forneça autoridade para um usuário apropriado acessar o gerenciador de filas.
- Crie um arquivo CCDT que contenha as informações de conexão para o gerenciador de fila
- Inclua as informações de conexão no messaging REST API usando o comando **setmqweb remote** ..

Em seguida, é possível usar o gerenciador de filas remotas fornecendo o nome exclusivo na URL do recurso no lugar do nome do gerenciador de filas..

Também é possível configurar seus gerenciadores de filas remotas como parte de um grupo de gerenciadores. Para obter informações adicionais, consulte "Configurando um grupo de gerenciadores de filas para usar com a API REST do sistema de mensagens" na página 724.

Procedimento

1. No gerenciador de filas remotas, crie um canal de conexão do servidor para permitir conexões remotas com o gerenciador de fila... É possível criar canais de conexão do servidor usando o comando MQSC do **DEFINE CHANNEL** na linha de comandos
   Por exemplo, para criar um canal de conexão do servidor `QM1.SVRCONN` para o gerenciador de fila QM1, insira o seguinte comando:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Para obter mais informações sobre o **DEFINE CHANNEL** e as opções disponíveis, consulte DEFINIR CANAL.
2. Assegure que um usuário apropriado esteja autorizado a acessar o gerenciador de filas. Esse usuário também deve ser autorizado para acessar quaisquer filas ou tópicos que você usar para o sistema de mensagens O usuário precisa de autoridade `connect`, `inquire`, `alternate usere set context` sobre o gerenciador de fila No UNIX, Linux, and Windows , use o comando de controle **setmqaut** na linha de comandos No z/OS, defina perfis RACF para fornecer ao usuário autorizado acesso ao gerenciador de filas.
   Por exemplo, no UNIX, Linux, and Windows, para autorizar um usuário, `exampleUser`, para acessar o gerenciador de filas QM1insira o comando a seguir:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Para obter mais informações sobre qual usuário precisa ser autorizado. Consulte "Determinando o principal de segurança usado pelo messaging REST API" na página 727
3. ALW
   Se nenhum listener existir no gerenciador de filas remotas, crie um listener para aceitar conexões de rede recebidas usando o comando MQSC do **DEFINE LISTENER** na linha de comandos.
   Por exemplo, para criar um listener `REMOTE.LISTENER` na porta 1414 para o gerenciador de filas remotas QM1, insira o seguinte comando:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. Assegure-se de que o listener esteja em execução usando o comando do MQSC **START LISTENER** na linha de comandos:

ALW Por exemplo, em AIX, Linux, and Windows para iniciar o listener REMOTE.LISTENER para o gerenciador de filas QM1, insira o comando a seguir:

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS Por exemplo, no z/OS, para iniciar o listener, insira o comando a seguir:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

O espaço de endereço do inicializador de canais deve ser iniciado antes que você possa iniciar um listener no z/OS...

5. No sistema no qual o servidor mqweb que hospeda o messaging REST API está em execução, crie ou atualize um arquivo JSON CCDT que contenha as informações de conexão do gerenciador de filas

   O arquivo CCDT deve incluir as informações `name`, `clientConnection` e `type`. Opcionalmente, é possível incluir informações adicionais como `transmissionSecurity`. Para obter mais informações sobre todas as definições de atributo de canal CCDT, consulte a Lista completa de definições de atributo de canal CCDT.

   O exemplo a seguir mostra um arquivo JSON CCDT para uma conexão de gerenciador de filas remotas. Ele configura o nome do canal para o mesmo nome que o canal de conexão do servidor de exemplo criado na etapa “1” na página 722 A porta de conexão é configurada para o mesmo valor que a porta usada pelo listener. O host de conexão é configurado para o nome do host do sistema no qual o gerenciador de fila remoto, QM1está em execução.

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "example.com",
               "port": 1414
             }],
           "queueManager": "QM1"
         },
         "type": "clientConnection"
       }]
   }
   ```

6. Na instalação que está executando o servidor mqweb que hospeda o messaging REST API, use o comando **setmqweb remote** para incluir as informações do gerenciador de filas remotas na configuração do servidor mqweb.

   Como mínimo, deve-se especificar os parâmetros a seguir:

   - **-qmgrName**, em que você especifica o nome do gerenciador de filas
   - **-ccdtURL**, em que você especifica a URL do CCDT para o gerenciador de fila
   - **-uniqueName**, onde você especifica um nome exclusivo para o gerenciador de fila. O nome exclusivo é usado para diferenciar os gerenciadores de fila remotos que podem ter o mesmo nome e, portanto, não deve existir para identificar outro gerenciador de fila

   É possível especificar várias outras opções, como o nome do usuário e a senha a serem usados para a conexão do gerenciador de filas remotas ou detalhes do armazenamento confiável e do keystore. Para obter uma lista completa de parâmetros que podem ser especificados no comando **setmqweb remote** , consulte setmqweb remote.

   Por exemplo, para incluir o gerenciador de filas remotas QM1, com o arquivo CCDT de exemplo, insira o comando a seguir:

   ```
   setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
   ```

Resultados

O gerenciador de filas remotas pode ser usado com o messaging REST API usando o nome exclusivo na URL do recurso no lugar do nome do gerenciador de fila

Exemplo

O exemplo a seguir configura a conexão do gerenciador de filas remotas para um gerenciador de fila QM1.. O IBM MQ Console autorizado para administrar o gerenciador de filas com base na autorização que é fornecida ao usuário exampleUser. As credenciais desse usuário são fornecidas ao IBM MQ Console quando o **setmqweb remote** é usado para configurar a conexão do gerenciador de filas.

1. No sistema em que o gerenciador de filas remotas QM1 é, um canal de conexão do servidor e um listener são criados.. O listener é iniciado e a autorização é dada para o usuário `exampleUser` se conectar ao gerenciador de filas e acessar uma fila que é usada para o sistema de mensagens:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. No sistema no qual o servidor mqweb está em execução, um arquivo `QM1_ccdt.json` é criado com as seguintes informações de conexão:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. No sistema no qual o servidor mqweb está em execução, as informações de conexão para o gerenciador de filas QM1 são incluídas no servidor mqweb.. As credenciais para `exampleUser` são incluídas nas informações de conexão:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. O messaging REST API pode se conectar ao gerenciador de fila remoto QM1 usando o nome exclusivo para a conexão do gerenciador de filas no lugar do nome do gerenciador de filas na URL do recurso:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:
text/plain;charset=utf-8" --data "Hello World!"
```

V 9.4.0 Configurando um grupo de gerenciadores de filas para usar com a API REST do sistema de mensagens

É possível usar o messaging REST API para se conectar a grupos de gerenciadores de filas para sistema de mensagens. Antes de poder se conectar a um grupo de gerenciadores de filas, deve-se configurar a configuração do gerenciador de filas remotas para o grupo Em seguida, é possível se conectar ao grupo de gerenciadores de fila usando o nome exclusivo definido nas informações de configuração.

Antes de começar

- Assegure-se de que você tenha configurado o servidor mqweb para uso pelo administrative REST API, o administrative REST API para MFT, o messaging REST APIou IBM MQ Console.Para obter mais informações sobre como configurar o servidor mqweb com um registro básico, consulte Configuração básica para o servidor mqweb
- Se o servidor mqweb já estiver configurado, assegure-se de que tenha incluído os usuários apropriados para ativar o sistema de mensagens na etapa 5 de Configuração básica para o servidor mqweb.
  Os usuários do messaging REST API devem ser um membro da função `MQWebUser` As funções `MQWebAdmin` e `MQWebAdminRO` não são aplicáveis à messaging REST API.
  - Se **mqRestMessagingAdoptWebUserContext** for configurado como `true` na configuração do servidor mqweb, os usuários na função `MQWebUser` deverão estar autorizados a acessar filas e tópicos que são usados para o sistema de mensagens É possível autorizar esses usuários por meio do OAM ou do RACF
  - Se **mqRestMessagingAdoptWebUserContext** for configurado como `false` na configuração do servidor mqweb, o ID do usuário que inicia o servidor mqweb deverá ser autorizado a acessar filas e tópicos que são usados para o sistema de mensagens. É possível autorizar esse usuário por meio do OAM ou do RACF
- Assegure-se de que o messaging REST API esteja configurado para conectar a gerenciadores de filas remotas. Para obter mais informações, consulte Configurando o modo de conexão para o messaging REST API

Sobre esta tarefa

Um grupo de gerenciadores de filas permite conectar aplicativos a qualquer gerenciador de filas dentro do grupo. O grupo é definido como um conjunto de conexões em uma tabela de definição de canal cliente (CCDT). Ao usar uma chamada MQCONN ou MQCONNX, você faz referência ao grupo prefixando um asterisco no nome do gerenciador de filas. Com o messaging REST API, você faz referência ao grupo usando o nome exclusivo associado ao grupo de gerenciadores de filas. O nome exclusivo é incluído na URL de recurso no lugar do nome do gerenciador de filas Para obter mais informações sobre os grupos de gerenciadores de filas, consulte “Grupos de gerenciadores de filas na CCDT” na página 935

Também é possível configurar os seus gerenciadores de filas remotas individualmente Para obter informações adicionais, consulte “Configurando um gerenciador de filas remotas para usar com o messaging REST API” na página 721.

Procedimento

1. Em cada um dos gerenciadores de filas remotas no grupo, crie um canal de conexão do servidor para permitir conexões remotas ao gerenciador de filas. É possível usar o comando MQSC do **DEFINE CHANNEL** na linha de comandos para criar canais de conexão do servidor
   Por exemplo, para criar um canal de conexão do servidor `QM1.SVRCONN` para o gerenciador de fila QM1, insira o seguinte comando:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Para obter mais informações sobre o **DEFINE CHANNEL** e as opções disponíveis, consulte DEFINIR CANAL.
2. Em cada um dos gerenciadores de filas remotas no grupo, assegure que um usuário apropriado esteja autorizado a acessar o gerenciador de filas. Esse usuário também deve ser autorizado para acessar quaisquer filas ou tópicos que você usar para o sistema de mensagens O usuário precisa de autoridade `connect`, `inquire`, `alternate usere set context` sobre o gerenciador de fila No UNIX, Linux, and Windows , use o comando de controle **setmqaut** na linha de comandos No z/OS, defina perfis RACF para fornecer ao usuário autorizado acesso ao gerenciador de filas.

Por exemplo, em UNIX, Linux, and Windows, insira o seguinte comando para autorizar um usuário, `exampleUser`, a acessar o gerenciador de filas QM1:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

Para obter mais informações sobre qual usuário precisa ser autorizado. Consulte “Determinando o principal de segurança usado pelo messaging REST API” na página 727

3. ALW
   Se nenhum listener existir em cada um dos gerenciadores de filas remotas no grupo, crie listeners para aceitar conexões de rede recebidas. É possível usar o comando MQSC do **DEFINE LISTENER** na linha de comandos para criar listeners
   Por exemplo, para criar um listener `REMOTE.LISTENER` na porta 1414 para o gerenciador de filas remotas QM1, insira o seguinte comando:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```

4. Em cada um dos gerenciadores de filas remotas no grupo, assegure-se de que o listener esteja em execução usando o comando MQSC **START LISTENER** na linha de comandos.

   ALW Por exemplo, em AIX, Linux, and Windows para iniciar o listener `REMOTE.LISTENER` para o gerenciador de filas QM1, insira o comando a seguir:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS Por exemplo, no z/OS, para iniciar o listener, insira o comando a seguir:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

   O espaço de endereço do inicializador de canais deve ser iniciado antes que você possa iniciar um listener no z/OS...
5. No sistema no qual o servidor mqweb que hospeda o messaging REST API está em execução, crie um arquivo JSON CCDT Esse arquivo JSON contém informações de conexão para cada gerenciador de filas no grupo

   O arquivo CCDT deve incluir as informações `name`, `clientConnectione` `type` para cada conexão do gerenciador de filas. Opcionalmente, é possível incluir informações adicionais como `transmissionSecurity`. Para obter mais informações sobre todas as definições de atributo de canal CCDT, consulte a Lista completa de definições de atributo de canal CCDT.

   O exemplo a seguir mostra um arquivo JSON CCDT básico para duas conexões do gerenciador de filas A primeira conexão é para o gerenciador de fila QM1. Ele possui um canal de conexão do servidor de `QM1.SVRCONN`, um listener na porta 1414e é executado no host `QM1.example.com` A segunda conexão é para o gerenciador de filas QM2 Ele possui um canal de conexão do servidor de `QM2.SVRCONN`, um listener na porta 1415e é executado no host `QM2.example.com` No entanto, como as conexões fazem parte do grupo de gerenciadores de fila QMGRP, o campo **queueManager** para ambas as conexões é configurado para o nome do grupo de gerenciadores de filas

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "QM1.example.com",
               "port": 1414
             }],
           "queueManager": "QMGRP"
         },
         "type": "clientConnection"
     }],
   ```

```
   "channel": [{
      "name": "QM2.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM2.example.com",
            "port": 1415
          }],
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
    }]
}
```

6. Na instalação que está executando o servidor mqweb que hospeda o messaging REST API, use o comando **setmqweb remote** para incluir o grupo de gerenciadores de filas na configuração do servidor mqweb.

   Como mínimo, deve-se especificar os parâmetros a seguir:

   - **-qmgrName**, em que você especifica o nome do grupo para o grupo do gerenciador de filas
   - **-ccdtURL**, em que você especifica a URL do CCDT para os gerenciadores de filas
   - **-uniqueName**, em que você especifica um nome exclusivo para identificar o grupo de gerenciadores de filas
   - **-group**, para configurar as informações do gerenciador de filas como sendo para um grupo

   É possível especificar várias outras opções, como o nome de usuário e a senha para usar para a conexão ou detalhes do armazenamento confiável e do keystore. Para obter uma lista completa de parâmetros que podem ser especificados no comando **setmqweb remote** , consulte setmqweb remote.

   Por exemplo, para incluir o grupo de gerenciadores de filas QMGRP, com o arquivo CCDT de exemplo, insira o comando a seguir:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

Resultados

O grupo de gerenciadores de filas remotas pode ser usado com o messaging REST API usando o nome exclusivo na URL de recurso Um gerenciador de filas do grupo é selecionado para concluir a solicitação e informações sobre qual gerenciador de filas concluiu a solicitação são retornadas no cabeçalho de resposta `ibm-mq-resolved-qmgr`.

V 9.4.0 Determinando o principal de segurança usado pelo messaging REST API

Quando você usa o messaging REST API, um usuário apropriado deve estar autorizado a acessar os gerenciadores de filas, filas e tópicos aos quais você deseja se conectar para o sistema de mensagens O usuário que precisa ser autorizado depende como seu servidor mqweb está configurado e se você está usando gerenciadores de filas remotas com o messaging REST API.

Por padrão, o proprietário de segurança usado para autorizar o acesso ao gerenciador de filas é o usuário que inicia o servidor mqweb que executa o messaging REST API. O principal de segurança que é usado para autorizar o acesso às filas e tópicos é o usuário que efetuou login no messaging REST API. No entanto, a sua conexão do servidor mqweb ou do gerenciador de filas remotas pode ser configurada de forma que um proprietário de segurança diferente seja usado

Determinando o principal de segurança usado para se conectar ao gerenciador de filas

Para conexões do gerenciador de fila local, o principal de segurança usado para se conectar ao gerenciador de filas é o usuário que inicia o servidor mqweb que executa o messaging REST API. Para conexões do gerenciador de filas remotas, os principais de segurança a seguir são usados pelo messaging

REST API para autorizar o acesso ao gerenciador de filas, em ordem de prioridade Ou seja, se os usuários forem especificados de várias maneiras dentro da configuração do gerenciador de filas remotas, o primeiro na lista será usado para autorização

1. O proprietário de segurança é um contexto de usuário adotado de uma saída de segurança.
2. O principal de segurança é um contexto do usuário adotado em uma regra CHLAUTH no canal de conexão do servidor que é usado para conectar ao gerenciador de filas remotas.
3. O principal de segurança é o ID do usuário incluído na configuração do gerenciador de filas remotas para o messaging REST API. Esse ID do usuário é opcionalmente incluído nas informações de conexão do gerenciador de filas ao incluir o gerenciador de filas com o comando **setmqweb remote** .
4. O principal de segurança é o usuário que inicia o servidor mqweb que executa o messaging REST API

Para obter mais informações sobre como configurar gerenciadores de fila remotos para usar com o messaging REST API, consulte “Configurando um gerenciador de filas remotas para usar com o messaging REST API” na página 721.

Determinando o principal de segurança usado para se conectar a filas e tópicos

É possível configurar uma propriedade na configuração do servidor mqweb para determinar qual proprietário da segurança é usado para autorizar conexões com filas e tópicos quando você usa o messaging REST API Essa propriedade é a propriedade **mqRestMessagingAdoptWebUserContext** É possível visualizar para o que essa propriedade está configurada usando o comando **dspmqweb properties**

- Se **mqRestMessagingAdoptWebUserContext** for configurado como true, o messaging REST API usará o ID do usuário do usuário que efetuou login no messaging REST API para autorização. Portanto, o ID do usuário ou os IDs do usuário que existem na configuração do servidor mqweb para uso com o messaging REST API são os principais de segurança que devem ser autorizados a acessar as filas e os tópicos.
- Se **mqRestMessagingAdoptWebUserContext** for configurado como false, o messaging REST API usará o ID do usuário do usuário que iniciou o servidor mqweb que hospeda o messaging REST API para autorização. Portanto, um ID do usuário que é igual ao ID do usuário que inicia o servidor mqweb que hospeda o messaging REST API deve ser autorizado a acessar as filas e os tópicos.

  Se suas filas e tópicos estiverem em um gerenciador de fila remoto, o principal de segurança que é usado para autorização poderá ser determinado pelas configurações na configuração do gerenciador de filas Os principais de segurança a seguir podem ser usados, em ordem de prioridade:

  1. O proprietário de segurança é um contexto de usuário adotado de uma saída de segurança.
  2. O principal de segurança é um contexto do usuário adotado em uma regra CHLAUTH no canal de conexão do servidor que é usado para conectar ao gerenciador de filas remotas. Por exemplo, é possível configurar uma regra CHLAUTH no canal de conexão do servidor para usar o parâmetro MCAUSER.. Em seguida, todas as conexões são mapeadas para um ID do usuário que está autorizado a usar o gerenciador de fila..
  3. O principal de segurança é um contexto do usuário adotado do AUTHINFO do gerenciador de filas. Se o objeto AUTHINFO que é referido pelo atributo CONNAUTH do gerenciador de filas for configurado para usar **ADOPTCTX(yes)**, o proprietário de segurança que é usado para autorizar conexões com o gerenciador de filas também será usado para autorizar as filas e tópicos. Por exemplo, esse proprietário de segurança pode ser o ID do usuário incluído nas informações de conexão do gerenciador de filas remotas como parte do comando **setmqweb remote** .

Informações relacionadas

CHLAUTH

CONNAUTH

Propriedades dspmqweb

Desenvolvendo aplicativos MQI com o IBM MQ

IBM MQ fornece suporte para C, Visual Basic, COBOL, Assembler, RPG, pTAL e PL/I. Essas linguagens processuais utilizam a interface de fila de mensagens (MQI) para acessar serviços de enfileiramento de mensagens.

Para obter informações detalhadas sobre como escrever seus aplicativos em sua linguagem de escolha, consulte os subtópicos.

Para obter uma visão geral da interface de chamada para linguagens processuais, consulte Descrições de chamada. Este tópico contém uma lista das chamadas MQI e cada chamada mostra como codificar as chamadas em cada um dessas linguagens.

O IBM MQ fornece arquivos de definição de dados que ajudam a gravar seus aplicativos. Para obter uma descrição integral, consulte “Arquivos de definição de dados doIBM MQ” na página 729.

Para ajudá-lo a escolher em qual linguagem processual codificar seus programas, considere o comprimento máximo das mensagens que seus programas processarão. Se seus programas forem processar somente mensagens de um comprimento máximo conhecido, será possível codificá-los em qualquer uma das linguagens suportadas. Se você não souber o comprimento máximo das mensagens que os programas terão de processar, a linguagem escolhida dependerá de se você estiver escrevendo um aplicativos CICS, IMS ou em lote:

IMS e em lote
: Codifique os programas na linguagem C, PL/I ou assembler para usar os recursos que essas linguagens oferecem para obter e liberar quantias de memória arbitrárias. Como alternativa, você poderia codificar seus programas em COBOL, mas usar sub-rotinas na linguagem assembler, PL/I ou C para obter e liberar armazenamento.

CICS
: Codifique a programas em qualquer linguagem suportada pelo CICS. A interface EXEC CICS fornece as chamadas para gerenciar memória, se necessário.

Conceitos relacionados

“Aplicativos orientados a objetos” na página 16
IBM MQ fornece suporte para JMS, Java, C++ e .NET. Esses idiomas e estruturas usam o IBM MQ Object Model, cujas classes fornecem a mesma funcionalidade que chamadas e estruturas do IBM MQ.

Visão geral técnica

“Conceitos de desenvolvimento de aplicativos” na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

Referências relacionadas

Desenvolvendo a Referência do Aplicativo

Arquivos de definição de dados doIBM MQ

O IBM MQ fornece arquivos de definição de dados que ajudam a gravar seus aplicativos.

Os arquivos de definição de dados também são conhecidos como:

Idioma	Definições de dados
C	Arquivos de inclusão ou arquivos de cabeçalho
Visual Basic	Arquivos de módulo (apenas versões de 32 bits)
COBOL	Arquivos de cópia
Assembler	Macros
PL/I	Arquivos de inclusão

Os arquivos de definição de dados que ajudam a gravar as saídas de canal são descritos em IBM MQ Arquivos COPY, cabeçalho, inclusão e módulo.

Os arquivos de definição de dados que ajudam a gravar as saídas de serviços instaláveis estão descritos em “Saídas de usuário, saídas de API e serviços instaláveis do IBM MQ” na página 948.

Para os arquivos de definição de dados suportados em C++, consulte Usando C++.

IBM i

Para os arquivos de definição de dados suportados em RPG, consulte a IBM i Referência de Programação de Aplicativo (ILE/RPG).

Os nomes dos arquivos de definição de dados possuem o prefixo CMQ e um sufixo que é determinado pela linguagem de programação:

Sufixo	Idioma
a	Linguagem assembler
b	Visual Basic
c	C
l	COBOL (sem valores inicializados)
p	PL/I
v	COBOL (com valores padrão configurados)

Biblioteca de instalação

z/OS O nome **thlqual** é o qualificador de alto nível da biblioteca de instalação no z/OS.

Este tópico apresenta os arquivos de definição de dados do IBM MQ, sob estes títulos:

- “Arquivos de inclusão da Linguagem C” na página 730
- “Arquivos de módulo Visual Basic” na página 731
- “Arquivos de cópia COBOL” na página 731
- z/OS “Macros de linguagem do montador System/390” na página 732
- z/OS “Arquivos de inclusão PL/I” na página 732

Arquivos de inclusão da Linguagem C

Os arquivos de inclusão do IBM MQ C são listados em Arquivos de cabeçalho C. Eles são instalados nos diretórios ou bibliotecas a seguir:

Plataforma	Diretório de instalação ou biblioteca
IBM i IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS z/OS	**thlqual**.SCSQC370

em que *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Nota: Para o AIX and Linux, os arquivos de inclusão são simbolicamente vinculados em `/usr/include`.

Para obter mais informações sobre a estrutura de diretórios, consulte Planejando o suporte ao sistema de arquivos.

Arquivos de módulo Visual Basic

O IBM MQ for Windows fornece quatro arquivos de módulo Visual Basic.

Ele são listados em Arquivos de módulo do Visual Basic e instalados em

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

Arquivos de cópia COBOL

Para COBOL, o IBM MQ fornece arquivos de cópia separados que contém as constantes nomeadas e dois arquivos de cópia para cada uma das estruturas.

Existem dois arquivos de cópia para cada estrutura porque cada um é fornecido com e sem os valores iniciais:

- Em WORKING-STORAGE SECTION de um programa COBOL, use os arquivos que inicializam os campos de estrutura para os valores padrão. Essas estruturas são definidas nos arquivos de cópia que possuem nomes com sufixo de letra V (valores).
- Em LINKAGE SECTION de um programa COBOL, use as estruturas sem os valores iniciais. Essas estruturas são definidas nos arquivos de cópia que possuem nomes com sufixo de letra L (ligação).

IBM i Os arquivos de cópia que contêm os dados e as definições de interface para o IBM i são fornecidos para programas ILE COBOL usando chamadas de protótipo para o MQI. Existem arquivos no QMQM/QCBLLESRC com nomes de membro que possuem um sufixo L (para estruturas sem os valores iniciais) ou um sufixo V (para estruturas com os valores iniciais).

Os arquivos de cópia COBOL do IBM MQ são listados em Arquivos COBOL COPY. Eles são instalados nos diretórios a seguir:

Plataforma	Diretório de instalação ou biblioteca
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (para Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (para IBM VisualAge COBOL)
z/OS z/OS	**thlqual**.SCSQCOBC

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Inclua em seu programa apenas os arquivos que precisar. Faça isso com uma ou mais instruções COPY após uma declaração de nível 01. Isso significa que é possível incluir diversas versões das estruturas em um programa, se necessário. Observe que CMQV é um grande arquivo.

Segue um exemplo de código COBOL para incluir o arquivo de cópia CMQMDV:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

Cada declaração de estrutura inicia com um item de nível 01; é possível declarar diversas instâncias da estruturas codificando a declaração de nível 01 seguida por uma instrução COPY para copiar no restante da declaração de estrutura. Para consultar a instância apropriada, use a palavra-chave IN.

Segue um exemplo de código COBOL para incluir duas instâncias de CMQMDV:

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

Alinhe as estruturas em limites de 4 bytes. Se você usar a instrução COPY para incluir uma estrutura que segue um item que não seja o item de nível 01, certifique-se de que a estrutura seja um múltiplo de 4 bytes do início do item de nível 01. Se isso não for feito, você pode reduzir o desempenho do seu aplicativo.

As estruturas são descritas em Tipos de dados usados no MQI. As descrições dos campos nas estruturas mostram os nomes de campos sem um prefixo. Em programas COBOL, prefixe os nomes de campo com o nome da estrutura seguido por um hífen, conforme mostrado nas declarações COBOL. Os campos nos arquivos de cópia de estrutura recebem o prefixo desta maneira.

Os nomes de campo nas declarações nos arquivos de cópia da estrutura estão em maiúsculas. É possível usar maiúsculas ou minúsculas mistas no lugar. Por exemplo, o campo *StrucId* da estrutura MQGMO é mostrado como MQGMO-STRUCID na declaração COBOL e no arquivo de cópia.

As estruturas de sufixo V são declaradas com valores iniciais para todos os campos, portanto, você precisa configurar apenas os campos nos quais o valor necessário seja diferente do valor inicial.

Macros de linguagem do montador System/390

z/OS

O IBM MQ for z/OS fornece duas macros de linguagem de montador que contêm as constantes nomeadas e uma macro para gerar cada estrutura.

Elas são listadas em z/OSArquivos COPY do montador e instalados em **thlqual**.SCSQMACS.

Essas macros são chamadas usando o código como este:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

Arquivos de inclusão PL/I

z/OS

O IBM MQ for z/OS fornece os arquivos de inclusão que contêm todas as definições que você precisa ao gravar aplicativos IBM MQ em PL/I.

Os arquivos são listados em Arquivos de inclusão PL/I e instalados no diretório **thlqual**.SCSQPLIC:

Inclua esses campos em seu programa se for vincular o stub do IBM MQ a seu programa (consulte “Preparing your program to run” na página 1035). Inclua apenas CMQP se você pretender vincular as chamadas do IBM MQ dinamicamente (consulte “Dynamically calling the IBM MQ stub” na página 1041). A vinculação dinâmica pode ser executada para lote e IMS programas apenas.

Escrevendo um aplicativo processual para enfileiramento

Use essas informações para aprender como escrever aplicativos de enfileiramento, conectar e desconectar de um gerenciador de filas, publicar/assinar e abrir e fechar objetos.

Use os links a seguir para descobrir mais sobre como escrever aplicativos:

- “Visão geral da Message Queue Interface” na página 733
- “Conectando-se e desconectando-se de um gerenciador de filas” na página 747
- “Abrindo e fechando objetos” na página 754
- “Colocando mensagens em uma fila” na página 764
- “Obtendo mensagens de uma fila” na página 780
- “Escrevendo aplicativos de publicar/assinar” na página 821
- “Consultando e configurando atributos de objeto” na página 862
- “Confirmando e fazendo backup de unidades de trabalho” na página 865
- “Iniciando aplicativos IBM MQ usando acionadores” na página 877
- “Trabalhando com MQI e clusters” na página 896
- z/OS “Using and writing applications on IBM MQ for z/OS” na página 901
- z/OS “IMS and IMS bridge applications on IBM MQ for z/OS” na página 71

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

“Desenvolvendo aplicativos para o IBM MQ” na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

“Considerações de design para aplicativos IBM MQ” na página 50
Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

“Escrevendo aplicativos clientes processuais” na página 924
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

“Construindo um aplicativo processual” na página 1012
É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

“Manipulando erros de programa processual” na página 1050
Estas informações explicarão os erros associados às chamadas MQI de aplicativos, ao realizar uma chamada ou quando sua mensagem for entregue para seu destino final.

Tarefas relacionadas

“Usando os programas processuais de amostra do IBM MQ” na página 1070
Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

Visão geral da Message Queue Interface

Aprenda sobre os componentes de Message Queue Interface (MQI).

A Message Queue Interface consiste no seguinte:

- *Chamadas* por meio das quais os programas podem acessar o gerenciador de filas e seus recursos
- *Estruturas* que os programas usam para passar os dados e obter os dados do gerenciador de filas
- *Tipos de dados elementares* para passar os dados e obter os dados do gerenciador de filas

z/OS O IBM MQ for z/OS também fornece:

- Duas chamadas adicionais através das quais programas z/OS em lote podem confirmar e restaurar mudanças.
- *Arquivos de definição de dados* (às vezes conhecidos como arquivos de cópia, macros, arquivos de inclusão e arquivos de cabeçalho) que definem os valores das constantes fornecidas com o IBM MQ for z/OS.
- *Programas stub* para editar o link para seus aplicativos.
- Um conjunto de programas de amostra que demonstram como usar a MQI na plataforma z/OS. Para obter informações adicionais sobre essas amostras, consulte "Using the sample programs for z/OS" na página 1176.

IBM i O IBM MQ for IBM i também fornece:

- *Arquivos de definição de dados* (às vezes conhecidos como arquivos de cópia, macros, arquivos de inclusão e arquivos de cabeçalho) que definem os valores das constantes fornecidas com o IBM MQ for IBM i.
- Três programas stub para editar o link para seus aplicativos ILE C, ILE COBOL e ILE RPG.
- Um conjunto de programas de amostra que demonstram como usar a MQI na plataforma IBM i.

sistemas AIX, Linux, and Windows também fornecem:

- Chamadas através das quais os programas de sistemas IBM MQ for AIX, Linux, and Windows podem confirmar e desfazer mudanças.
- *Arquivos de inclusão* que definem os valores das constantes fornecidas nessas plataformas.
- *Arquivos de biblioteca* para vincular seus aplicativos.
- Um conjunto de programas de amostra que demonstra como usar a MQI nessas plataformas. Para obter informações adicionais sobre essas amostras, consulte "Usando os programas de amostra em multiplataformas" na página 1071.
- Código de amostra e código de executável para ligações aos gerenciadores de transações externos.

Use os links a seguir para descobrir mais sobre a MQI:

- "Chamadas MQI" na página 735
- "Chamadas de ponto de sincronização" na página 736
- "Conversão de dados, tipos de dados, definições de dados e estruturas" na página 736
- "Programas stub e arquivos de biblioteca do IBM MQ" na página 737
- "Parâmetros comuns a todas as chamadas" na página 742
- "Especificando buffers" na página 743
- z/OS "z/OS batch considerations" na página 743
- "Manipulação de sinais do AIX and Linux" na página 744

Conceitos relacionados

"Conectando-se e desconectando-se de um gerenciador de filas" na página 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

"Abrindo e fechando objetos" na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

"Colocando mensagens em uma fila" na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" na página 780
Use estas informações para aprender como obter mensagens de uma fila.

"Consultando e configurando atributos de objeto" na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Chamadas MQI

Use estas informações para aprender sobre chamadas no Message Queue Interface (MQI).

As chamadas no MQI podem ser agrupadas da seguinte maneira:

MQCONN, MQCONNX e MQDISC
Use essas chamadas para conectar um programa a (com ou sem as opções), e desconectar um programa de, um gerenciador de filas.

z/OS Se você gravar programas do CICS para o z/OS, não será necessário usar essas chamadas. No entanto, é recomendado que você as utilize se desejar portar seu aplicativo para outras plataformas.

MQOPEN e MQCLOSE
Use essas chamadas para abrir e fechar um objeto, como uma fila.

MQPUT e MQPUT1
Use essas chamadas para colocar uma mensagem em uma fila.

MQGET
Use esta chamada para procurar mensagens em uma fila ou remover mensagens de uma fila.

MQSUB, MQSUBRQ
Use essas chamadas para registrar uma assinatura em um tópico e para solicitar publicações que correspondam à assinatura.

MQINQ
Use esta chamada para pesquisar sobre os atributos de um objeto.

MQSET
Use esta chamada para configurar alguns dos atributos de uma fila. Não é possível configurar os atributos de outros tipos de objeto.

MQBEGIN, MQCMIT e MQBACK
Use essas chamadas quando o IBM MQ for o coordenador de uma unidade de trabalho. MQBEGIN inicia a unidade de trabalho. MQCMIT e MQBACK encerram a unidade de trabalho, seja confirmando ou retrocedendo as atualizações feitas durante a unidade de trabalho. IBM i O controlador de compromisso do IBM i é usado para coordenar unidades de trabalho globais no IBM MQ for IBM i. Os comandos native start commitment control, commit e rollback são usados.

MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH
Use essas chamadas para criar um identificador de mensagens, para converter um identificador de mensagens em um buffer ou um buffer em um identificador de mensagens e para excluir um identificador de mensagens.

MQSETMP, MQINQMP, MQDLTMP
Use essas chamadas para configurar uma propriedade da mensagem em um identificador de mensagens, pesquisar sobre uma propriedade da mensagem e excluir uma propriedade de um identificador de mensagens.

MQCB, MQCB_FUNCTION, MQCTL
Use essas chamadas para registrar e controlar uma função de retorno de chamada.

MQSTAT
Use esta chamada para recuperar as informações de status sobre as operações put assíncronas anteriores.

Consulte Descrições de chamadas para obter uma descrição das chamadas MQI.

Chamadas de ponto de sincronização

Use estas informações para descobrir chamadas de ponto de sincronização em diferentes plataformas.

Chamadas de ponto de sincronização estão disponíveis da seguinte forma:

Chamadas do IBM MQ for z/OS

O IBM MQ for z/OS fornece as chamadas MQCMIT e MQBACK.

Use essas chamadas em programas em lote z/OS para informar ao gerenciador de filas que todas as operações MQGET e MQPUT desde o último ponto de sincronização devem se tornar permanentes (confirmadas) ou devem ser restauradas. Para confirmar e restaurar mudanças em outros ambientes:

CICS
Use comandos como EXEC CICS SYNCPOINT e EXEC CICS SYNCPOINT ROLLBACK.

IMS
Use os recursos de ponto de sincronização do IMS, como as chamadas GU (get unique) para IOPCB, CHKP (checkpoint) e ROLB (rollback).

RRS
Use MQCMIT e MQBACK ou SRRCMIT e SRRBACK, conforme apropriado. (Consulte “Transaction management and recoverable resource manager services” na página 870.)

Nota: SRRCMIT e SRRBACK são comandos nativos RRS, não são chamadas MQI.

Chamadas do IBM i

O IBM MQ for IBM i fornece os comandos MQCMIT e MQBACK. Também é possível usar os comandos COMMIT e ROLLBACK do IBM i ou quaisquer outros comandos ou chamadas que iniciem os recursos de controle de compromisso do IBM i (por exemplo, EXEC CICS SYNCPOINT).

chamadas IBM MQ em plataformas AIX, Linux, and Windows

IBM MQ for AIX, Linux, and Windows fornecem as chamadas MQCMIT e MQBACK.

Use chamadas de ponto de sincronização em programas para informar ao gerenciador de filas que todas as operações MQGET e MQPUT desde o último ponto de sincronização devem se tornar permanentes (confirmadas) ou devem ser restauradas. Para confirmar e recuperar mudanças no ambiente do CICS, use comandos como EXEC CICS SYNCPOINT e EXEC CICS SYNCPOINT ROLLBACK.

Conversão de dados, tipos de dados, definições de dados e estruturas

Use estas informações para aprender sobre conversões de dados, tipos de dados elementares, definições de dados do IBM MQ e estruturas ao usar a Message Queue Interface.

Conversão de Dados

A chamada MQXCNVC (converter caracteres) converte dados de caracteres de mensagens de um conjunto de caracteres para outro. Essa chamada é usada apenas a partir de uma saída de conversão de dados, exceto no IBM MQ for z/OS

Consulte MQXCNVC - Converter caracteres para a sintaxe usada com a chamada MQXCNVC e “Escrevendo saídas de conversão de dados” na página 995 para obter orientação sobre como gravar e chamar saídas de conversão de dados.

Tipos de dados elementares

Para as linguagens de programação suportadas, o MQI fornece tipos de dados elementares ou campos não estruturados.

Esses tipos de dados são totalmente descritos em Tipos de dados elementares.

Definições de dados do IBM MQ

z/OS O IBM MQ for z/OS fornece definições de dados na forma de arquivos de cópia COBOL, macros de linguagem assembly, um único arquivo de inclusão PL/I, um único arquivo de inclusão de linguagem C e arquivos de inclusão de linguagem C++.

IBM i O IBM MQ for IBM i fornece definições de dados no formato de arquivos de cópia COBOL, arquivos de cópia RPG, arquivos de inclusão de linguagem C e arquivos de inclusão de linguagem C++.

Os arquivos de definição de dados fornecidos com o IBM MQ contêm:

- Definições de todas as constantes e códigos de retorno do IBM MQ
- Definições das estruturas e tipos de dados do IBM MQ
- Definições de constantes para inicializar as estruturas
- Protótipos de funções para cada uma das chamadas (para PL/I e a linguagem C apenas)

Para obter uma descrição completa dos arquivos de definição de dados do IBM MQ, consulte “Arquivos de definição de dados doIBM MQ” na página 729.

Estruturas

Estruturas, usadas com as chamadas MQI listadas no “Chamadas MQI” na página 735, são fornecidas nos campos de definição de dados para cada uma das linguagens de programação suportadas.

Consulte Tipos de dados de estrutura para obter um resumo das estruturas

IBM i z/OS IBM MQ for z/OS e IBM MQ for IBM i fornecem arquivos que contêm constantes para você usar ao preencher alguns dos campos dessas estruturas. Para obter mais informações sobre isso, consulte Definições de dados do IBM MQ.

Programas stub e arquivos de biblioteca do IBM MQ

Os programas de stub e arquivos de biblioteca fornecidos estão listados aqui, para cada plataforma.

Para obter mais informações sobre como usar os programas stub e arquivos de biblioteca ao construir um aplicativo executável, consulte “Construindo um aplicativo processual” na página 1012. Para obter informações sobre como vincular para arquivos de biblioteca C++, consulte Usando C++ *IBM MQ Usando C++*.

AIX *Arquivos de biblioteca do IBM MQ for AIX*

No IBM MQ for AIX, deve-se vincular o programa aos arquivos de biblioteca MQI fornecidos para o ambiente no qual você está executando o aplicativo, além daqueles fornecidos pelo sistema operacional.

Em um aplicativo não encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 106. Arquivos de biblioteca para aplicativos AIX não encadeados	
Arquivo de biblioteca	**Meio ambiente**
libmqm.a	Servidor para C
libmqic.a e libmqm.a	Cliente para C
libmqmzf.a	Saídas de serviço instalável para C
libmqmxa.a	Interface XA do servidor
libmqmxa64.a	Interface XA alternativa do servidor
libmqcxa.a	Interface XA do cliente
libmqcxa64.a	Interface XA alternativa do cliente
libmqmcbrt.o	IBM MQ biblioteca de tempo de execução para suporte do Micro Focus COBOL
libmqmcb.a	Servidor para COBOL
libmqicb.a	Cliente para COBOL
libimqc23ia.a	Cliente para C++ (XLC 16)
libimqs23ia.a	Servidor para C++ (XLC 16)
V 9.4.0 libimqc23ca.a	Cliente para C++ (XLC 17)
V 9.4.0 libimqs23ca.a	Servidor para C++ (XLC 17)

V 9.4.0 Bibliotecas contendo "`ia`" foram construídas com o compilador XLC 16, enquanto bibliotecas com "`ca`" no nome foram construídas com o compilador XLC 17.

Em um ambiente encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 107. Arquivos de biblioteca para aplicativos AIX encadeados. Uma tabela com duas colunas listando os arquivos de biblioteca e o ambiente de cada arquivo de biblioteca.	
Arquivo de biblioteca	**Meio ambiente**
libmqm_r.a	Servidor para C
libmqic_r.a e libmqm_r.a	Cliente para C
libmqmzf_r.a	Saídas de serviço instalável para C
libmqmxa_r.a	Interface XA do servidor
libmqmxa64_r.a	Interface XA alternativa do servidor
libmqcxa_r.a	Interface XA do cliente
libmqcxa64_r.a	Interface XA alternativa do cliente
libimqc23ia_r.a	Cliente para C++ (XLC 16)
libimqs23ia_r.a	Servidor para C++ (XLC 16)
V 9.4.0 libimqc23ca_r.a	Cliente para C++ (XLC 17)
V 9.4.0 libimqs23ca_r.a	Servidor para C++ (XLC 17)

V 9.4.0 Bibliotecas com nomes que incluem `ia` foram construídas com o compilador XLC 16, enquanto bibliotecas com nomes que incluem `ca` foram construídas com o compilador XLC 17.

Nota: Não é possível se vincular a mais de uma biblioteca. Ou seja, não é possível se vincular a uma biblioteca encadeada e a uma não encadeada ao mesmo tempo.

IBM i *Arquivos de biblioteca do IBM MQ for IBM i*

No IBM MQ for IBM i, vincule o seu programa aos arquivos de biblioteca da MQI fornecidos para o ambiente no qual você está executando o seu aplicativo, além dos fornecidos pelo sistema operacional.

Para aplicativos não encadeados:

Tabela 108. Arquivos de biblioteca para aplicativos IBM i não encadeados

Arquivo de biblioteca	**Meio ambiente**
LIBMQM	Programa de serviços do servidor e do cliente
LIBMQIC	Programa de serviços do cliente
IMQB23I4	Programa de serviços base C++
IMQS23I4	Programa de serviços do servidor C++
LIBMQMZF	Saídas instaláveis para C

Em um aplicativo encadeado:

Tabela 109. Arquivos de biblioteca para aplicativos IBM i encadeados

Arquivo de biblioteca	**Meio ambiente**
LIBMQM_R	Programa de serviços do servidor e do cliente
IMQB23I4_R	Programa de serviços base C++
IMQS23I4_R	Programa de serviços do servidor C++
LIBMQMZF_R	Saídas instaláveis para C
LIBMQIC_R	Programa de serviços do cliente

No IBM MQ for IBM i, é possível gravar seus aplicativos em C++. Para ver como vincular seus aplicativos C++ e para obter detalhes completos de todos os aspectos do uso de C++, consulte Usando C++.

Linux *Arquivos de biblioteca do IBM MQ for Linux*

No IBM MQ para Linux, deve-se vincular o programa aos arquivos de biblioteca MQI fornecidos para o ambiente no qual você está executando o aplicativo, além daqueles fornecidos pelo sistema operacional.

Em um aplicativo não encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 110. Arquivos de biblioteca para aplicativos Linux não encadeados

Arquivo de biblioteca	**Meio ambiente**
libmqm.so	Servidor para C
libmqic.so e libmqm.so	Cliente para C
libmqmzf.so	Saídas de serviço instalável para C
libmqmxa.so	Interface XA do servidor
libmqmxa64.so	Interface XA alternativa do servidor
libmqcxa.so	Interface XA do cliente

Tabela 110. Arquivos de biblioteca para aplicativos Linux não encadeados (continuação)	
Arquivo de biblioteca	**Meio ambiente**
libmqcxa64.so	Interface XA alternativa do cliente
libimqc23gl.so	Client for C++
libimqs23gl.so	Servidor para C++

Em um ambiente encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 111. Arquivos de biblioteca para aplicativos Linux encadeados	
Arquivo de biblioteca	**Meio ambiente**
libmqm_r.so	Servidor para C
libmqic_r.so e libmqm_r.so	Cliente para C
libmqmzf_r.so	Saídas de serviço instalável para C
libmqmxa_r.so	Interface XA do servidor
libmqmxa64_r.so	Interface XA alternativa do servidor
libmqcxa_r.so	Interface XA do cliente
libmqcxa64_r.so	Interface XA alternativa do cliente
libimqc23gl_r.so	Client for C++
libimqs23gl_r.so	Servidor para C++

Nota: Não é possível se vincular a mais de uma biblioteca. Ou seja, não é possível se vincular a uma biblioteca encadeada e a uma não encadeada ao mesmo tempo.

Windows *Arquivos de biblioteca do IBM MQ for Windows*

No IBM MQ for Windows, deve-se vincular o seu programa aos arquivos da biblioteca do MQI fornecidos para o ambiente no qual você está executando seu aplicativo, além dos fornecidos pelo sistema operacional:

Tabela 112. Arquivos de biblioteca para aplicativos do Windows	
Arquivo de biblioteca	**Meio ambiente**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Server for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Client for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	Server XA interface for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	Client XA interface for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Client MTS for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	Server TXSeries CICS support for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	Client TXSeries CICS support for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Installable services exits for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Server for IBM COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Server for Micro Focus COBOL (32 bits)

Tabela 112. Arquivos de biblioteca para aplicativos do Windows (continuação)	
Arquivo de biblioteca	**Meio ambiente**
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Client for IBM COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Client for Micro Focus COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Server for C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Client for C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	Base for C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Client MTS for C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Server for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Client for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	Server XA interface for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	Client XA interface for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	Client MTS for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Server for IBM COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Server for Micro Focus COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	Client for IBM COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Client for Micro Focus COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	Server for C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	Client for C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	Base for C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	Client MTS for C++ (64 bits)

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Use o `amqmdnet.dll` para compilar os programas .NET. Consulte "Compilando programas IBM MQ .NET" na página 622 na seção "Desenvolvendo aplicativos do .NET" na página 565 para obter mais informações.

Estes arquivos são enviados para compatibilidade com liberações anteriores:

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS stub programs*

Before you can run a program written with IBM MQ for z/OS, you must link-edit it to the stub program supplied with IBM MQ for z/OS for the environment in which you are running the application.

The stub program provides the first stage of the processing of your calls into requests that IBM MQ for z/OS can process.

IBM MQ for z/OS supplies the following stub programs:

CSQBSTUB
Stub program for z/OS batch programs

CSQBRRSI
Stub program for z/OS batch programs using RRS by way of the MQI

CSQBRSTB
Stub program for z/OS batch programs using RRS directly

CSQCSTUB
Stub program for CICS programs

CSQQSTUB
Stub program for IMS programs

CSQXSTUB
Stub program for distributed queuing non-CICS exits

CSQASTUB
Stub program for data-conversion exits

Attention: If you use a stub program other than one listed for a specific environment, it might have unpredictable results.

Note: If you use the CSQBRSTB stub program, link-edit with ATRSCSS from SYS1.CSSLIB. (SYS1.CSSLIB is also known as the *Callable Services Library*). For more information about RRS see "Transaction management and recoverable resource manager services" on page 870.

Alternatively, you can dynamically call the stub from within your program. This technique is described in "Dynamically calling the IBM MQ stub" on page 1041.

In IMS, you might also need to use a special language interface module that is supplied by IBM MQ.

Do not run applications that are link-edited with CSQBSTUB and CSQQSTUB in the same IMS MPP region. This can cause problems such as DFS3607I or CSQQ005E messages. The first MQCONN call in an address space determines which interface is used, therefore CSQQSTUB and CSQBSTUB transactions must run in different IMS message regions.

Parâmetros comuns a todas as chamadas

Há dois tipos de parâmetros comuns a todas as chamadas: identificadores e códigos de retorno.

Usando identificadores

Todas as chamadas MQI usam um ou mais *identificadores*. Eles identificam o gerenciador de filas, a fila ou outro objeto, mensagem ou assinatura, conforme apropriado para a chamada.

Para um programa se comunicar com um gerenciador de filas, o programa deve ter um identificador exclusivo pelo qual conhece aquele gerenciador de filas. Esse identificador é chamado de *manipulação de conexões*, às vezes referido como *Hconn*. Para programas CICS, a manipulação de conexões é sempre zero. Para todas as outras plataformas ou estilos de programas, a manipulação de conexões é retornada pela chamada MQCONN ou MQCONNX quando o programa se conecta ao gerenciador de filas. Os programas passam a manipulação de conexões como um parâmetro de entrada quando eles usam as outras chamadas.

Para um programa funcionar com um objeto do IBM MQ, o programa deve ter um identificador exclusivo pelo qual ele conhece esse objeto. Esse identificador é chamado de *manipulação de objetos*, às vezes referido como um *Hobj*. O identificador é retornado pela chamada MQOPEN quando o programa abre o objeto para trabalhar com ele. Programas passam a manipulação de objetos como um parâmetro de entrada quando eles usam chamadas MQPUT, MQGET, MQINQ, MQSET ou MQCLOSE subsequentes.

De forma semelhante, a chamada MQSUB retorna um *identificador de assinatura* ou *Hsub*, que é usado para identificar a assinatura em chamadas MQGET, MQCB ou MQSUBRQ subsequentes e determinadas chamadas que processam propriedades de mensagens usam um *identificador de mensagem* ou *Hmsg*.

Entendendo códigos de retorno

Um código de conclusão e um código de razão são retornados como parâmetros de saída por cada chamada. Eles são conhecidos coletivamente como *códigos de retorno*.

Para mostrar se uma chamada é bem-sucedida, cada chamada retorna um *código de conclusão* quando a chamada é concluída. O código de conclusão geralmente é MQCC_OK, indicando sucesso ou MQCC_FAILED, indicando falha. Algumas chamadas podem retornar um estado intermediário, MQCC_WARNING, indicando sucesso parcial.

Cada chamada também retorna um *código de razão* que mostra a razão da falha, ou sucesso parcial, da chamada. Há vários códigos de razão, cobrindo circunstâncias como uma fila cheia, operações get não permitidas para uma fila e uma fila específica não estando definida para o gerenciador de filas. Os programas podem usar o código de razão para decidir como proceder. Por exemplo, podem solicitar aos usuários que mudem seus dados de entrada, em seguida, façam a chamada novamente ou podem retornar uma mensagem de erro ao usuário.

Quando o código de conclusão for MQCC_OK, o código de razão é sempre MQRC_NONE.

Os códigos de conclusão e de razão para cada chamada são listados com a descrição da chamada. Consulte Descrições de chamada e selecione a chamada apropriada na lista.

Para obter informações mais detalhadas, incluindo ideias para ação corretiva, consulte:

- z/OS mensagens, conclusão e códigos de razão do IBM MQ for z/OS for IBM MQ for z/OS
- Mensagens e códigos de razão para todas as outras plataformas IBM MQ

Especificando buffers

O gerenciador de filas se refere a buffers somente se eles forem necessários. Se não for necessário um buffer em uma chamada ou o buffer for zero em comprimento, é possível usar um ponteiro nulo para um buffer.

Sempre use datalength ao especificar o tamanho do buffer que você requer.

Ao usar um buffer para reter a saída de uma chamada (por exemplo, para reter os dados da mensagem para uma chamada MQGET ou os valores de atributos consultados pela chamada MQINQ), o gerenciador de filas tentará retornar um código de razão se o buffer que você especificar não for válido ou estiver em armazenamento de leitura. No entanto, pode nem sempre ser capaz de retornar um código de razão.

z/OS *z/OS batch considerations*

z/OS batch programs that call the MQI can be in either supervisor or problem state.

However, they must meet the following conditions:

- They must be in task mode, not service request block (SRB) mode.
- They must be in Primary address space control (ASC) mode (not Access Register ASC mode).
- They must not be in cross-memory mode. The primary address space number (ASN) must be equal to the secondary ASN and the home ASN.
- They must not be used as MPF exit programs.
- No z/OS locks can be held.
- There can be no function recovery routines (FRRs) on the FRR stack.
- Any program status word (PSW) key can be in force for the MQCONN or MQCONNX call (provided the key is compatible with using storage that is in the TCB key), but subsequent calls that use the connection handle returned by MQCONN or MQCONNX:
  - Must have the same PSW key that was used on the MQCONN or MQCONNX call
  - Must have parameters accessible (for write, where appropriate) under the same PSW key
  - Must be issued under the same task (TCB), but not in any subtask of the task
- They can be in either 24-bit or 31-bit addressing mode. However, if 24-bit addressing mode is in force, parameter addresses must be interpreted as valid 31-bit addresses.

If any of these conditions is not met, a program check might occur. In some cases the call will fail and a reason code will be returned.

Linux AIX *considerações AIX and Linux*

Considerações que você precisa ter conhecimento ao desenvolver aplicativos AIX and Linux.

Linux AIX *A chamada de sistema fork em sistemas AIX and Linux*

Observe estas considerações ao usar uma chamada do sistema fork em aplicativos IBM MQ.

Se o seu aplicativo desejar usar `fork`, o processo pai desse aplicativo deve chamar `fork` antes de fazer qualquer chamada do IBM MQ, por exemplo, `MQCONN` ou criar um objeto do IBM MQ usando **`ImqQueueManager`**.

Se o seu aplicativo deseja criar um processo filho após fazer quaisquer chamadas do IBM MQ, o código do aplicativo deve usar um `fork()` com `exec()` para assegurar que o filho seja uma nova instância e não uma cópia exata do pai.

Se o seu aplicativo não usar o `exec()`, a chamada de API do IBM MQ feita dentro do processo filho retorna MQRC_ENVIRONMENT_ERROR.

Linux AIX *Manipulação de sinais do AIX and Linux*

Em geral, os sistemas AIX and Linux mudaram de um ambiente não encadeado (processo) para um ambiente multiencadeado. Em muitas instâncias, sinais e manipulação de sinais, embora suportados, não se ajustam bem no ambiente multiencadeado e existem diversas restrições.

Em geral, os sistemas AIX and Linux mudaram de um ambiente não encadeado (processo) para um ambiente multiencadeado. No ambiente não encadeado, algumas funções podiam ser implementadas somente usando sinais, embora a maioria dos aplicativos não precisasse estar ciente de sinais e da manipulação de sinais. No ambiente multiencadeado, as primitivas baseadas em encadeamento suportam algumas das funções que costumavam ser implementadas nos ambientes não encadeados usando sinais.

Em muitas instâncias, sinais e manipulação de sinais, embora suportados, não se ajustam bem no ambiente multiencadeado e existem diversas restrições. Isso pode ser problemático quando você estiver integrando o código do aplicativo com diferentes bibliotecas de middleware (em execução como parte do aplicativo) em um ambiente multiencadeado em que cada uma está tentando manipular sinais. A abordagem tradicional de salvar e restaurar os manipuladores de sinais (definida por processo), que funcionava quando havia apenas um encadeamento de execução dentro de um processo, não funciona em um ambiente multiencadeado. Isso ocorre porque muitos encadeamentos de execução poderiam estar tentando salvar e restaurar um recurso de todo o processo, com resultados imprevisíveis.

Linux AIX *Aplicativos não encadeados*

Cada função MQI configura seu próprio manipulador de sinais para os sinais. Manipuladores de usuários para esses são substituídos para a duração da chamada de função MQI. Outros sinais podem ser capturados da maneira normal por manipuladores escritos pelo usuário.

Cada função MQI configura seu próprio manipulador de sinais para os sinais:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

Manipuladores de usuários para esses são substituídos para a duração da chamada de função MQI. Outros sinais podem ser capturados da maneira normal por manipuladores escritos pelo usuário. Se você não instalar um manipulador, as ações padrão (por exemplo, ignore, core dump ou exit) são deixadas no local.

Após o IBM MQ manipular um sinal síncrono (SIGSEGV, SIGBUS, SIGFPE, SIGILL), ele tenta passar o sinal para qualquer manipulador de sinais registrado antes de fazer a chamada de função MQI.

Linux AIX *Aplicativos encadeados*

Um encadeamento é considerado para ser conectado ao IBM MQ a partir de MQCONN (ou MQCONNX) até MQDISC.

Sinais síncronos

Sinais síncronos surgem em um encadeamento específico.

Sistemas AIX and Linux permitem de forma segura a configuração de um manipulador de sinais para esses sinais para todo o processo. No entanto, o IBM MQ configura seu próprio manipulador para os sinais a seguir, no processo do aplicativo, enquanto qualquer encadeamento estiver conectado ao IBM MQ:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Se você estiver escrevendo aplicativos multiencadeados, há somente um manipulador de sinais para todo o processo para cada sinal. Quando o IBM MQ configura seus próprios manipuladores de sinais síncronos, salva quaisquer manipuladores registrados anteriormente para cada sinal. Após o IBM MQ manipular um dos sinais listados, o IBM MQ tenta chamar o manipulador de sinais que estava em vigor no momento da primeira conexão do IBM MQ dentro do processo. Os manipuladores registrados anteriormente são restaurados quando todos os encadeamentos de aplicativos tiverem sido desconectados do IBM MQ.

Como manipuladores de sinais são salvos e restaurados pelo IBM MQ, os encadeamentos do aplicativo não devem estabelecer manipuladores de sinais para esses sinais enquanto houver qualquer possibilidade de que outro encadeamento do mesmo processo também esteja conectado ao IBM MQ.

Nota: Quando um aplicativo ou uma biblioteca de middleware (em execução como parte de um aplicativo), estabelece um manipulador de sinais enquanto um encadeamento estiver conectado ao IBM MQ, o manipulador de sinais do aplicativo deve chamar o manipulador correspondente do IBM MQ durante o processamento desse sinal.

Ao estabelecer e restaurar manipuladores de sinais, o princípio geral é que o último manipulador de sinais a ser salvo deve ser o primeiro a ser restaurado:

- Quando um aplicativo estabelece um manipulador de sinais após se conectar ao IBM MQ, o manipulador de sinais anterior deve ser restaurado antes que o aplicativo se desconecte do IBM MQ.
- Quando um aplicativo estabelece um manipulador de sinais antes de se conectar ao IBM MQ, o aplicativo deve se desconectar a partir do IBM MQ antes de restaurar seu manipulador de sinais.

Nota: A falha em observar o princípio geral de que o último manipulador de sinais a ser salvo deve ser o primeiro a ser restaurado pode resultar em manipulação de sinais inesperada no aplicativo e, potencialmente, na perda de sinais pelo aplicativo.

Sinais assíncronos

O IBM MQ não usa quaisquer sinais assíncronos em aplicativos encadeados, a menos que sejam aplicativos clientes.

Considerações adicionais para aplicativos clientes não encadeados

O IBM MQ manipula os sinais a seguir durante a E/S em um servidor. Esses sinais são definidos pela pilha de comunicações. O aplicativo não deve estabelecer um manipulador de sinais para esses sinais enquanto um encadeamento estiver conectado a um gerenciador de filas:

SIGPIPE (para TCP/IP)

Linux AIX *Considerações adicionais ao usar a manipulação de sinal do AIX and Linux na MQI*

Ao utilizar o MQI para a manipulação de sinais no AIX and Linux, há considerações adicionais para aplicações de atalho, chamadas de função MQI nos manipuladores de sinal, sinais durante chamadas MQI, saídas de usuário e serviços instaláveis, assim como e manipuladores de saída de VMS.

Aplicativos Fastpath (confiáveis)

Aplicativos Fastpath são executados no mesmo processo que o IBM MQ e, portanto, estão em execução no ambiente multiencadeado.

Neste ambiente, o IBM MQ manipula os sinais síncronos SIGSEGV, SIGBUS, SIGFPE e SIGILL. Todos os outros sinais não devem ser entregues ao aplicativo Fastpath enquanto estiver conectado ao IBM MQ. Em vez disso, eles devem ser bloqueados ou manipulados pelo aplicativo. Se um aplicativo Fastpath interceptar tal evento, o gerenciador de filas deve ser interrompido e reiniciado ou pode ser deixado em um estado indefinido. Para obter uma lista completa das restrições para aplicativos Fastpath sob MQCONNX, consulte "Conectando-se a um gerenciador de filas usando a chamada MQCONNX" na página 749.

Chamadas de função MQI nos manipuladores de sinais

Enquanto você estiver em um manipulador de sinal, não chame uma função MQI.

Se você tentar chamar uma função MQI a partir de um manipulador de sinal enquanto outra função MQI estiver ativo, MQRC_CALL_IN_PROGRESS será retornado. Se você tentar chamar uma função MQI a partir de um manipulador de sinal enquanto nenhuma outra função MQI estiver ativa, provavelmente falhará em algum momento durante a operação devido às restrições do sistema operacional em que somente chamadas seletivas possam ser emitidas a partir de um manipulador ou dentro dele.

Para métodos destruidores C++, que podem ser chamado automaticamente durante a saída do programa, pode ser que não seja possível parar a chamada de funções MQI. Ignore qualquer erro sobre MQRC_CALL_IN_PROGRESS. Se um manipulador de sinal chamar exit(), o IBM MQ restaura as mensagens não confirmadas no ponto de sincronização como de costume e fecha quaisquer filas abertas.

Sinais durante chamadas MQI

Funções de MQI não retornam o código EINTR ou qualquer equivalente para programas aplicativos.

Se um sinal ocorrer durante uma chamada MQI e o manipulador chamar *return*, a chamada continuará a ser executada como se o sinal não tivesse acontecido. Especificamente, MQGET não pode ser interrompido por um sinal para retornar controle imediatamente para o aplicativo. Se desejar sair de um MQGET, configure a fila para GET_DISABLED; como alternativa, use um loop em torno de uma chamada para MQGET com um tempo de expiração finito (MQGMO_WAIT com gmo.WaitInterval configurado) e use seu manipulador de sinal (em um ambiente não encadeado) ou função equivalente em um ambiente encadeado para configurar uma sinalização que interrompe o loop.

AIX No ambiente do AIX, o IBM MQ requer que as chamadas de sistema interrompidas por sinais sejam reiniciadas. Ao estabelecer seu próprio manipulador de sinal com sigaction(2), configure a sinalização SA_RESTART no campo sa_flags da nova estrutura de ação, caso contrário, o IBM MQ pode ser incapaz de concluir qualquer chamada interrompida por um sinal.

Saídas de usuário e serviços instaláveis

As saídas de usuário e os serviços instaláveis que são executados como parte de um processo do IBM MQ em um ambiente multiencadeado têm as mesmas restrições para aplicativos de atalho. Considere que estejam permanentemente conectados ao IBM MQ e, portanto, sem usar sinais ou chamadas do sistema operacional não thread-safe.

Conectando-se e desconectando-se de um gerenciador de filas

Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

A forma com que essa conexão é feita depende da plataforma e do ambiente no qual o programa está operacional:

Multi **IBM MQ for Multiplatforms**

Os programas que são executados nesses ambientes podem usar a chamada MQI MQCONN para conectar-se a, e a chamada MQDISC para desconectar-se de, um gerenciador de filas. Como alternativa, programas podem usar a chamada MQCONNX.

z/OS **Lote do IBM MQ for z/OS**

Os programas que são executados nesse ambiente podem usar a chamada MQI MQCONN para conectar-se a um gerenciador de filas e a chamada MQDISC para desconectar-se dele. Como alternativa, programas podem usar a chamada MQCONNX.

Programas z/OS em lote podem se conectar, consecutivamente ou simultaneamente, para vários gerenciadores de filas no mesmo TCB.

z/OS **IMS**

A região de controle do IMS é conectada a um ou mais gerenciadores de filas quando ele inicia. Essa conexão é controlada pelos comandos do IMS. Para obter informações sobre como controlar o adaptador do IMS no z/OS, consulte Administrando IBM MQ for z/OS. No entanto, os escritores de programas IMS de enfileiramento de mensagens devem usar a chamada MQI MQCONN para especificar o gerenciador de filas ao qual desejam se conectar. Eles podem usar a chamada MQDISC para desconectar do gerenciador de filas.

Após uma chamada do IMS que estabelece um ponto de sincronização e antes de processar uma mensagem para outro usuário, o adaptador do IMS assegura que o aplicativo fecha identificadores e desconecta-se do gerenciador de filas. Consulte “Syncpoints in IMS applications” na página 869.

Os programas IMS podem se conectar, consecutivamente ou simultaneamente, a vários gerenciadores de filas no mesmo TCB.

z/OS **CICS Servidor de Transação para z/OS .**

Os programas CICS não precisam executar nenhum trabalho para se conectar a um gerenciador de filas porque o próprio sistema CICS está conectado. Essa conexão normalmente é estabelecida automaticamente na inicialização, mas é possível também usar a transação CKQC que é fornecida com o IBM MQ for z/OS. Para obter mais informações sobre CKQC, consulte Administrando IBM MQ for z/OS.

As tarefas do CICS só podem se conectar ao gerenciador de filas ao qual a região do CICS está conectada.

Programas do CICS também podem usar as chamadas MQI de conexão e desconexão (MQCONN e MQDISC). Você pode querer fazer isso para que seja possível ter portabilidade desses aplicativos para ambientes não CICS com um mínimo de recodificação. No entanto, essas chamadas *sempre* são concluídas com sucesso em um ambiente do CICS. Isso significa que o código de retorno pode não refletir o estado verdadeiro da conexão com o gerenciador de filas.

TXSeries for Windows e Open Systems

Esses programas não precisam executar nenhum trabalho para se conectar a um gerenciador de filas, porque o próprio sistema CICS está conectado. Portanto, somente uma conexão por vez é suportada. Aplicativos CICS devem emitir uma chamada MQCONN para obter uma manipulação de conexões e uma chamada MQDISC antes de sair.

Use os links a seguir para descobrir mais sobre como se conectar e desconectar de um gerenciador de filas:

- “Conectando-se a um gerenciador de filas usando a chamada MQCONN” na página 748
- “Conectando-se a um gerenciador de filas usando a chamada MQCONNX” na página 749

- “Desconectando programas de um gerenciador de filas usando MQDISC” na página 753

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Abrindo e fechando objetos” na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Conectando-se a um gerenciador de filas usando a chamada MQCONN

Use estas informações para aprender como se conectar a um gerenciador de filas utilizando a chamada MQCONN.

Em geral, é possível conectar tanto a um gerenciador de filas específico, quanto para o gerenciador de filas padrão:

- z/OS Para o IBM MQ for z/OS, no ambiente em lote, o gerenciador de filas padrão é especificado no módulo CSQBDEFV.
- Multi Para IBM MQ for Multiplatforms, o gerenciador de filas padrão é especificado no arquivo mqs.ini.

z/OS Alternativamente, nos ambientes z/OS MVS em lote, TSO e RRS, é possível se conectar a qualquer gerenciador de filas em um grupo de filas compartilhadas. O pedido MQCONN ou MQCONNX seleciona qualquer um dos membros ativos do grupo.

Ao se conectar a um gerenciador de filas, ele deve ser local para a tarefa. Ele deve pertencer ao mesmo sistema que o aplicativo IBM MQ.

z/OS No ambiente IMS, o gerenciador de filas deve ser conectado à região de controle do IMS e à região dependente que o programa usa. O gerenciador de filas padrão é especificado no módulo CSQQDEFV quando o IBM MQ for z/OS é instalado.

Com o ambiente TXSeries CICS e o TXSeries for Windows e AIX, o gerenciador de filas deve ser definido como um recurso XA para o CICS.

Para se conectar ao gerenciador de filas padrão, faça a chamada MQCONN especificando um nome que consista inteiramente de espaços em branco ou começando com um caractere nulo (X'00').

Um aplicativo deve ser autorizado para que possa se conectar com êxito a um gerenciador de filas. Para obter mais informações, consulte Protegendo.

A saída de MQCONN é:

- Uma manipulação de conexões (**Hconn**)
- Um código de conclusão
- Um código de razão

Use a manipulação de conexões em chamadas MQI subsequentes.

Se o código de razão indicar que o aplicativo já está conectado a esse gerenciador de filas, a manipulação de conexões que é retornada será a mesma que foi retornada quando o primeiro aplicativo se conectou. O aplicativo não deve emitir a chamada MQDISC nessa situação, pois o aplicativo de chamada espera permanecer conectado.

O escopo da manipulação de conexões é o mesmo que o escopo da manipulação de objetos (consulte “Abrindo objetos usando a chamada MQOPEN” na página 755).

Descrições dos parâmetros são fornecidas na descrição da chamada MQCONN em MQCONN.

A chamada MQCONN falhará se o gerenciador de filas estiver em um estado de quiesce quando a chamada for emitida ou se o gerenciador de filas estiver encerrando.

Escopo de MQCONN ou MQCONNX

O escopo de uma chamada MQCONN ou MQCONNX é geralmente o encadeamento que o emitiu. Ou seja, a manipulação de conexões retornada da chamada é válida apenas dentro do encadeamento que a emitiu. Apenas uma chamada pode ser feita a qualquer momento usando a manipulação. Se for usada a partir de um encadeamento diferente, será rejeitado como inválida. Se houver vários encadeamentos em seu aplicativo e cada um desejar usar chamadas IBM MQ, cada um deve emitir MQCONN ou MQCONNX.

Não é necessário que cada chamada seja feita para o gerenciador de filas mesmo quando um processo fizer várias chamadas MQCONN. No entanto, apenas uma conexão do IBM MQ pode ser feita a partir de um encadeamento por vez. Como alternativa, considere “Conexões compartilhadas (independentes de encadeamento) com MQCONNX” na página 751 para permitir que várias conexões do IBM MQ a partir de um único encadeamento e uma conexão do IBM MQ sejam usadas a partir de qualquer encadeamento.[7]

Se o seu aplicativo estiver em execução como um cliente, ele pode se conectar a mais de um gerenciador de filas em um encadeamento.

Conectando-se a um gerenciador de filas usando a chamada MQCONNX

A chamada MQCONNX é semelhante à chamada MQCONN, mas inclui opções para controlar a maneira com que a chamada funciona.

Como entrada para MQCONNX, é possível fornecer um nome do gerenciador de filas z/OS ou um nome de grupo de filas compartilhadas nos sistemas de filas compartilhadas z/OS. As opções para controlar como a conexão é feita com o gerenciador de filas são fornecidas em uma estrutura chamada MQCNO

A saída de MQCONNX é:

- Uma manipulação de conexões (Hconn)
- Um código de conclusão
- Um código de razão

[7] Ao usar aplicativos multiencadeados com sistemas IBM MQ for AIX or Linux, é necessário assegurar que os aplicativos tenham um tamanho de pilha suficiente para as encadeamentos. Considere usar um tamanho de pilha de 256 KB, ou maior, quando aplicativos multiencadeados estiverem fazendo chamadas MQI, seja por eles mesmos ou, com outros manipuladores de sinal (por exemplo, CICS).

A manipulação de conexões é usada em chamadas MQI subsequentes.

As opções de conexão, configuradas no campo *Options* da estrutura MQCNO, permitem que vários atributos de conexão sejam controlados De uma nota particular são os seguintes grupos de opções:

- As opções de ligação permitem que *aplicativos confiáveis* sejam criados.. Os aplicativos confiáveis implicam que o aplicativo IBM MQ e o agente do gerenciador de fila local se tornem o mesmo processo. Como o processo do agente não precisa mais usar uma interface para acessar o gerenciador de filas, esses aplicativos tornam-se uma extensão do gerenciador de filas. Esse comportamento é solicitado ao especificar a opção MQCNO_FASTPATH_BINDING Para obter mais informações sobre as restrições que se aplicam a aplicativos confiáveis, consulte “Restrições para aplicativos confiáveis” na página 750
- As opções de compartilhamento de identificador permitem que as conexões compartilhadas sejam criadas As conexões compartilhadas podem compartilhar identificadores entre diferentes encadeamentos dentro do mesmo processo Para obter mais informações sobre as conexões compartilhadas, consulte “Conexões compartilhadas (independentes de encadeamento) com MQCONNX” na página 751

O MQCNO também permite que o aplicativo controle como a conexão ao gerenciador de filas é autenticada. As credenciais de autenticação podem ser especificadas em uma estrutura MQCSP que é referenciada a partir da estrutura MQCNO.

Para obter uma descrição completa dos parâmetros para a chamada MQCONNX e os atributos de conexão que podem ser controlados, consulte MQCONNX-Connect queue manager (extended).

Restrições para aplicativos confiáveis
Restrições que se aplicam a aplicativos confiáveis.. Algumas restrições aplicam-se a todas as plataformas e outras são específicas da plataforma

T

- Deve-se desconectar explicitamente aplicativos confiáveis do gerenciador de filas.
- Deve-se parar os aplicativos confiáveis antes de terminar o gerenciador de fila com o comando **endmqm**
- Não deve-se usar sinais assíncronos e interrupções do cronômetro (como `sigkill`) com MQCNO_FASTPATH_BINDING.
- Em todas as plataformas, um encadeamento dentro de um aplicativo confiável não pode se conectar a um gerenciador de filas enquanto outro encadeamento no mesmo processo estiver conectado a um gerenciador de filas diferente.
- Linux AIX Nos sistemas AIX and Linux, deve-se usar mqm como o userID e o groupID efetivos para todas as chamadas MQI. É possível mudar esses IDs antes de fazer uma chamada não MQI que requer autenticação (por exemplo, abrindo um arquivo), mas deve-se mudá-los de volta para mqm antes de fazer a próxima chamada MQI.
- IBM i No IBM i:

  1. Aplicativos confiáveis devem ser executados sob o perfil do usuário QMQM. Não é suficiente o perfil do usuário ser membro do grupo QMQM ou o programa adotar autoridade QMQM. Talvez não seja possível para o perfil do usuário QMQM ser usado para conexão com tarefas interativas ou ser especificado na descrição da tarefa para tarefas executando aplicativos confiáveis. Neste caso, uma abordagem é usar as funções de API de troca de perfil do IBM i , QSYGETPH, QWTSETP, e QSYRLSPH para alterar temporariamente o usuário atual da tarefa para QMQM enquanto os programas do IBM MQ são executados Os detalhes dessas funções, juntamente com um exemplo de seu uso, são fornecidos na seção APIs de Segurança da documentação IBM i*Interfaces de programação de aplicativos* .
  2. Não cancele aplicativos confiáveis usando System-Request Option 2 ou finalizando as tarefas nas quais eles estão sendo executados usando ENDJOB.
- ALW Nos sistemas AIX, Linux, and Windows, aplicativos confiáveis de 32 bits não são suportados. Se você tentar executar um aplicativo confiável de 32 bits, será feito downgrade para uma conexão limite padrão.

Multi *Conexões compartilhadas (independentes de encadeamento) com MQCONNX*

Use estas informações para aprender sobre conexões Compartilhadas com MQCONNX e algumas observações de uso a considerar.

Nota: z/OS Não suportado no IBM MQ for z/OS.

Em Multiplataformas, uma conexão feita com o MQCONN está disponível apenas para o encadeamento que fez a conexão As opções na chamada MQCONNX permitem criar uma conexão que pode ser compartilhada por todos os encadeamentos em um processo. Se o seu aplicativo estiver em execução em um ambiente transacional que requer que chamadas MQI sejam emitidas no mesmo encadeamento, deve-se usar a opção padrão a seguir:

MQCNO_HANDLE_SHARE_NONE
Cria uma conexão não compartilhada.

Na maioria dos outros ambientes, é possível usar uma das opções de conexão compartilhada independente do encadeamento:

MQCNO_HANDLE_SHARE_BLOCK
Cria uma conexão compartilhada. Em uma conexão MQCNO_HANDLE_SHARE_BLOCK, se a conexão estiver atualmente em uso por uma chamada MQI em outro encadeamento, a chamada MQI espera até que a chamada MQI atual tenha sido concluída.

MQCNO_HANDLE_SHARE_NO_BLOCK
Cria uma conexão compartilhada. Em uma conexão MQCNO_HANDLE_SHARE_NO_BLOCK, se a conexão estiver atualmente em uso por uma chamada MQI em outro encadeamento, a chamada MQI falhará imediatamente com uma razão MQRC_CALL_IN_PROGRESS.

Exceto para o ambiente do MTS (Microsoft Transaction Server), o valor padrão é MQCNO_HANDLE_SHARE_NONE. No ambiente do MTS, o valor padrão é MQCNO_HANDLE_SHARE_BLOCK.

Uma manipulação de conexões é retornada da chamada MQCONNX. O identificador pode ser usado pelas chamadas MQI subsequentes a partir de qualquer encadeamento no processo, associando essas chamadas ao identificador retornado de MQCONNX. As chamadas MQI que usam um único identificador compartilhado são serializadas entre encadeamentos.

Por exemplo, a sequência de atividade a seguir é possível com um identificador compartilhado:

1. O encadeamento 1 emite MQCONNX e obtém um identificador compartilhado *h1*
2. O encadeamento 1 abre uma fila e emite uma solicitação get usando *h1*
3. O encadeamento 2 emite uma solicitação put usando *h1*
4. O encadeamento 3 emite uma solicitação put usando *h1*
5. O encadeamento 2 emite MQDISC usando *h1*

Enquanto o identificador estiver em uso por algum encadeamento, o acesso à conexão estará indisponível para outros encadeamentos. Em circunstâncias em que é aceitável que um encadeamento espere a conclusão de qualquer chamada anterior de outro encadeamento, use MQCONNX com a opção MQCNO_HANDLE_SHARE_BLOCK.

No entanto, o bloqueio pode causar dificuldades. Suponha que na etapa “2” na página 751, o encadeamento 1 emita uma solicitação get que espera mensagens que podem ainda não ter chegado (um get com espera). Nesse caso, os encadeamentos 2 e 3 também ficam esperando (bloqueados) pelo tempo que a solicitação get no encadeamento 1 levar. Se preferir que uma chamada MQI retorne com um erro se outra chamada MQI já estiver em execução no identificador, use MQCONNX com a opção MQCNO_HANDLE_SHARE_NO_BLOCK.

Notas de uso da conexão compartilhada

1. Qualquer identificador de objeto (Hobj) criado abrindo um objeto está associado a um Hconn; portanto, para um Hconn compartilhado, os Hobjs também são compartilhados e utilizáveis por qualquer encadeamento que estiver usando o Hconn. De forma semelhante, qualquer unidade de

trabalho iniciada sob um Hconn está associada a esse Hconn; de forma que este também seja compartilhado entre encadeamentos com o Hconn compartilhado.

2. *Qualquer* encadeamento pode chamar MQDISC para desconectar um Hconn compartilhado, não somente o encadeamento que chamou o MQCONNX correspondente. O MQDISC finaliza o Hconn tornando-o indisponível para todos os encadeamentos.
3. Um único encadeamento pode usar vários Hconns compartilhados em série, por exemplo, usar MQPUT para colocar uma mensagem sob um Hconn compartilhado, em seguida, colocar outra mensagem usando outro Hconn compartilhado, com cada operação sendo sob uma unidade de trabalho local diferente.
4. Hconns compartilhados não podem ser usados em uma unidade de trabalho global.

Multi *Uso de opções de chamada MQCONNX com MQ_CONNECT_TYPE*

Use estas informações para entender as diferentes opções de chamada MQCONNX e como elas são usadas com a variável de ambiente **MQ_CONNECT_TYPE** .

Nota: MQ_CONNECT_TYPE tem apenas qualquer efeito para ligações STANDARD Para outras ligações, **MQ_CONNECT_TYPE** é ignorado.

No IBM MQ for Multiplatforms, é possível usar a variável de ambiente **MQ_CONNECT_TYPE** em combinação com o tipo de ligação especificado no campo `Options` da estrutura MQCNO usada em uma chamada MQCONNX.

*Tabela 113. Como as opções de chamada MQCONNX são usadas com a variável de ambiente **MQ_CONNECT_TYPE***

opção de chamada MQCONNX	variável de ambiente MQ_CONNECT_TYPE	Resultado
STANDARD	UNDEFINED	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
STANDARD	CLIENTE	CLIENTE
STANDARD	LOCAL	STANDARD

Se o MQCNO_STANDARD_BINDING não for especificado, será possível usar o MQCNO_NONE, que é padronizado para MQCNO_STANDARD_BINDING.

Autenticação e Identidade para MQCONN e MQCONNX

Use esta tarefa para saber como os aplicativos podem fornecer credenciais que são usadas para autenticação quando eles se conectam ao IBM MQ

A identidade do usuário padrão

Quando um aplicativo usa a interface da fila de mensagens (MQI) para se conectar ao IBM MQ com MQCONN ou MQCONNX, uma identidade do usuário é sempre estabelecida e associada à conexão.

Por padrão, a identidade do usuário inicial é sempre aquela do processo do sistema operacional sob o qual o aplicativo está em execução Essa identidade inicial pode ser suficiente para conexões de aplicativos localmente ligadas ou confiáveis

Quando um aplicativo se conecta a um gerenciador de filas com uma chamada MQCONN, o aplicativo não pode modificar o ID do usuário padrão. No entanto, os mecanismos a seguir podem alterar o ID do usuário associado à conexão:

- Uma saída de segurança do lado do cliente ou do lado do servidor
- Regras de autenticação de canal no gerenciador de filas.
- Um ID do usuário do cliente estabelecido durante a autenticação mútua TLS

Usando MQCONNX para fornecer credenciais

MQCONNX fornece um aplicativo mais controle sobre a identidade que está associada à conexão. Um aplicativo pode fornecer uma estrutura MQCSP como parte das opções de conexão especificadas no parâmetro **ConnectOpts** para MQCONNX. A estrutura do MQCSP pode conter credenciais que são usadas para estabelecer uma identidade do usuário O IBM MQ suporta as seguintes credenciais na estrutura MQCSP:

- Um ID do usuário e senha.
- V 9.4.0 No IBM MQ 9.3.4, um token de autenticação, se o aplicativo se conectar a um gerenciador de filas que é executado em sistemas AIX ou Linux .

A autenticação de conexão do gerenciador de filas e a configuração de autenticação de canal controlam como as credenciais fornecidas por um aplicativo são processadas. Por exemplo, essa configuração afeta os aspectos a seguir:

- Se as credenciais na estrutura MQCSP são validadas e como elas são validadas.
- Se o ID do usuário nas credenciais na estrutura MQCSP está mapeado para outro ID do usuário.
- Se o usuário autenticado é então adotado como o contexto para o aplicativo.

Para obter mais informações sobre a autenticação de conexão, consulte Autenticação de conexão Para obter mais informações sobre a autenticação de canal, consulte Registros de Autenticação de Canal.

Vários dos programas de amostra escritos em C que usam o MQI demonstram como a estrutura MQCSP é usada para fornecer credenciais de autenticação Para obter mais informações, consulte os seguintes programas de amostra:

- “Os programas de amostra Get” na página 1105
- “Os programas de amostra Put” na página 1117
- “O programa de amostra Browser” na página 1093
- “O programa de amostra TLS” na página 1134

Informações relacionadas

Identificando e autenticando usuários usando a estrutura MQCSP

MQCSP-Parâmetros de segurança

Identificando e autenticando usuários

Desconectando programas de um gerenciador de filas usando MQDISC

Use essas informações para aprender sobre desconectar programas de um gerenciador de filas usando MQDISC.

Quando um programa que foi conectado a um gerenciador de filas usando a chamada MQCONN ou MQCONNX tiver concluído toda a interação com o gerenciador de filas, ele interromperá a conexão usando a chamada MQDISC, exceto:

- z/OS Nos aplicativos CICS Transaction Server for z/OS, em que a chamada é opcional a menos que MQCONNX tenha sido usado e você desejar descartar a tag de conexão antes de o aplicativo ser encerrado.
- IBM i No IBM MQ for IBM i, no qual ao efetuar sign off do sistema operacional uma chamada MQDISC implícita é feita.

Como entrada para a chamada MQDISC, deve-se fornecer a manipulação de conexões (Hconn) que foi retornada pelo MQCONN ou MQCONNX ao se conectar ao gerenciador de filas.

No CICS em execução em Multiplataformas, após MQDISC ser chamado o identificador de conexão (Hconn) não é mais válido e não é possível emitir nenhuma chamada MQI adicional até que você chame MQCONN ou MQCONNX novamente. MQDISC faz uma chamada MQCLOSE implícita para quaisquer objetos que ainda estiverem abertos usando essa manipulação.

z/OS Para um cliente conectado ao z/OS, quando uma chamada MQDISC é emitida, ocorre uma confirmação implícita, mas qualquer manipulador de filas que ainda estiver aberto não será encerrado até que o canal realmente termine.

z/OS Ao usar MQCONNX para conectar-se ao IBM MQ for z/OS, MQDISC também finalizará o escopo da tag de conexão estabelecida pelo MQCONNX. No entanto, em um aplicativo CICS, IMS ou RRS, se houver uma unidade ativa de recuperação associada a uma tag de conexão, o MQDISC será rejeitado com um código de razão MQRC_CONN_TAG_NOT_RELEASED.

Descrições dos parâmetros são fornecidas na descrição da chamada MQDISC no MQDISC.

Quando nenhum MQDISC for emitido

Uma conexão padrão não compartilhada (Hconn) será limpa quando a criação do encadeamento finalizar. Uma conexão compartilhada será implicitamente restaurada e desconectada apenas quando o processo inteiro for finalizado. Se o encadeamento que criou a Hconn compartilhada é finalizada enquanto o Hconn ainda existe, o Hconn ainda será utilizável.

Verificação de autoridade

As chamadas MQCLOSE e MQDISC geralmente não executam nenhuma verificação de autoridade.

No curso normal de eventos de uma tarefa que possui a autoridade para abrir ou se conectar a um objeto do IBM MQ fechar ou desconectar-se desse objeto. Mesmo se a autoridade de uma tarefa que foi conectada a ou abriu um objeto IBM MQ for revogada, as chamadas MQCLOSE e MQDISC serão aceitas.

Abrindo e fechando objetos

Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

Para executar qualquer uma das seguintes operações, primeiro deve-se *abra* o objeto IBM MQ relevante:

- Colocar mensagens em uma fila
- Obter (procurar ou recuperar) mensagens de uma fila
- Defina os atributos de um objeto
- Consulta sobre os atributos de qualquer objeto

Use a chamada MQOPEN para abrir o objeto, usando as opções da chamada para especificar o que você deseja fazer com o objeto. A única exceção é se você desejar colocar uma única mensagem em uma fila e então fechar a fila imediatamente. Neste caso, é possível ignorar a etapa de *abrir* usando a chamada MQPUT1 (veja “Colocando uma mensagem em uma fila usando a chamada MQPUT1” na página 773).

Antes de abrir um objeto usando a chamada MQOPEN, deve-se conectar o programa a um gerenciador de filas. Isso é explicado em detalhes, para todos os ambientes, em “Conectando-se e desconectando-se de um gerenciador de filas” na página 747.

Há quatro tipos de objeto IBM MQ que se podem abrir:

- Fila
- Lista de Nomes
- Definição de processo
- Gerenciador de filas

Todos esses objetos são abertos de maneira semelhante usando a chamada MQOPEN. Para obter mais informações sobre os objetos IBM MQ, consulte Tipos de objetos.

É possível abrir o mesmo objeto mais de uma vez, e, a cada vez, você obterá uma nova manipulação de objetos. Você pode desejar procurar mensagens em uma fila usando um manipulador e remover mensagens da mesma fila usando outro manipulador. Isso salva usando os recursos para fechar e reabrir o mesmo objeto. Também é possível abrir uma fila para procurar *e* remover mensagens ao mesmo tempo.

Além disso, é possível abrir vários objetos com uma única chamada MQOPEN e feche-os usando MQCLOSE. Veja “Listas de distribuição” na página 775 para obter informações sobre como fazer isso.

Ao tentar abrir um objeto, o gerenciador de filas verifica se você está autorizado a abrir esse objeto para as opções especificadas na chamada MQOPEN.

Os objetos são fechados automaticamente quando um programa se desconecta do gerenciador de filas. No ambiente IMS, a desconexão é forçada quando um programa começa a processar para um novo usuário apenas uma chamada GU (Get Unique) IMS. Na plataforma IBM i, os objetos são fechados automaticamente quando uma tarefa é concluída.

É uma prática recomendada de programação fechar objetos que você abriu. Use a chamada MQCLOSE para fazer isso.

Use os seguintes links para descobrir mais sobre como abrir e fechar objetos:

- “Abrindo objetos usando a chamada MQOPEN” na página 755
- “Criando filas dinâmicas” na página 763
- “Abrindo filas remotas” na página 763
- “Fechando objetos usando a chamada MQCLOSE” na página 764

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Colocando mensagens em uma fila” na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Abrindo objetos usando a chamada MQOPEN

Use essas informações para aprender sobre como abrir objetos usando a chamada MQOPEN.

Como entrada para a chamada MQOPEN, deve-se fornecer:

- Uma manipulação de conexões. Para CICS aplicativos em z/OS, é possível especificar a constante MQHC_DEF_HCONN (que tem o valor zero) ou usar a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX. Para Multiplataformas, sempre use a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX.

- Uma descrição do objeto que você deseja abrir, usando a estrutura do descritor de objeto (MQOD).
- Uma ou mais opções que controlam a ação da chamada.

A saída de MQOPEN é:

- Uma manipulação de objetos que representa seu acesso ao objeto. Use-o em entrada para quaisquer chamadas MQI subsequentes.
- Uma estrutura do descritor de objeto modificado, se você estiver criando uma fila dinâmica (e for suportada em sua plataforma).
- Um código de conclusão.
- Um código de razão.

Escopo de uma manipulação de objetos

O escopo de uma manipulação de objetos (Hobj) é o mesmo que o escopo de uma manipulação de conexões (Hconn).

Isso é coberto em "Escopo de MQCONN ou MQCONNX" na página 749 e "Conexões compartilhadas (independentes de encadeamento) com MQCONNX" na página 751. No entanto, há considerações adicionais em alguns ambientes:

CICS
Em um programa CICS, é possível usar o identificador somente na mesma tarefa do CICS a partir da qual foi feita a chamada MQOPEN.

IMS e z/OS em lote
No ambiente de IMS e z/OS lote , é possível usar o identificador dentro da mesma tarefa, mas não dentro de nenhuma subtarefa

Descrições dos parâmetros da chamada MQOPEN são fornecidas em MQOPEN.

As seções a seguir descrevem as informações que deve-se fornecer como entrada para MQOPEN.

Identificando objetos (a estrutura MQOD)

Use a estrutura MQOD para identificar o objeto que deseja abrir. Essa estrutura é um parâmetro de entrada para a chamada MQOPEN. (A estrutura é modificada pelo gerenciador de filas quando a chamada MQOPEN é usada para criar uma fila dinâmica.)

Para obter detalhes completos da estrutura MQOD, consulte MQOD.

Para obter informações sobre como usar a estrutura MQOD para listas de distribuição, consulte "Usando a estrutura MQOD" na página 776 em "Listas de distribuição" na página 775.

Resolução do Nome
Como a chamada MQOPEN resolve nomes de fila e de gerenciador de filas.

Nota: Um alias do gerenciador de filas é uma definição de fila remota sem um campo RNAME.

Quando você abre uma fila do IBM MQ, a chamada MQOPEN executa uma função de resolução de nome no nome da fila que você especificar. Isso determina em qual fila o gerenciador de filas executa operações subsequentes. Isso significa que quando você especifica o nome de uma fila de alias ou uma fila remota em seu descritor de objeto (MQOD), a chamada resolve o nome ou para uma fila local ou para uma fila de transmissão. Se uma fila é aberta para qualquer tipo de entrada, procurar ou configurar, ele resolve para uma fila local, se houver uma, e falhará se não houver nenhuma. Ele resolve para uma fila que não seja local apenas se ele for aberto somente para saída, consulta ou saída e consulta. Consulte Tabela 114 na página 757 para obter uma visão geral do processo de resolução de nome. O nome que você fornece em *ObjectQMgrName* será resolvido *antes* que em *ObjectName*.

O Tabela 114 na página 757 também mostra como é possível usar uma definição local de uma fila remota para definir um alias para o nome de um gerenciador de filas. Isso permite selecionar qual fila de transmissão é usada quando se coloca mensagens em uma fila remota, portanto, é possível, por exemplo, usar uma única fila de transmissão para mensagens destinadas a muitos gerenciadores de filas remotas.

Para usar a tabela a seguir, primeiro leia a duas colunas à esquerda, sob o título **Entrada para MQOD** e selecione o caso apropriado. Em seguida, leia toda a linha correspondente, seguindo todas as instruções. Seguindo as instruções nas colunas **Nomes resolvidos**, é possível retornar para as colunas **Entrada para MQOD** e inserir valores conforme orientado ou sair da tabela com os resultados fornecidos. Por exemplo, pode ser necessário inserir o *ObjectName*.

Tabela 114. Resolvendo nomes de filas ao usar MQOPEN

Entrada para MQOD	**Entrada para MQOD**	**Nomes resolvidos**	**Nomes resolvidos**	**Nomes resolvidos**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
Gerenciador de filas em branco ou local	Fila local sem o atributo CLUSTER	Gerenciador de filas locais	Entrada de *ObjectName*	Não aplicável (fila local usada)
Gerenciador de filas em branco	Fila local com o atributo CLUSTER	Gerenciador de filas de cluster selecionado do gerenciamento de carga de trabalho ou gerenciador de filas de cluster específico selecionado em PUT	Entrada de *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE e fila local usada SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)
Gerenciador de filas locais	Fila local com o atributo CLUSTER	Gerenciador de filas locais	Entrada de *ObjectName*	Não aplicável (fila local usada)
Gerenciador de filas em branco ou local	Fila modelo	Gerenciador de filas locais	Nome gerado	Não aplicável (fila local usada)
Gerenciador de filas em branco ou local	Fila de alias com ou sem o atributo CLUSTER	Execute a resolução de nome novamente com *ObjectQMgrName* inalterado, e a entrada *ObjectName* configurada para o *BaseQName* no objeto de definição de fila de alias. Não deve ser resolvido para um alias definido localmente, no qual o *ObjectQMgrName* é especificado, mas pode ser resolvido para um alias em cluster (hospedado em outros gerenciadores de filas) no qual o *ObjectQMgrName* está em branco.		

Tabela 114. Resolvendo nomes de filas ao usar MQOPEN (continuação)

Entrada para MQOD	**Entrada para MQOD**	**Nomes resolvidos**	**Nomes resolvidos**	**Nomes resolvidos**
Gerenciador de filas locais	Fila de alias com o atributo CLUSTER	O alias não deve ser resolvido para uma fila de clusters que não é definida localmente ou uma fila de cluster que possui o mesmo *ObjectName* como o alias.		
Gerenciador de filas em branco	Fila de alias com o atributo CLUSTER	O alias pode ser resolvido para uma fila de clusters com o mesmo *ObjectName* como o alias.		
Gerenciador de filas em branco ou local	Definição local de uma fila remota	Execute a resolução de nome novamente com *ObjectQMgrName* configurado como *RemoteQMgrName* e *ObjectName* configurado como *RemoteQName*. Não é necessário resolver as filas remotas		Nome do atributo *XmitQName*, se não estiver em branco; caso contrário, *RemoteQMgrName* no objeto de definição de fila remota. SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)
Gerenciador de filas em branco	Nenhum objeto local correspondente; fila de cluster localizada	Gerenciador de filas de cluster selecionado do gerenciamento de carga de trabalho ou gerenciador de filas de cluster específico selecionado em PUT	Entrada de *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)
Gerenciador de filas em branco ou local	Nenhum objeto local correspondente; fila de cluster não localizada		Erro, fila não foi localizada	Não-aplicável
Nome do gerenciador de filas no mesmo grupo de filas compartilhadas que o gerenciador de ficas locais	Fila compartilhada local	Gerenciador de filas locais	Entrada de *ObjectName*	Não-aplicável
Nome de uma fila de transmissão local	(Não resolvido)	Insira *ObjectQMgrName*	Entrada de *ObjectName*	Insira *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)

Tabela 114. Resolvendo nomes de filas ao usar MQOPEN (continuação)

Entrada para MQOD	**Entrada para MQOD**	**Nomes resolvidos**	**Nomes resolvidos**	**Nomes resolvidos**
Definição de alias do gerenciador de filas (*RemoteQMgrName* pode ser o gerenciador de filas locais)	(Não resolvido, fila remota)	Execute resolução de nome novamente com *ObjectQMgrName* configurado como *RemoteQMgrName*. Não se deve resolver para filas remotas	Entrada de *ObjectName*	Nome do atributo *XmitQName*, se não estiver em branco; caso contrário, *RemoteQMgrName* no objeto de definição de fila remota. SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)
O gerenciador de filas não é o nome de qualquer objeto local; gerenciadores de filas do cluster ou alias do gerenciador de filas localizado	(Não resolvido)	*ObjectQMgrName* ou gerenciador de filas do cluster específico selecionado em PUT	Entrada de *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)
O gerenciador de filas não é o nome de nenhum objeto local; nenhum objeto de cluster foi localizado	(Não resolvido)	Insira *ObjectQMgrName*	Entrada de *ObjectName*	O atributo *DefXmitQName* do gerenciador de filas no qual o *DefXmitQName* é suportado. SYSTEM.QSG.TRANSMIT.QUEUE (veja a nota)

Notas:

1. *BaseQName* é o nome da fila base da definição da fila de alias.
2. *RemoteQName* é o nome da fila remota da definição local da fila remota.
3. *RemoteQMgrName* é o nome do gerenciador de filas remotas da definição local da fila remota.
4. *XmitQName* é o nome da fila de transmissão da definição local da fila remota.
5. z/OS Para gerenciadores de filas do IBM MQ for z/OS que fazem parte de um grupo de filas compartilhadas (QSG), o nome do grupo de filas compartilhadas pode ser usado em vez do nome do gerenciador de fila local em Tabela 114 na página 757. Se o gerenciador de filas locais não puder abrir a fila de destino ou colocar uma mensagem na fila, a mensagem será transferida para o ObjectQMgrName especificado por meio de enfileiramento intragrupo ou um canal do IBM MQ.
6. Na coluna *ObjectName* da tabela, CLUSTER se refere aos atributos CLUSTER e CLUSNL da fila.
7. A SYSTEM.QSG.TRANSMIT.QUEUE será usada se os gerenciadores de filas locais e remotos estiverem no mesmo grupo de filas compartilhadas; o enfileiramento intragrupo está ativado.
8. Se você tiver designado uma fila de transmissão do cluster diferente para cada canal do emissor de clusters, SYSTEM.CLUSTER.TRANSMIT.QUEUE não poderá ser o nome da fila de transmissão do cluster. Para obter mais informações sobre diversas filas de transmissão do cluster, consulte Armazenamento em cluster: planejando como configurar filas de transmissão do cluster.
9. Na situação em que o gerenciador de filas não é o nome de nenhum objeto local; gerenciadores de filas do cluster ou alias do gerenciador de filas localizados.

   Quando você tiver fornecido um nome do gerenciador de filas usando **ObjectQMgrName** e houver múltiplos canais de cluster com diferentes nomes de cluster conhecidos pelo gerenciador de filas

locais que atingem esse destino, qualquer um desses canais pode ser usado para mover a mensagem, independentemente do nome do cluster da fila de destino.

Isso pode ser inesperado, se você estava antecipando mensagens para essa fila somente para serem enviadas por meio de um canal que tem o mesmo nome do cluster que a fila.

No entanto, o **ObjectQMgrName** tem precedência neste caso e o balanceamento de carga de trabalho do cluster leva em consideração todos os canais que podem atingir esse gerenciador de filas, independentemente do nome do cluster em que eles estão.

Abrir uma fila de alias também abre a fila de base para a qual o alias resolve, e abrir uma fila remota também abre a fila de transmissão. Portanto, não é possível excluir a fila que você especifica ou a fila para a qual ela resolve enquanto a outra está aberta.

Enquanto uma fila de alias é incapaz de resolver para outra fila de alias definida localmente (compartilhada em um cluster ou não), resolver para uma fila de alias do cluster definido remotamente é permitido e, portanto, pode ser especificado como a fila base.

O nome da fila resolvida e o nome do gerenciador de filas resolvido são armazenados nos campos *ResolvedQName* e *ResolvedQMgrName* na MQOD.

Para obter mais informações sobre a resolução de nome em um ambiente de enfileiramento distribuído, consulte O que é a resolução de nome de fila?.

Usando as opções da chamada MQOPEN

No parâmetro **Options** da chamada MQOPEN, deve-se escolher um ou mais opções para controlar o acesso ao objeto que estiver abrindo que foi atribuído a você. Com essas opções, é possível:

- Abrir uma fila e especificar que todas as mensagens colocadas nessa fila devem ser direcionadas para a mesma instância dela
- Abra uma fila para permitir que você coloque mensagens nela
- Abra uma fila para permitir que você procure mensagens nela
- Abra uma fila para permitir que você remova as mensagens dela
- Abra um objeto para permitir que você consulte sobre e configure seus atributos (mas será possível configurar os atributos somente de filas)
- Abra um tópico ou uma sequência de tópicos para publicar mensagens nele
- Informações de contexto associada com uma mensagem
- Denomine um identificador de usuários alternativo para ser usado para verificações de segurança
- Controle a chamada se o gerenciador de filas estiver em um estado de quiesce

Opção MQOPEN para fila de cluster
A ligação usada para a manipulação de fila é obtida a partir do atributo da fila **DefBind**, que pode obter o valor MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED ou MQBND_BIND_ON_GROUP.

Para rotear todas as mensagens colocadas em uma fila usando MQPUT para o mesmo gerenciador de filas pela mesma rota, use a opção MQOO_BIND_ON_OPEN na chamada MQOPEN.

Para especificar que um destino deve ser selecionado no tempo do MQPUT, ou seja, mensagem por mensagem, use a opção MQOO_BIND_NOT_FIXED na chamada MQOPEN.

Para especificar que todas as mensagens em um grupos de mensagens colocados em uma fila usando MQPUT sejam alocadas para a mesma instância de destino, use a opção MQOO_BIND_ON_GROUP na chamada MQOPEN.

MQOO_BIND_ON_OPEN ou MQOO_BIND_ON_GROUP deve ser especificado ao usar grupos de mensagens com clusters para assegurar que todas as mensagens no grupo sejam processadas no mesmo destino.

Se você não especificar nenhuma dessas opções, o padrão, MQOO_BIND_AS_Q_DEF, será usado.

Se você especificar o nome de um gerenciador de filas no MQOD, a fila nesse gerenciador de filas será selecionada. Se o nome do gerenciador de filas estiver em branco, qualquer instância poderá ser selecionada. Consulte o "MQOPEN e clusters" na página 897 para obter informações adicionais.

Se você abrir uma fila de clusters usando uma definição QALIAS, alguns atributos da fila são definidos pela fila de alias e não a fila base. Os atributos de cluster estão entre os atributos da definição de fila base que são substituídos pela fila de alias. Por exemplo, no seguinte trecho, a fila de clusters é aberta com MQOO_BIND_NOT_FIXED e não MQOO_BIND_ON_OPEN. A definição de fila de clusters é anunciada em todo o cluster, a definição de fila de alias é local para o gerenciador de filas.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

Opção de MQOPEN para colocação de mensagens
Para abrir uma fila ou um tópico para colocar mensagens nele, use a opção MQOO_OUTPUT.

Opção MQOPEN para procurar mensagens
Para abrir uma fila de modo que seja possível *procurar* as mensagens nele, use a chamada MQOPEN com a opção MQOO_BROWSE.

Isso cria um *cursor de procura* que o gerenciador de filas usa para identificar a próxima mensagem na fila. Para obter mais informações, consulte "Procurando mensagens em uma fila" na página 814.

Nota:

1. Não é possível procurar as mensagens em uma fila remota; não abra uma fila remota usando a opção MQOO_BROWSE.
2. Não é possível especificar essa opção ao abrir uma lista de distribuição. Para obter informações adicionais sobre listas de distribuição, consulte "Listas de distribuição" na página 775.
3. Use o MQOO_CO_OP em conjunto com MQOO_BROWSE se estiver usando a navegação cooperativa; consulte Opções

Opções de MQOPEN para remoção de mensagens
Três opções controlam a abertura de uma fila para remover as mensagens dela.

É possível usar somente uma delas em qualquer chamada MQOPEN. Essas opções definem se o seu programa tem acesso exclusivo ou compartilhado para a fila. *Acesso exclusivo* significa que, até você fechar a fila, apenas é possível remover mensagens dela. Se outro programa tenta abrir a fila para remover mensagens, a chamada MQOPEN falha. *Acesso compartilhado* significa que mais de um programa pode remover mensagens da fila.

A abordagem mais aconselhável é aceitar o tipo de acesso que foi destinado para a fila quando a fila foi definida. A definição de fila envolveu a configuração do **Shareability** e o Atributos **DefInputOpenOption**. Para aceitar esse acesso, use a opção MQOO_INPUT_AS_Q_DEF. Consulte Tabela 115 na página 761 para ver como a definição desses atributos afeta o tipo de acesso que você receberá quando usar esta opção.

Tabela 115. Como atributos da fila e opções de chamada MQOPEN afetam o acesso a filas

Atributos da Fila		**Tipo de acesso com as opções MQOPEN**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
SHAREABLE	SHARED	compartilhado	compartilhado	exclusivo
SHAREABLE	EXCLUSIVE	exclusivo	compartilhado	exclusivo
NOT_SHAREABLE*	SHARED*	exclusivo	exclusivo	exclusivo
NOT_SHAREABLE	EXCLUSIVE	exclusivo	exclusivo	exclusivo

Nota: * Embora seja possível definir que uma fila tenha essa combinação de atributos, a opção de abertura de entrada padrão é substituída pelo atributo de compartilhamento.

Alternativamente:

- Se você souber que seu aplicativo pode funcionar com êxito, mesmo se outros programas puderem remover mensagens da fila ao mesmo tempo, use a opção MQOO_INPUT_SHARED. O Tabela 115 na página 761 mostra como, em alguns casos, você terá acesso exclusivo à fila, mesmo com essa opção.
- Se você souber que seu aplicativo pode funcionar com êxito somente se outros programas forem impedidos de remover mensagens da fila ao mesmo tempo, use a opção MQOO_INPUT_EXCLUSIVE.

Nota:

1. Não é possível remover mensagens de uma fila remota. Portanto, não é possível abrir uma fila remota usando qualquer uma das opções MQOO_INPUT_*.
2. Não é possível especificar essa opção ao abrir uma lista de distribuição. Veja informações adicionais na publicação "Listas de distribuição" na página 775.

Opções de MQOPEN para configuração e consulta de atributos
Para abrir uma fila de modo que seja possível configurar seus atributos, use a opção MQOO_SET.

Não é possível configurar os atributos de qualquer outro tipo de objeto (consulte "Consultando e configurando atributos de objeto" na página 862).

Para abrir um objeto para que seja possível consultar sobre seus atributos, use a opção MQOO_INQUIRE.

Nota: Não é possível especificar essa opção ao abrir uma lista de distribuição.

Opções de MQOPEN relacionadas ao contexto da mensagem
Se deseja conseguir associar as informações a uma mensagem ao colocá-la em uma fila, deve-se usar uma das opções de contexto da mensagem ao abrir a fila.

As opções permitem diferenciar entre as informações de contexto que se relacionam ao *usuário* que originou a mensagem e que se relacionam ao *aplicativo* que originou a mensagem. Além disso, é possível escolher configurar as informações de contexto ao colocar a mensagem na fila ou ter o contexto obtido automaticamente a partir de outro identificador de filas.

Conceitos relacionados
"Contexto da mensagem" na página 48
As informações de *Contexto da Mensagem* permitem que o aplicativo recupere a mensagem para descobrir sobre o originador da mensagem.

"Controlando informações de contexto da mensagem" na página 771
Quando você usa a chamada MQPUT ou MQPUT1 para colocar uma mensagem em uma fila, é possível especificar se o gerenciador de filas deve incluir algumas informações de contexto padrão para o descritor de mensagens. Aplicativos que possuem o nível apropriado de autoridade podem incluir informações de contexto adicionais. É possível usar o campo de opções na estrutura MQPMO para controlar informações de contexto.

Opção MQOPEN para a autoridade de usuário alternativo
Quando você tenta abrir um objeto usando a chamada MQOPEN, o gerenciador de filas verifica se você tem autoridade para abrir esse objeto. Se você não estiver autorizado, a chamada falhará.

No entanto, os programas do servidor podem desejar que o gerenciador de filas verifique a autorização do usuário para o qual estão trabalhando em vez da autorização própria do servidor. Para fazer isso, eles devem usar a opção MQOO_ALTERNATE_USER_AUTHORITY da chamada MQOPEN e especificar o ID de usuário alternativo no campo *AlternateUserId* da estrutura MQOD. Geralmente, o servidor poderia obter o ID do usuário das informações de contexto na mensagem que ele está processando.

z/OS *MQOPEN option for queue manager quiescing*
If you use the MQOPEN call when the queue manager is in a quiescing state, the call might fail, depending on which environment you are using.

In the CICS environment on z/OS, if you use the MQOPEN call when the queue manager is in a quiescing state, the call always fails.

In other z/OS and Multiplatforms environments, the call fails when the queue manager is quiescing only if you use the MQOO_FAIL_IF_QUIESCING option of the MQOPEN call.

Opção de MQOPEN para resolver nomes de filas locais
Ao abrir uma fila local, de alias ou de modelo, a fila local é retornada.

No entanto, ao abrir uma fila remota ou fila de clusters, os campos *ResolvedQName* e *ResolvedQMgrName* da estrutura MQOD são preenchidos com os nomes da fila remota e do gerenciador de filas remotas encontrados na definição de fila remota ou com a fila de clusters remota escolhida.

Use a opção MQOO_RESOLVE_LOCAL_Q da chamada MQOPEN para preencher o *ResolvedQName* na estrutura MQOD com o nome da fila local que foi aberta. O *ResolvedQMgrName* é preenchido de forma semelhante com o nome do gerenciador de filas locais que hospeda a fila local. Esse campo está disponível somente com a Versão 3 da estrutura MQOD; se a estrutura for anterior à Versão 3, MQOO_RESOLVE_LOCAL_Q será ignorado sem que um erro seja retornado.

Se você especificar MQOO_RESOLVE_LOCAL_Q ao abrir, por exemplo, uma fila remota, *ResolvedQName* será o nome da fila de transmissão na qual as mensagens serão colocadas. *ResolvedQMgrName* é o nome do gerenciador de filas locais que hospeda a fila de transmissão.

Criando filas dinâmicas
Use uma fila dinâmica quando não precisar da fila depois que o seu aplicativo for finalizado.

Por exemplo, você poderia usar uma fila dinâmica para a sua fila de resposta. Especifique o nome da fila de resposta no campo *ReplyToQ* da estrutura MQMD quando colocar uma mensagem em uma fila (consulte "Definindo mensagens usando a estrutura MQMD" na página 766).

Para criar uma fila dinâmica, use um modelo conhecido como uma fila modelo, junto com a chamada MQOPEN. Você cria uma fila modelo usando os comandos IBM MQ ou as operações e painéis de controle. A fila dinâmica que você cria assume os atributos da fila modelo.

Quando chamar MQOPEN, especifique o nome da fila modelo no campo *ObjectName* da estrutura MQOD. Quando a chamada é concluída, o campo *ObjectName* é configurado como o nome da fila dinâmica que é criada. Além disso, o campo *ObjectQMgrName* é configurado como o nome do gerenciador de filas local.

É possível especificar o nome da fila dinâmica criada de três maneiras:

- Forneça o nome completo que você deseja no campo *DynamicQName* da estrutura MQOD.
- Especifique um prefixo (menos de 33 caracteres) para o nome e deixe que o gerenciador de filas gere o restante do nome. Isso significa que o gerenciador de filas gera um nome exclusivo, mas você ainda tem algum controle (por exemplo, você talvez queira que cada usuário use um certo prefixo ou queira dar uma classificação de segurança especial para filas com um certo prefixo em seus nomes). Para usar esse método, especifique um asterisco (*) para o último caractere que não estiver em branco do campo *DynamicQName*. Não especifique um único asterisco (*) para o nome da fila dinâmica.
- Deixe que o gerenciador de filas gere o nome completo. Para usar esse método, especifique um asterisco (*) na primeira posição do caractere do campo *DynamicQName*.

Para obter informações adicionais sobre esses métodos, consulte a descrição do campo DynamicQName.

Há mais informações sobre filas dinâmicas em filas Dinâmicas e Modelo.

Abrindo filas remotas
Uma fila remota é uma fila que é de propriedade de um gerenciador de filas diferente daquela à qual o aplicativo está conectado.

Para abrir uma fila remota, use a chamada MQOPEN como para uma fila local. É possível especificar o nome da fila da seguinte forma:

1. No campo *ObjectName* da estrutura MQOD, especifique o nome da fila remota como conhecido pelo gerenciador de filas *local*.

Nota: Neste caso, deixe o campo *ObjectQMgrName* em branco.

2. No campo *ObjectName* da estrutura MQOD, especifique o nome da fila remota, como conhecido pelo gerenciador de filas *remoto*. No campo *ObjectQMgrName*, especifique também:
   - O nome da fila de transmissão que tem o mesmo nome que o gerenciador de filas remotas. O nome e as letras maiúsculas, minúsculas ou uma combinação delas deve corresponder *exatamente*.
   - O nome de um objeto de alias do gerenciador de filas que é resolvido para o gerenciador de filas de destino ou para a fila de transmissão.

   Isso informa ao gerenciador de filas o destino da mensagem, assim como a fila de transmissão na qual precisa ser colocada para chegar lá.
3. Se *DefXmitQname* for suportado, no campo *ObjectName* da estrutura MQOD, especifique o nome da fila remota como conhecido pelo gerenciador de filas *remoto*.

   Nota: Configure o campo *ObjectQMgrName* para o nome do gerenciador de filas remoto (neste caso, não pode ser deixado em branco).

Somente nomes locais são validados quando você chama MQOPEN; a última verificação é para a existência da fila de transmissão a ser usada.

Esses métodos estão resumidos em Tabela 114 na página 757.

Fechando objetos usando a chamada MQCLOSE

Para fechar um objeto, use a chamada MQCLOSE.

Se o objeto for uma fila, observe o seguinte:

- Não é preciso esvaziar uma fila dinâmica temporária antes de fechá-la.

  Ao fechar uma fila dinâmica temporária, ela será excluída juntamente com quaisquer mensagens que ainda possam estar nela. Isso é verdadeiro mesmo se houver chamadas MQGET, MQPUT ou MQPUT1 não confirmadas pendentes na fila.
- z/OS No IBM MQ for z/OS, se você tiver quaisquer solicitações MQGET com uma opção MQGMO_SET_SIGNAL pendente para essa fila, elas serão canceladas.
- Se você abriu a fila usando a opção MQOO_BROWSE, o cursor de pesquisa é destruído.

O fechamento não está relacionado ao ponto de sincronização, portanto, é possível fechar filas antes ou após o ponto de sincronização.

Como entrada para a chamada MQCLOSE, deve-se fornecer:

- Uma manipulação de conexões. Use a mesma alça de conexão usada para abri-la, Para CICS aplicativos em z/OS, é possível especificar alternativamente a constante MQHC_DEF_HCONN (que tem o valor zero).
- O manipulador do objeto que você deseja fechar. Obtenha este a partir da saída da chamada MQOPEN.
- MQCO_NONE no campo *Options* (a menos que você esteja fechando uma fila dinâmica permanente).
- A opção de controle para determinar se o gerenciador de filas deve excluir a fila mesmo se ainda houver mensagens nela (ao fechar uma fila dinâmica permanente).

A saída de MQCLOSE é:

- Um código de conclusão
- Um código de razão
- A manipulação de objetos, redefinido para o valor MQHO_UNUSABLE_HOBJ

Descrições dos parâmetros da chamada MQCLOSE são fornecidas em MQCLOSE.

Colocando mensagens em uma fila

Use estas informações para aprender como colocar mensagens em uma fila.

Use a chamada MQPUT para colocar mensagens na fila. É possível usar o MQPUT repetidamente para colocar várias mensagens na mesma fila, após a chamada MQOPEN inicial. Chame MQCLOSE quando você tiver concluído todas as suas mensagens na fila.

Se você deseja colocar uma mensagem única em uma fila e fechar a fila imediatamente depois, é possível usar a chamada MQPUT1. MQPUT1 executa as mesmas funções que a seguinte sequência de chamadas:

- MQOPEN
- MQPUT
- MQCLOSE

, no entanto, se você tiver mais de uma mensagem para colocar na fila, é mais eficiente usar a chamada MQPUT. Isso depende do tamanho da mensagem e da plataforma em que se está trabalhando.

Use os seguintes links para descobrir mais sobre colocar mensagens em uma fila:

- “Colocando mensagens em uma fila local usando a chamada MQPUT” na página 766
- “Colocando mensagens em uma fila remota” na página 771
- “Configurando propriedades de uma mensagem” na página 771
- “Controlando informações de contexto da mensagem” na página 771
- “Colocando uma mensagem em uma fila usando a chamada MQPUT1” na página 773
- “Listas de distribuição” na página 775
- “Alguns casos em que as chamadas put falham” na página 780

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Abrindo e fechando objetos” na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Obtendo mensagens de uma fila” na página 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Colocando mensagens em uma fila local usando a chamada MQPUT

Use estas informações para aprender sobre colocar mensagens em uma fila local usando a chamada MQPUT.

Como entrada para a chamada MQPUT, deve-se fornecer:

- Uma manipulaçao de conexões (Hconn).
- Um identificador de fila (Hobj).
- Uma descrição da mensagem que você deseja colocar na fila. Isso no formato de uma estrutura do descritor de mensagens (MQMD).
- Informações de controle, no formato de uma estrutura de opções put-message (MQPMO).
- O comprimento dos dados contidos dentro da mensagem (MQLONG).
- Os dados da mensagem em si.

A saída da chamada MQPUT é a seguinte:

- Um código de razão (MQLONG)
- Um código de conclusão (MQLONG)

Se a chamada for concluída com sucesso, ela também retornará a estrutura de suas opções e a estrutura de ser descritor de mensagens. A chamada modifica a estrutura de suas opções para mostrar o nome da fila e o gerenciador de filas para o qual a mensagem foi enviada. Se você solicitar que o gerenciador de filas gere um valor exclusivo para o identificador da mensagem para a qual está efetuando put (especificando zero binário no campo *MsgId* da estrutura MQMD), a chamada insere o valor no campo *MsgId* antes de retornar essa estrutura para você. Reconfigure esse valor antes de emitir outro MQPUT.

Existe uma descrição da chamada MQPUT em MQPUT.

Para obter mais descrição sobre as informações necessárias como entrada para a chamada MQPUT, consulte os links a seguir:

- “Especificando identificadores” na página 766
- “Definindo mensagens usando a estrutura MQMD” na página 766
- “Especificando opções usando a estrutura MQPMO” na página 767
- “Os dados em sua mensagem” na página 769
- “Efetuando put de mensagens: usando identificadores de mensagens” na página 770

Especificando identificadores

Para a manipulação de conexões (*Hconn*) em aplicativos CICS no z/OS, é possível especificar a constante MQHC_DEF_HCONN (que tem o valor zero) ou usar a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX. Para outros aplicativos, use sempre a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX.

Independentemente do ambiente no qual está trabalhando, use o mesmo identificador de fila (*Hobj*) que é retornado pela chamada MQOPEN.

Definindo mensagens usando a estrutura MQMD

A estrutura do descritor de mensagens (MQMD) é um parâmetro de entrada/saída para as chamadas MQPUT e MQPUT1. Use-a para definir a mensagem que está colocando em uma fila.

Se MQPRI_PRIORITY_AS_Q_DEF ou MQPER_PERSISTENCE_AS_Q_DEF for especificado para a mensagem e a fila for uma fila de cluster, os valores usados serão aqueles da fila para a qual a chamada MQPUT é resolvida. Se essa fila estiver desativada para MQPUT, a chamada falhará. Consulte Configurando um cluster de gerenciador de filas para obter mais informações.

Nota: Use MQPMO_NEW_MSG_ID e MQPMO_NEW_CORREL_ID antes de efetuar put de uma nova mensagem para assegurar que *MsgId* e *CorrelId* sejam exclusivos. Os valores nesses campos são retornados em um MQPUT bem-sucedido.

Há uma introdução para as propriedades de mensagens que o MQMD descreve no "Mensagens IBM MQ" na página 18 e há uma descrição da própria estrutura no MQMD.

Especificando opções usando a estrutura MQPMO

Use a estrutura de MQPMO (Put Message Option) para passar opções para as chamadas MQPUT e MQPUT1.

As seções a seguir fornecem ajuda sobre como preencher os campos dessa estrutura. Há uma descrição da estrutura em MQPMO.

A estrutura inclui os campos a seguir:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

O conteúdo desses campos é o seguinte:

StrucId
Identifica a estrutura como uma estrutura de opções put-message. Esse é um campo de quatro caracteres. Sempre especifique MQPMO_STRUC_ID.

Versão
Descreve o número da versão da estrutura. O padrão é MQPMO_VERSION_1. Se inserir MQPMO_VERSION_2, será possível usar listas de distribuição (consulte "Listas de distribuição" na página 775). Se você inserir MQPMO_VERSION_3, será possível usar identificadores de mensagens e propriedades de mensagens. Se inserir MQPMO_CURRENT_VERSION, seu aplicativo será configurado para sempre usar o nível mais recente.

Opções
Isso controla o seguinte:

- Se a operação put está incluída em uma unidade de trabalho
- Quanto de informações de contexto está associado a uma mensagem
- De onde as informações de contexto são obtidas
- Se a chamada falhará se o gerenciador de filas estiver em um estado de quiesce
- Se agrupamento ou segmentação é permitido
- Geração de um novo identificador de mensagem e identificador de correlação
- A ordem na qual as mensagens e os segmentos são colocados em uma fila
- Se nomes de filas locais devem ser resolvidos

Se deixar o campo *Options* configurado para o valor padrão (MQPMO_NONE), a mensagem para a qual put foi efetuado tem informações de contexto padrão associadas a ela.

A forma como a chamada opera com pontos de sincronização é determinada pela plataforma. O padrão de controle do ponto de sincronização é yes para z/OS e no para Multiplataformas

Context
Indica o nome do identificador de fila do qual deseja que as informações de contexto sejam copiadas (se solicitado no campo *Options*).

Para obter uma introdução ao contexto da mensagem, consulte "Contexto da mensagem" na página 48. Para obter informações sobre como usar a estrutura MQPMO para controlar as informações de contexto em uma mensagem, consulte "Controlando informações de contexto da mensagem" na página 771.

ResolvedQName
Contém o nome (após a resolução de qualquer nome de alias) da fila que foi aberta para receber a mensagem. Esse é um campo de saída.

ResolvedQMgrName
Contém o nome (após a resolução de qualquer nome alternativo) do gerenciador de filas que possui a fila em *ResolvedQName*. Esse é um campo de saída.

O MQPMO também pode acomodar campos necessários para listas de distribuição (consulte "Listas de distribuição" na página 775). Se desejar usar esse recurso, a Versão 2 da estrutura MQPMO será usada. Inclui os campos a seguir:

RecsPresent
Esse campo contém o número de filas na lista de distribuição; ou seja, o número de Put Message Records (MQPMR) e os Response Records (MQRR) correspondentes presentes.

O valor inserido pode ser o mesmo que o número de Object Records fornecidos na chamada MQOPEN. No entanto, se o valor for menor que o número de Object Records fornecido na chamada MQOPEN ou se você não fornecer Put Message Records, os valores das filas não definidos serão obtidos dos valores padrão fornecidos pelo descritor de mensagens. Além disso, se o valor for maior que o número de Object Records fornecido, os Put Message Records em excesso serão ignorados.

É recomendável executar um dos seguintes:

- Se desejar receber um relatório ou resposta de cada destino, insira o mesmo valor que aparece na estrutura MQOR e use MQPMRs que contém campos *MsgId*. Inicialize esses campos *MsgId* para zeros ou especifique MQPMO_NEW_MSG_ID.

  Quando tiver colocado a mensagem na fila, os valores de *MsgId* que o gerenciador de filas criou serão disponibilizados nos MQPMRs; será possível usá-los para identificar qual destino está associado a cada relatório ou resposta.
- Se não deseja receber relatórios ou respostas, escolha uma das opções a seguir:
  1. Se desejar identificar destinos que falham imediatamente, pode ser que queira inserir o mesmo valor no campo *RecsPresent* como aparece na estrutura MQOR e fornecer MQRRs para identificar esses destinos. Não especifique nenhum MQPMRs.
  2. Se não desejar identificar destinos com falha, insira zero no campo *RecsPresent* e não forneça MQPMRs nem MQRRs.

Nota: Se estiver usando MQPUT1, o número de Response Record Pointers e Response Record Offsets deve ser zero.

Para obter uma descrição completa de Put Message Records (MQPMR) e Response Records (MQRR), consulte MQPMR e MQRR.

PutMsgRecFields
Isso indica quais campos estão presentes em cada Put Message Record (MQPMR). Para obter uma lista desses campos, consulte "Usando a estrutura MQPMR" na página 779.

PutMsgRecOffset* e *PutMsgRecPtr
Ponteiros (geralmente em C) e deslocamentos (geralmente em COBOL) são usados para direcionar os Put Message Records (consulte "Usando a estrutura MQPMR" na página 779 para obter uma visão geral da estrutura MQPMR).

Use o campo *PutMsgRecPtr* para especificar um ponteiro para o primeiro Put Message Record ou o campo *PutMsgRecOffset* para especificar o deslocamento do primeiro Put Message Record. Esse é

o deslocamento do início de MQPMO. Dependendo do campo *PutMsgRecFields*, insira um valor não nulo para *PutMsgRecOffset* ou *PutMsgRecPtr*.

ResponseRecOffset e ResponseRecPtr
Você também usa ponteiros e deslocamentos para direcionar Response Records (consulte "Usando a estrutura MQRR" na página 778 para obter informações adicionais sobre Response Records).

Use o campo *ResponseRecPtr* para especificar um ponteiro para o Response Record ou o campo *ResponseRecOffset* para especificar o deslocamento do primeiro Response Record. Esse é o deslocamento do início da estrutura MQPMO. Insira um valor não nulo para *ResponseRecOffset* ou *ResponseRecPtr*.

Nota: Se estiver usando MQPUT1 para colocar mensagens em uma lista de distribuição, *ResponseRecPtr* deve ser nulo ou zero e *ResponseRecOffset* deve ser zero.

A Versão 3 da estrutura MQPMO inclui adicionalmente os campos a seguir:

OriginalMsgHandle
O uso que é possível fazer desse campo depende do valor do campo *Action*. Se estiver efetuando put de uma nova mensagem com propriedades de mensagem associadas, configure esse campo para o identificador de mensagem criado anteriormente e ative as propriedades. Se estiver encaminhando, respondendo ou gerando um relatório em resposta a uma mensagem anteriormente recuperada, esse campo contém o identificador dessa mensagem.

NewMsgHandle
Se especificar um *NewMsgHandle*, quaisquer propriedades associadas ao identificados substituem as propriedades associadas a *OriginalMsgHandle*. Para obter mais informações, consulte Action (MQLONG).

Ação
Use esse campo para especificar o tipo de put que está sendo executado. Valores possíveis e seus significados são os seguintes:

MQACTP_NEW
Esta é uma nova mensagem não relacionada a qualquer outra.

MQACTP_FORWARD
Esta mensagem foi recuperada anteriormente e agora está sendo encaminhada.

MQACTP_REPLY
Esta mensagem é uma resposta a uma mensagem recuperada anteriormente.

MQACTP_REPORT
Esta mensagem é um relatório gerado como resultado de uma mensagem recuperada anteriormente.

Para obter mais informações, consulte Action (MQLONG).

PubLevel
Se esta mensagem for uma publicação, será possível definir esse campo para determinar quais as assinaturas a recebem. Somente assinaturas com um *SubLevel* menor ou igual a esse valor receberão essa publicação. O valor padrão é 9, que é o nível mais alto e significa que assinaturas com qualquer *SubLevel* podem receber esta publicação.

Os dados em sua mensagem

Forneça o endereço do buffer que contém seus dados no parâmetro **Buffer** da chamada MQPUT. É possível incluir qualquer coisa nos dados em suas mensagens. A quantia de dados nas mensagens, no entanto, afeta o desempenho do aplicativo que as está processando.

O tamanho máximo dos dados é determinado por:

- O atributo **MaxMsgLength** do gerenciador de filas
- O atributo **MaxMsgLength** da fila na qual você está colocando a mensagem
- O tamanho de qualquer cabeçalho da mensagem incluído pelo IBM MQ (incluindo o cabeçalho de mensagens não entregues, MQDLH e o cabeçalho da lista de distribuição, MQDH)

O atributo **MaxMsgLength** do gerenciador de filas retém o tamanho da mensagem que o gerenciador de filas pode processar. O padrão é de 100 MB para todos os produtos IBM MQ na V6 ou superior.

Para determinar o valor desse atributo, use a chamada MQINQ no objeto do gerenciador de filas. Para mensagens grandes, é possível mudar esse valor.

O atributo **MaxMsgLength** de uma fila determina o tamanho máximo da mensagem que é possível colocar na fila. Se tentar colocar uma mensagem com um tamanho maior do que o valor desse atributo, a chamada MQPUT falhará. Se estiver colocando uma mensagem em uma fila remota, o tamanho máximo de mensagem que é possível colocar com sucesso é determinado pelo atributo **MaxMsgLength** da fila remota, de quaisquer filas de transmissão intermediárias nas quais a mensagem é colocada ao longo da rota para seu destino e dos canais usados.

Para uma operação MQPUT, o tamanho da mensagem deve ser menor que ou igual ao atributo **MaxMsgLength** da fila e do gerenciador de filas. Os valores desses atributos são independentes, mas é recomendado configurar o *MaxMsgLength* da fila para um valor menor ou igual ao do gerenciador de filas.

O IBM MQ inclui informações do cabeçalho nas mensagens nas circunstâncias a seguir:

- Ao colocar uma mensagem em uma fila remota, o IBM MQ inclui uma estrutura de cabeçalho de transmissão (MQXQH) na mensagem. Essa estrutura inclui o nome da fila de destino e seu gerenciador de filas proprietário.
- Se o IBM MQ não puder entregar uma mensagem em uma fila remota, ele tenta colocar a mensagem na fila de mensagens não entregues. Ela inclui uma estrutura MQDLH na mensagem. Essa estrutura inclui o nome da fila de destino e a razão pela qual a mensagem foi colocada na fila de mensagens não entregues.
- Se você deseja enviar uma mensagem para várias filas de destino, o IBM MQ inclui um cabeçalho MQDH na mensagem. Isso descreve os dados que estão presentes em uma mensagem, pertencente a uma lista de distribuição, em uma fila de transmissão. Considere isso ao escolher um valor ideal para o comprimento máximo da mensagem.
- Se a mensagem for um segmento ou uma mensagem em um grupo, o IBM MQ pode incluir um MQMDE.

Essas estruturas são descritas em MQDH e MQMDE.

Se suas mensagens forem do tamanho máximo permitido para essas filas, a adição desses cabeçalhos significa que as operações put falham porque agora as mensagens são muito grandes. Para reduzir a possibilidade das operações put com falha:

- Torne o tamanho de suas mensagens menor do que o atributo **MaxMsgLength** das filas de transmissão e de mensagens não entregues. Permita pelo menos o valor da constante MQ_MSG_HEADER_LENGTH (mais para listas de distribuição grandes).
- Certifique-se de que o atributo **MaxMsgLength** da fila de mensagens não entregues esteja configurado para o mesmo que o *MaxMsgLength* do gerenciador de filas que possui a fila de mensagens não entregues.

Os atributos do gerenciador de filas e as constantes de enfileiramento de mensagens estão descritos em Atributos do gerenciador de filas.

z/OS Para obter informações sobre como as mensagens não entregues são tratadas em um ambiente de enfileiramento distribuído, consulte Mensagens não entregues/não processadas.

Efetuando put de mensagens: usando identificadores de mensagens

Dois identificadores de mensagens estão disponíveis na estrutura MQPMO, *OriginalMsgHandle* e *NewMsgHandle*. O relacionamento entre esses identificadores de mensagens é definido pelo valor do campo *Action* de MQPMO.

Para obter detalhes integrais, consulte Action (MQLONG). Um identificador de mensagem não é necessariamente requerido para efetuar put de uma mensagem. Seu propósito é associar propriedades a uma mensagem, portanto, é necessário somente se você estiver usando as propriedades de mensagem.

Colocando mensagens em uma fila remota

Quando você deseja colocar uma mensagem em uma fila remota (ou seja, uma fila pertencente a um gerenciador de filas diferente do que aquele ao qual seu aplicativo está conectado) em vez de uma fila local, a única consideração adicional é como especificar o nome da fila quando você a abre.Isso é descrito no "Abrindo filas remotas" na página 763. Não há mudança em como você usa a chamada MQPUT ou MQPUT1 para uma fila local.

Para obter mais informações sobre como usar filas de transmissão e remotas, consulte IBM MQtécnicas de enfileiramento distribuído.

Configurando propriedades de uma mensagem

Chamada MQSETMP para cada propriedade que você deseja configurar. Ao efetuar put da mensagem, configure o identificador de mensagem e os campos de ação da estrutura MQPMO.

Para associar propriedades a uma mensagem, a mensagem deve ter um identificador de mensagem. Crie um identificador de mensagem usando a chamada de função MQCRTMH. Chame MQSETMP especificando esse identificador de mensagem para cada propriedade que deseja configurar. Um programa de amostra, amqsstma.c, é fornecido para ilustrar o uso de MQSETMP.

Se essa for uma nova mensagem, ao colocá-la em uma fila, usando MQPUT ou MQPUT1, configure o campo OriginalMsgHandle em MQPMO para o valor desse identificador de mensagem e configure o campo Action de MQPMO para MQACTP_NEW (esse é o valor padrão).

Se esta for uma mensagem anteriormente recuperada e agora você está encaminhado ou respondendo a mesma ou enviando um relatório em resposta a ela, coloque o identificador de mensagem original no campo OriginalMsgHandle de MQPMO e o novo identificador de mensagem no campo NewMsgHandle. Configure o campo Action para MQACTP_FORWARD, MQACTP_REPLY ou MQACTP_REPORT, conforme apropriado.

Se tiver propriedades em um cabeçalho MQRFH2 de uma mensagem anteriormente recuperada, será possível convertê-las para propriedades do identificador de mensagem usando a chamada MQBUFMH.

Se estiver colocando sua mensagem para uma fila em um gerenciador de filas em um nível anterior ao IBM WebSphere MQ 7.0, que não pode processar as propriedades de mensagem, será possível configurar o parâmetro PropertyControl na definição de canal para especificar como as propriedades devem ser tratadas.

Controlando informações de contexto da mensagem

Quando você usa a chamada MQPUT ou MQPUT1 para colocar uma mensagem em uma fila, é possível especificar se o gerenciador de filas deve incluir algumas informações de contexto padrão para o descritor de mensagens. Aplicativos que possuem o nível apropriado de autoridade podem incluir informações de contexto adicionais. É possível usar o campo de opções na estrutura MQPMO para controlar informações de contexto.

As informações de contexto da mensagem permitem que o aplicativo que recupera a mensagem descubra sobre o originador da mensagem. Todas as informações de contexto são armazenadas nos campos de contexto do descritor de mensagens. O tipo de informação é classificado em identidade, origem e informações de contexto do usuário.

Para controlar informações de contexto, use o campo *Options* na estrutura MQPMO.

Se você não especificar opções para informações de contexto, o gerenciador de filas sobrescreverá as informações de contexto que já possam estar no descritor de mensagens com as informações de identidade e contexto que foram geradas para sua mensagem. Isso é o mesmo que especificar a opção MQPMO_DEFAULT_CONTEXT.Talvez você queira essas informações de contexto padrão ao criar uma nova mensagem (por exemplo, ao processar a entrada do usuário a partir de uma tela de consulta).

Se você não deseja informações de contexto associadas a sua mensagem, use a opção MQPMO_NO_CONTEXT. Ao colocar uma mensagem sem contexto, todas as verificações de autoridade feitas pelo IBM MQ são feitas usando um ID de usuário em branco. Um ID de usuário em branco não pode ter a autoridade explícita designada aos recursos do IBM MQ, mas é tratado como um membro do

grupo especial ‘nobody'. Para obter mais detalhes sobre o grupo especial nobody, consulte Informações de referência da interface de serviços instaláveis.

É possível fazer a configuração de contexto usando MQOPEN seguida por MQPUT usando a opção MQOO_ e a opção MQPMO_ indicadas nas seções a seguir. Também é possível fazer a configuração do contexto usando apenas um MQPUT1, neste caso, você só precisará selecionar a opção MQPMO_ indicada nas seções a seguir.

As seções a seguir deste tópico explicam o uso de contexto de identidade, o contexto do usuário e todo o contexto.

- “Transmitindo contexto de identidade” na página 772
- “Transmitindo contexto de usuário” na página 772
- “Transmitindo todo o contexto” na página 773
- “Configurando contexto de identidade” na página 773
- “Configurando contexto do usuário” na página 773
- “Configurando todo o contexto” na página 773

Transmitindo contexto de identidade

Em geral, os programas devem transmitir informações de contexto de identidade de mensagem para mensagem em torno de um aplicativo até que os dados atinjam seu destino final.

Os programas devem mudar as informações de contexto de origem cada vez que eles mudarem os dados. No entanto, os aplicativos que desejam mudar ou configurar qualquer informação de contexto devem ter o nível apropriado de autoridade. O gerenciador de filas verifica essa autoridade quando os aplicativos abrem as filas; eles devem ter autoridade para usar as opções de contexto apropriadas para a chamada MQOPEN.

Se seu aplicativo obtiver uma mensagem, processar os dados da mensagem e, em seguida, colocar os dados mudados em outra mensagem (possivelmente para processamento por outro aplicativo), o aplicativo deverá transmitir as informações do contexto de identidade da mensagem original para a nova mensagem. É possível permitir que o gerenciador de filas crie as informações de contexto de origem.

Para salvar as informações de contexto da mensagem original, use a opção MQOO_SAVE_ALL_CONTEXT ao abrir a fila para obter a mensagem. Isso está em adição a quaisquer outras opções que você usar com a chamada MQOPEN. Observe, no entanto, que você não pode salvar informações de contexto se você só procurar a mensagem.

Quando você cria a segunda mensagem:

- Abra a fila usando a opção MQOO_PASS_IDENTITY_CONTEXT (além da opção MQOO_OUTPUT).
- No campo *Context* da estrutura de opções de mensagem put, forneça o identificador da fila a partir da qual você salvou as informações de contexto.
- No campo *Options* da estrutura de opções de mensagem put, especifique a opção MQPMO_PASS_IDENTITY_CONTEXT.

Transmitindo contexto de usuário

Não é possível optar por transmitir apenas contexto do usuário. Para transmitir o contexto do usuário ao colocar uma mensagem, especifique MQPMO_PASS_ALL_CONTEXT. Quaisquer propriedades no contexto do usuário são transmitidas da mesma maneira que o contexto de origem.

Quando um MQPUT ou MQPUT1 ocorre e o contexto está sendo transmitido, todas as propriedades no contexto do usuário são transmitidas a partir da mensagem recuperada para a mensagem colocada. Quaisquer propriedades de contexto do usuário que o aplicativo put tenha alterado são colocadas com seus valores originais. Quaisquer propriedades de contexto do usuário que o aplicativo put excluiu são restauradas na mensagem colocada. Quaisquer propriedades de contexto do usuário que o aplicativo put incluiu na mensagem são retidas.

Transmitindo todo o contexto

Se o seu aplicativo obtiver uma mensagem e colocar os dados da mensagem (inalterada) em outra mensagem, o aplicativo deverá transmitir todas (identidade, origem e usuário) as informações de contexto da mensagem original para a nova mensagem. Um exemplo de um aplicativo que pode fazer isso é um transportador de mensagem que move mensagens de uma fila para outra.

Siga o mesmo procedimento que para transmitir contexto de identidade, exceto que você usa a opção MQOPEN MQOO_PASS_ALL_CONTEXT e a opção de mensagem colocada MQPMO_PASS_ALL_CONTEXT.

Configurando contexto de identidade

Se desejar configurar as informações de contexto de identidade para uma mensagem:

- Abra a fila usando a opção MQOO_SET_IDENTITY_CONTEXT.
- Coloque a mensagem na fila, especificando a opção MQPMO_SET_IDENTITY_CONTEXT. No descritor de mensagens, especifique quaisquer informações de contexto de identidade que você requerer.

Nota: Quando você configura alguns (mas não todos) dos campos de contexto de identidade usando as opções MQOO_SET_IDENTITY_CONTEXT e MQPMO_SET_IDENTITY_CONTEXT, é importante perceber que o gerenciador de filas não configura nenhum um dos outros campos.

Para modificar qualquer uma das opções de contexto da mensagem, deve-se ter as autorizações apropriadas para emitir a chamada. Por exemplo, para usar MQOO_SET_IDENTITY_CONTEXT ou MQPMO_SET_IDENTITY_CONTEXT, deve-se ter a permissão `+setid`.

Configurando contexto do usuário

Para configurar uma propriedade no contexto do usuário, configure o campo Contexto do descritor de propriedade de mensagem (MQPD) para MQPD_USER_CONTEXT quando você fizer a chamada MQSETMP.

Você não precisa de nenhuma autoridade especial para configurar uma propriedade no contexto do usuário. O contexto do usuário não tem as opções de contexto MQOO_SET_* ou MQPMO_SET_*.

Configurando todo o contexto

Se você desejar configurar ambas as informações de contexto de origem e de identidade para uma mensagem:

1. Abra a fila usando a opção MQOO_SET_ALL_CONTEXT.
2. Coloque a mensagem na fila, especificando a opção MQPMO_SET_ALL_CONTEXT. No descritor de mensagens, especifique quaisquer informações de contexto de origem e de identidade de que você precisa.

A autoridade apropriada é necessária para cada tipo de configuração de contexto.

Conceitos relacionados

"Contexto da mensagem" na página 48
As informações de *Contexto da Mensagem* permitem que o aplicativo recupere a mensagem para descobrir sobre o originador da mensagem.

Referências relacionadas

"Opções de MQOPEN relacionadas ao contexto da mensagem" na página 762
Se deseja conseguir associar as informações a uma mensagem ao colocá-la em uma fila, deve-se usar uma das opções de contexto da mensagem ao abrir a fila.

Colocando uma mensagem em uma fila usando a chamada MQPUT1

Use a chamada MQPUT1 quando desejar fechar a fila imediatamente após ter colocado uma única mensagem nela. Por exemplo, um aplicativo do servidor provavelmente usará a chamada MQPUT1 quando estiver enviando uma resposta para cada uma das filas diferentes.

MQPUT1 é funcionalmente equivalente a chamar MQOPEN seguida de MQPUT, seguida de MQCLOSE. A única diferença na sintaxe para as chamadas MQPUT e MQPUT1 é que, para MQPUT, você especifica uma manipulação de objetos, enquanto que, para MQPUT1, especifica uma estrutura do descritor de objeto (MQOD) conforme definido em MQOPEN (consulte "Identificando objetos (a estrutura MQOD)" na página 756). Isso acontece porque você precisa fornecer informações para a chamada MQPUT1 sobre a fila que ela precisa abrir, enquanto que a ao chamar MQPUT, a fila já deve estar aberta.

Como entrada para a chamada MQPUT1, deve-se fornecer:

- Uma manipulação de conexões.
- Uma descrição do objeto que deseja abrir. Isso no formato de uma estrutura do descritor de objeto (MQOD).
- Uma descrição da mensagem que você deseja colocar na fila. Isso no formato de uma estrutura do descritor de mensagens (MQMD).
- Informações de controle no formato de uma estrutura de opções put-message (MQPMO).
- O comprimento dos dados contidos dentro da mensagem (MQLONG).
- O endereço dos dados da mensagem.

A saída de MQPUT1 é:

- Um código de conclusão
- Um código de razão

Se a chamada for concluída com sucesso, ela também retornará a estrutura de suas opções e a estrutura de ser descritor de mensagens. A chamada modifica a estrutura de suas opções para mostrar o nome da fila e o gerenciador de filas para o qual a mensagem foi enviada. Se você solicitar que o gerenciador de filas gere um valor exclusivo para o identificador da mensagem que está colocando (especificando zero binário no campo *MsgId* da estrutura MQMD), a chamada inserirá o valor no campo *MsgId* antes de retornar essa estrutura para você.

Nota: Não é possível usar MQPUT1 com um nome de fila modelo; no entanto, quando uma fila modelo tiver sido aberta, será possível emitir um MQPUT1 para uma fila dinâmica.

Os seis parâmetros de entrada para MQPUT1 são:

Hconn
Essa é uma manipulação de conexões. Para aplicativos CICS, é possível especificar a constante MQHC_DEF_HCONN (que tem o valor zero) ou usar a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX. Para outros programas, sempre use a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX.

ObjDesc
Esta é uma estrutura do descritor de objeto (MQOD).

Nos campos *ObjectName* e *ObjectQMgrName*, dê o nome da fila no qual você deseja colocar uma mensagem e o nome do gerenciador de filas que possui essa fila.

O campo *DynamicQName* é ignorado para a chamada MQPUT1 porque ela não pode usar filas modelo.

Use o campo *AlternateUserId* se desejar denominar um identificador de usuário alternativo que deve ser usado para testar a autoridade para abrir a fila.

MsgDesc
Essa é uma estrutura de descritor de mensagens (MQMD). Como com a chamada MQPUT, use esta estrutura para definir a mensagem que está colocando na fila.

PutMsgOpts
Esta é uma estrutura de opções put-message (MQPMO). Use-a como o faria para a chamada MQPUT (consulte "Especificando opções usando a estrutura MQPMO" na página 767).

Quando o campo *Options* estiver configurado para zero, o gerenciador de filas usa o seu próprio ID do usuário ao executar testes de autoridade para acessar a fila. Além disso, o gerenciador de

filas ignora qualquer identificador de usuário alternativo fornecido no campo *AlternateUserId* da estrutura MQOD.

BufferLength
Este é o comprimento da mensagem.

Buffer
Essa é a área de buffer que contém o texto de sua mensagem.

Ao usar clusters, MQPUT1 opera como se MQOO_BIND_NOT_FIXED estivesse em vigor. Os aplicativos devem usar os campos resolvidos na estrutura MQPMO em vez da estrutura MQOD para determinar para onde a mensagem foi enviada. Consulte Configurando um cluster de gerenciador de filas para obter mais informações.

Há uma descrição da chamada MQPUT1 em MQPUT1.

Multi *Listas de distribuição*

No IBM MQ for Multiplatforms, as listas de distribuição permitem que você coloque uma mensagem em diversos destinos em uma única chamada MQPUT ou MQPUT1 . Uma chamada única MQOPEN pode abrir diversas filas e uma única chamada MQPUT pode, então, colocar uma mensagem em cada uma dessas filas. Algumas informações genéricas de estruturas MQI usadas para este processo podem ser substituídas pelas informações específicas relativas aos destinos individuais incluídos na lista de distribuição.

Atenção: As listas de distribuição não suportam o uso de filas de alias que apontam para objetos do tópico. Se uma fila de alias apontar para um objeto de tópico em uma lista de distribuição, o IBM MQ retornará MQRC_ALIAS_BASE_Q_TYPE_ERROR.

Quando uma chamada MQOPEN é emitida, informações genéricas são obtidas do Descritor de objeto (MQOD). Se você especificar MQOD_VERSION_2 no campo *Version* e um valor maior que zero no campo *RecsPresent*, o *Hobj* poderá ser definido como uma manipulação de uma lista (de uma ou mais filas) em vez daquela de uma fila. Neste caso, informações específicas são fornecidas por meio dos registros de objeto (MQORs), que fornecem detalhes de destino (ou seja, *ObjectName* e *ObjectQMgrName*).

A manipulação de objeto (*Hobj*) é transmitida para a chamada MQPUT, permitindo que você coloque em uma lista em vez de em uma única fila.

Quando uma mensagem é colocada nas filas (MQPUT), informações genéricas são obtidas a partir da estrutura de Put Message Option (MQPMO) e o Message Descriptor (MQMD). Informações específicas são concedidas sob a forma de Put Message Records (MQPMRs).

Os Response Records (MQRR) podem receber um código de conclusão e código de razão específicos para cada fila de destino.

Figura 56 na página 776 mostra como as listas de distribuição funcionam.

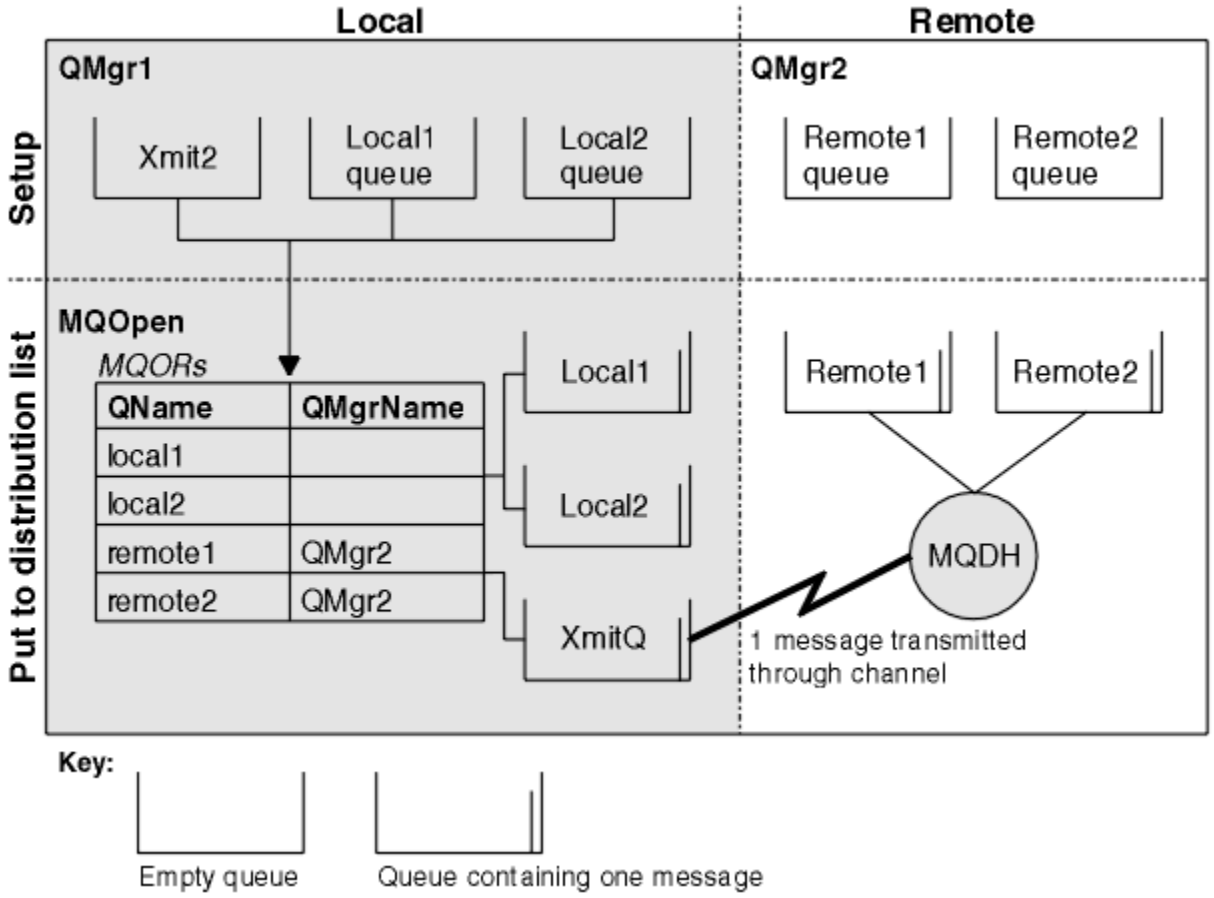


Figura 56. Como as listas de distribuição funcionam

Abrindo listas de distribuição
Use a chamada MQOPEN para abrir uma lista de distribuição e use as opções da chamada para especificar o que você deseja fazer com a lista.

Como entrada para MQOPEN, deve-se fornecer:

- Uma manipulação de conexões (consulte “Colocando mensagens em uma fila” na página 764 para obter uma descrição)
- Informações genéricas na estrutura Objeto Descriptor (MQOD)
- O nome de cada fila que você deseja abrir, usando a estrutura Object Record (MQOR)

A saída de MQOPEN é:

- Uma manipulação de objetos que representa seu acesso à lista de distribuição
- Um código de conclusão genérico
- Um código de razão genérico
- Response Records (opcionais), contendo um código de conclusão e de razão para cada destino

Usando a estrutura MQOD

Use a estrutura MQOD para identificar as filas que deseja abrir.

Para definir uma lista de distribuição, deve-se especificar MQOD_VERSION_2 no campo *Version*, um valor maior que zero no campo *RecsPresent* e MQOT_Q no campo *ObjectType* campo. Consulte MQOD para obter uma descrição de todos os campos da estrutura MQOD.

Usando a estrutura MQOR

Forneça uma estrutura MQOR para cada destino.

A estrutura contém a fila de destino e nomes do gerenciador de filas. Os campos *ObjectName* e *ObjectQMgrName* no MQOD não são usados para listas de distribuição. Deve haver um ou mais registros de objeto. Se *ObjectQMgrName* for deixado em branco, o gerenciador de filas locais será usado. Consulte ObjectName e ObjectQMgrName para obter informações adicionais sobre esses campos.

É possível especificar as filas de destino de duas maneiras:

- Usando o campo de deslocamento *ObjectRecOffset*.

  Nesse caso, o aplicativo deve declarar sua própria estrutura contendo uma estrutura MQOD, seguida pela matriz de registros MQOR (com quantos elementos de matriz forem necessários) e configurar *ObjectRecOffset* para o deslocamento do primeiro elemento na matriz a partir do início do MQOD. Assegure que esse deslocamento esteja correto.

  O uso de recursos integrados fornecidos pela linguagem de programação é recomendado, se estiverem disponíveis em todos os ambientes nos quais o aplicativo é executado. O código a seguir ilustra essa técnica para a linguagem de programação COBOL:

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  Como alternativa, use a constante MQOD_CURRENT_LENGTH, se a linguagem de programação não suportar os recursos internos necessários em todos os ambientes em questão. O código a seguir ilustra essa técnica:

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  No entanto, isso funcionará corretamente somente se a estrutura MQOD e a matriz de registros MQOR forem contíguas; se o compilador inserir bytes para ignorar entre a matriz MQOD e MQOR, eles devem ser incluídos no valor armazenado em *ObjectRecOffset*.

  Usar *ObjectRecOffset* é recomendado para linguagens de programação que não suportam o tipo de dados do ponteiro ou que implementam o tipo de dados do ponteiro sem portabilidade para diferentes ambientes (por exemplo, a linguagem de programação COBOL).

- Usando o campo do ponteiro *ObjectRecPtr*

  Nesse caso, o aplicativo pode declarar a matriz de estruturas MQOR separadamente da estrutura MQOD e configurar *ObjectRecPtr* para o endereço da matriz. O código a seguir ilustra essa técnica para a linguagem de programação C:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  Usar *ObjectRecPtr* é recomendado para linguagens de programação que suportam o tipo de dados do ponteiro de uma maneira que tenha portabilidade para os diferentes ambientes (por exemplo, a linguagem de programação C).

Independentemente da técnica escolhida, deve-se usar um de *ObjectRecOffset* e *ObjectRecPtr*; a chamada falha com o código de razão MQRC_OBJECT_RECORDS_ERROR se ambos forem zero ou ambos forem diferentes de zero.

Usando a estrutura MQRR

Essas estruturas são específicas do destino; cada Response Record contém um campo *CompCode* e *Reason* para cada fila de uma lista de distribuição. Deve-se usar essa estrutura para permitir que você distingua onde residem os problemas.

Por exemplo, se você receber um código de razão MQRC_MULTIPLE_REASONS e sua lista de distribuição contiver cinco filas de destino, não saberá a quais filas os problemas se aplicam se não usar essa estrutura. No entanto, se tiver um código de conclusão e um código de razão para cada destino, será possível localizar os erros mais facilmente.

Consulte MQRR para obter informações adicionais sobre a estrutura MQRR.

Figura 57 na página 778 mostra como é possível abrir uma lista de distribuição em C.

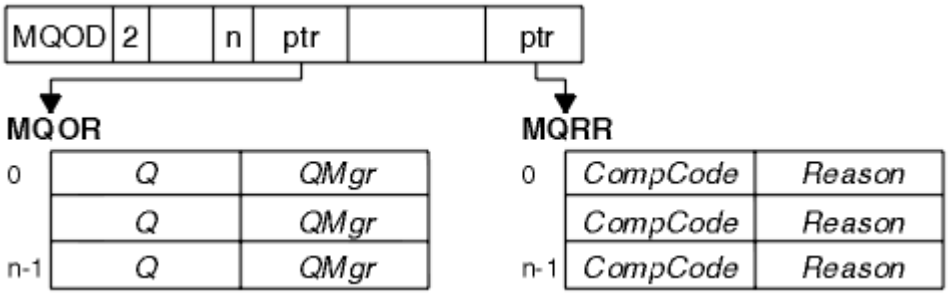


Figura 57. Abrindo uma lista de distribuição em C

Figura 58 na página 778 mostra como é possível abrir uma lista de distribuição em COBOL.

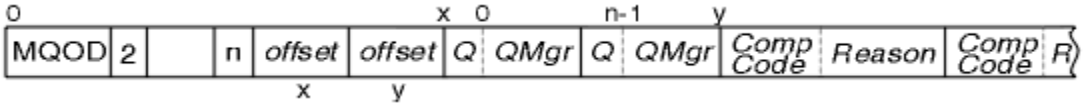


Figura 58. Abrindo uma lista de distribuição em COBOL

Usando as opções de MQOPEN

É possível especificar as opções a seguir ao abrir uma lista de distribuição:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (opcional)
- MQOO_ALTERNATE_USER_AUTHORITY (opcional)
- MQOO_*_CONTEXT (opcional)

Consulte “Abrindo e fechando objetos” na página 754 para obter uma descrição dessas opções.

Colocando mensagens em uma lista de distribuição
Para colocar mensagens em uma lista de distribuição, é possível usar MQPUT ou MQPUT1.

Como entrada, deve-se fornecer:

- Uma manipulação de conexões (consulte “Colocando mensagens em uma fila” na página 764 para obter uma descrição).
- Uma manipulação de objetos. Se uma lista de distribuição for aberta usando MQOPEN, *Hobj* permite somente colocar na lista.
- Uma estrutura do descritor de mensagens (MQMD). Consulte MQMD para obter uma descrição dessa estrutura.

- Informações de controle na forma de uma estrutura da opção put-message (MQPMO). Consulte “Especificando opções usando a estrutura MQPMO” na página 767 para obter informações sobre como preencher os campos da estrutura MQPMO.
- informações de controle na forma de Put Message Records (MQPMR).
- O comprimento dos dados contidos dentro da mensagem (MQLONG).
- Os dados da mensagem em si.

A saída é:

- Um código de conclusão
- Um código de razão
- Response Records (opcional)

Usando a estrutura MQPMR

Esta estrutura é opcional e fornece informações específicas do destino para alguns campos que você pode desejar identificar de forma diferente daqueles já identificados no MQMD.

Para obter uma descrição desses campos, consulte MQPMR.

O conteúdo de cada registro depende das informações fornecidas no campo *PutMsgRecFields* do MQPMO. Por exemplo, no programa de amostra AMQSPTL0.C (consulte “O programa de amostra Distribution List” na página 1103 para obter uma descrição) que mostra o uso de listas de distribuição, a amostra optar por fornecer valores para *MsgId* e *CorrelId* no MQPMR. Essa seção do programa de amostra é semelhante à seguinte:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

Isso sugere que *MsgId* e *CorrelId* são fornecidos para cada destino de uma lista de distribuição. Put Message Records são fornecidos como uma matriz.

Figura 59 na página 779 mostra como é possível colocar uma mensagem em uma lista de distribuição em C.

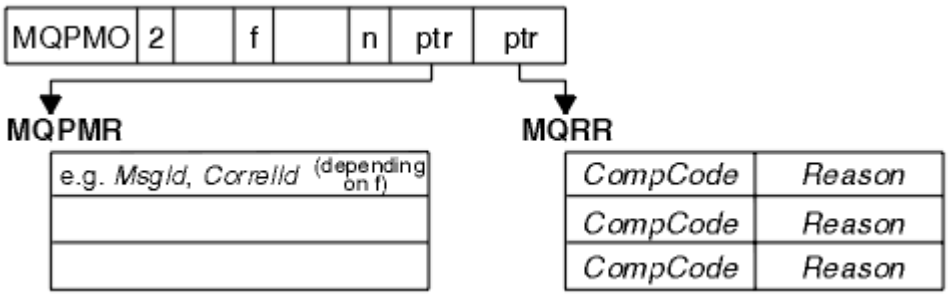


Figura 59. Colocando uma mensagem em uma lista de distribuição em C

Figura 60 na página 779 mostra como é possível colocar uma mensagem em uma lista de distribuição em COBOL.

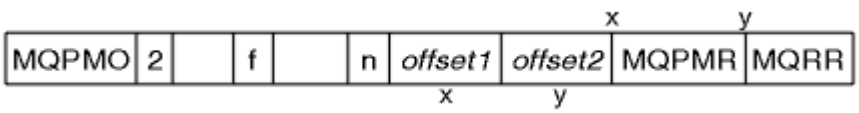


Figura 60. Colocando uma mensagem em uma lista de distribuição em COBOL

Usando MQPUT1

Se você estiver usando MQPUT1, considere os pontos a seguir:

1. Os valores dos campos *ResponseRecOffset* e *ResponseRecPtr* devem ser nulos ou zero.
2. O Response Records, se necessário, deve ser direcionado a partir do MQOD.

Alguns casos em que as chamadas put falham

Se determinados atributos de uma fila forem alterados usando a opção FORCE em um comando durante o intervalo entre a emissão de um MQOPEN e uma chamada MQGET, a chamada MQGET falhará e retornará o código de razão MQRC_OBJECT_CHANGED.

O gerenciador de filas marca a manipulação de objetos como não sendo mais válida. Isso também acontece se as mudanças forem feitas enquanto uma chamada MQPUT1 estiver sendo processada ou se as mudanças se aplicarem a qualquer fila para a qual o nome da fila é resolvido. Os atributos que afetam a manipulação dessa forma são listados na descrição da chamada MQOPEN no MQOPEN. Se sua chamada retornar o código de razão MQRC_OBJECT_CHANGED, feche a fila, reabra-a e, em seguida, tente colocar uma mensagem novamente.

Se as operações put forem inibidas para uma fila na qual você está tentando colocar mensagens (ou qualquer fila para a qual o nome da fila é resolvido), a chamada MQPUT ou MQPUT1 falhará e retornará o código de razão MQRC_PUT_INHIBITED. É possível colocar uma mensagem com êxito se você tentar a chamada posteriormente, se o design do aplicativo for tal que outros programas mudem os atributos de filas regularmente.

Além disso, se a fila na qual você está tentando colocar a mensagem estiver completa, a chamada MQPUT ou MQPUT1 falhará e retornará MQRC_Q_FULL.

Se uma fila dinâmica (temporária ou permanente) foi excluída, chamadas MQPUT que usam uma manipulação de objetos adquirida anteriormente falharão e retornarão o código de razão MQRC_Q_DELETED. Nesta situação, é uma boa prática fechar a manipulação de objetos, pois não é mais útil para você.

No caso de listas de distribuição, diversos códigos de conclusão e códigos de razão podem ocorrer em uma única solicitação. Eles não podem ser manipulados usando somente os campos de saída *CompCode* e *Reason* em MQOPEN e MQPUT.

Quando você usa listas de distribuição para colocar mensagens em múltiplos destinos, os Registros de resposta contêm o *CompCode* e o *Reason* específicos para cada destino. Se você receber um código de conclusão de MQCC_FAILED, nenhuma mensagem será colocada em nenhuma fila de destino com êxito. Se o código de conclusão for MQCC_WARNING, a mensagem será colocada com êxito em uma ou mais das filas de destino. Se você receber um código de retorno de MQRC_MULTIPLE_REASONS, os códigos de razão não serão todos iguais para cada destino. Portanto, é recomendado usar a estrutura MQRR para que seja possível determinar qual fila ou quais filas causaram um erro e as razões para cada.

Obtendo mensagens de uma fila

Use estas informações para aprender como obter mensagens de uma fila.

É possível obter mensagens de uma fila de duas maneiras:

1. É possível remover uma mensagem da fila para que outros programas não possam mais vê-la.
2. É possível copiar uma mensagem, deixando a mensagem original na fila. Isso é conhecido como *procura*. É possível remover a mensagem quando a tiver procurado.

Em ambos os casos, você usa a chamada MQGET, mas primeiro seu aplicativo deve ser conectado ao gerenciador de filas, e deve-se usar a chamada MQOPEN para abrir a fila (para entrada, procura ou ambos). Essas operações estão descritas em “Conectando-se e desconectando-se de um gerenciador de filas” na página 747 e “Abrindo e fechando objetos” na página 754.

Quando tiver aberto a fila, será possível usar a chamada MQGET repetidamente para procurar ou remover mensagens na mesma fila. Chame MQCLOSE quando tiver terminado de obter todas as mensagens desejadas da fila.

Use os links a seguir para descobrir mais sobre como obter mensagens a partir de uma fila:

- “Obtendo mensagens de uma fila usando a chamada MQGET” na página 781
- “A ordem em que as mensagens são recuperadas de uma fila” na página 786
- “Obtendo uma mensagem específica” na página 797
- “Melhorando o desempenho de mensagens não persistentes” na página 799
- z/OS “Type of index” na página 803
- “Manipulando mensagens com mais de 4 MB de comprimento” na página 803
- “Esperando mensagens” na página 809
- z/OS “Signaling” na página 810
- z/OS “Ignorando restauração” na página 811
- “Conversão de Dados do Aplicativo” na página 813
- “Procurando mensagens em uma fila” na página 814
- “Alguns casos em que a chamada MQGET falha” na página 820

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Abrindo e fechando objetos” na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Consultando e configurando atributos de objeto” na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Obtendo mensagens de uma fila usando a chamada MQGET

A chamada MQGET recebe uma mensagem de uma fila local aberta. Ela não pode obter uma mensagem de uma fila em outro sistema.

Como entrada para a chamada MQGET, deve-se fornecer:

- Uma manipulação de conexões.
- Um identificador de fila.

- Uma descrição da mensagem que você deseja obter da fila. Ela está na forma de uma estrutura do descritor de mensagens (MQMD).
- Informações de controle na forma de uma estrutura Get Message Options (MQGMO).
- O tamanho do buffer que você designou para manter a mensagem (MQLONG).
- O endereço do armazenamento no qual colocar a mensagem.

A saída de MQGET é:

- Um código de razão
- Um código de conclusão
- A mensagem na área de buffers que você especificou, se a chamada for concluída com êxito
- Sua estrutura de opções, modificada para mostrar o nome da fila da qual a mensagem foi recuperada
- Sua estrutura do descritor de mensagem, com o conteúdo dos campos modificado para descrever a mensagem que foi recuperada
- O comprimento da mensagem (MQLONG)

Existe uma descrição da chamada MQGET em MQGET.

As seções a seguir descrevem as informações que devem ser fornecidas como entrada para a chamada MQGET.

- “Especificando manipulações de conexões” na página 782
- “Descrevendo mensagens usando a estrutura MQMD e a chamada MQGET” na página 782
- “Especificando opções MQGET usando a estrutura MQGMO” na página 783
- “Especificando o tamanho da área de buffer” na página 785

Especificando manipulações de conexões

z/OS Para aplicativos CICS no z/OS, é possível especificar a constante MQHC_DEF_HCONN (que possui o valor zero) ou usar a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX. Para outros aplicativos, use sempre a manipulação de conexões retornada pela chamada MQCONN ou MQCONNX.

Use o identificador de filas (*Hobj*) que é retornado quando você chama MQOPEN.

Descrevendo mensagens usando a estrutura MQMD e a chamada MQGET

Para identificar a mensagem que você deseja obter de uma fila, use a estrutura do descritor de mensagens (MQMD).

Este é um parâmetro de entrada/saída para a chamada MQGET. Há uma introdução para as propriedades de mensagens que o MQMD descreve no “Mensagens IBM MQ” na página 18 e há uma descrição da própria estrutura no MQMD.

Se você souber qual mensagem deseja obter da fila, consulte “Obtendo uma mensagem específica” na página 797.

Se você não especificar uma mensagem determinada, MQGET recuperará a *primeira* mensagem na fila. O “A ordem em que as mensagens são recuperadas de uma fila” na página 786 descreve como a prioridade de uma mensagem, o atributo **MsgDeliverySequence** da fila e a opção MQGMO_LOGICAL_ORDER determinam a ordem das mensagens na fila.

Nota: Se desejar usar MQGET mais de uma vez (por exemplo, para percorrer as mensagens na fila), deve-se configurar os campos *MsgId* e *CorrelId* dessa estrutura como nulo após cada chamada. Isso limpa estes campos dos identificadores da mensagem que foi recuperada.

No entanto, se você desejar agrupar suas mensagens, o *GroupId* deverá ser o mesmo para mensagens no mesmo grupo, para que a chamada procure uma mensagem que possui os mesmos identificadores que a mensagem anterior para compor todo o grupo.

Especificando opções MQGET usando a estrutura MQGMO

A estrutura MQGMO é uma variável de entrada/saída para transmitir opções para a chamada MQGET. As seções a seguir ajudam a concluir alguns dos campos desta estrutura.

Há uma descrição da estrutura MQGMO em MQGMO.

StrucId
StrucId é um campo de 4 caracteres usado para identificar a estrutura como uma estrutura de opções get-message. Sempre especifique MQGMO_STRUC_ID.

Version
*Version*descreve o número da versão da estrutura. MQGMO_VERSION_1 é o padrão. Se desejar usar os campos Versão 2 ou recuperar mensagens em ordem lógica, especifique MQGMO_VERSION_2. Se desejar usar os campos Versão 3 ou recuperar mensagens em ordem lógica, especifique MQGMO_VERSION_3. MQGMO_CURRENT_VERSION configura seu aplicativo para usar o nível mais recente.

Options
No seu código, é possível selecionar as opções em qualquer ordem; cada opção é representada por um bit no campo *Options*.

O campo *Options* controla:

- Se a chamada MQGET aguarda que uma mensagem chegue na fila antes de ela ser concluída (consulte “Esperando mensagens” na página 809)
- Se a operação get está incluída em uma unidade de trabalho.
- Se uma mensagem não persistente é recuperada fora do ponto de sincronização, permitindo um sistema de mensagens rápido
- z/OS No IBM MQ for z/OS, se a mensagem recuperada é marcada como ignorando a restauração (consulte “Ignorando restauração” na página 811)
- Se a mensagem foi removida da fila ou simplesmente procurada
- Se uma mensagem deve ser selecionada usando um cursor de navegação ou outros critérios de seleção
- Se a chamada é bem-sucedida mesmo que a mensagem seja maior que o seu buffer
- z/OS No IBM MQ for z/OS, se deve permitir que a chamada seja concluída. Esta opção também configura um sinal para indicar que você deseja ser notificado quando uma mensagem chegar
- Se a chamada falhará se o gerenciador de filas estiver em um estado de quiesce
- z/OS No IBM MQ for z/OS, se a chamada falhará se a conexão estiver em um estado de quiesce
- Se a conversão de dados de mensagens do aplicativo é necessária (consulte “Conversão de Dados do Aplicativo” na página 813)
- A ordem na qual as mensagens e os segmentos são recuperados de uma fila z/OS (exceto para IBM MQ for z/OS)
- Se apenas mensagens completas, lógicas são recuperáveis z/OS (exceto para IBM MQ for z/OS)
- Se as mensagens em um grupo podem ser recuperadas apenas quando *todas* as mensagens no grupo estiverem disponíveis
- Se os segmentos em uma mensagem lógica podem ser recuperados apenas quando *todos* os segmentos na mensagem lógica estiverem disponíveis z/OS (exceto para IBM MQ for z/OS)

Se você deixar o campo *Options* configurado como o valor padrão (MQGMO_NO_WAIT), a chamada MQGET operará dessa maneira:

- Se não houver nenhuma mensagem correspondente aos seus critérios de seleção na fila, a chamada não aguardará que uma mensagem chegue, mas será concluída imediatamente. z/OS Além disso, no IBM MQ for z/OS, a chamada não configura um sinal solicitando notificação quando essa mensagem chega.
- A maneira como a chamada opera com pontos de sincronização é determinada pela plataforma:

Plataforma	Sob controle do ponto de sincronização
IBM i	No
SistemasAIX and Linux	No
z/OS z/OS z/OS	Sim
SistemasWindows	No

- z/OS No IBM MQ for z/OS, a mensagem recuperada não é marcada como ignorando restauração.
- A mensagem selecionada é removida da fila (não procurada).
- Nenhuma conversão de dados de mensagens do aplicativo é necessária.
- A chamada falhará se a mensagem for mais longa que seu buffer.

WaitInterval
O campo *WaitInterval* especifica o tempo máximo (em milissegundos) que a chamada MQGET aguarda que uma mensagem chegue na fila quando você usa a opção MQGMO_WAIT. Se nenhuma mensagem chegar dentro do tempo especificado em *WaitInterval*, a chamada será concluída e retornará um código de razão mostrando que não havia nenhuma mensagem que correspondesse aos seus critérios de seleção na fila.

z/OS No IBM MQ for z/OS, se você usar a opção MQGMO_SET_SIGNAL, o campo *WaitInterval* especificará o horário para o qual o sinal é configurado

Para obter informações adicionais sobre essas opções, consulte “Esperando mensagens” na página 809 z/OS e “Signaling” na página 810 .

z/OS ***Signal1***
Signal1 é suportado apenas no IBM MQ for z/OS.

Se você usar a opção MQGMO_SET_SIGNAL para solicitar que seu aplicativo seja notificado quando uma mensagem adequada chegar, especifique o tipo de sinal no campo *Signal1*. Em IBM MQ em todas as outras plataformas, o campo *Signal1* é reservado e seu valor não é significativo

z/OS Para obter informações adicionais, consulte “Signaling” na página 810.

Signal2
O campo *Signal2* é reservado em todas as plataformas e seu valor não é significativo.

z/OS Para obter mais informações, consulte “Signaling” na página 810.

ResolvedQName
ResolvedQName é um campo de saída no qual o gerenciador de filas retorna o nome da fila (após resolução de qualquer alias) da qual a mensagem foi recuperada.

MatchOptions
MatchOptions controla os critérios de seleção para MQGET.

GroupStatus
GroupStatus indica se a mensagem que você recuperou está em um grupo.

SegmentStatus
SegmentStatus indica se o item que você recuperou é um segmento de uma mensagem lógica.

Segmentation
Segmentation indica se a segmentação é permitida para a mensagem recuperada.

MsgToken

MsgToken identifica exclusivamente uma mensagem.

ReturnedLength
ReturnedLength é um campo de saída no qual o gerenciador de filas retorna o comprimento de dados da mensagem retornados (em bytes).

MsgHandle
O identificador para uma mensagem que deve ser preenchida com as propriedades da mensagem que está sendo recuperada da fila. O identificador foi criado anteriormente por uma chamada MQCRTMH. Todas as propriedades já associadas ao identificador são limpas antes de recuperar uma mensagem.

Especificando o tamanho da área de buffer

No parâmetro **BufferLength** da chamada MQGET, especifique o tamanho da área de buffer para conter os dados de mensagens que você recuperar. Você decide o tamanho que isso deve ter de três maneiras:

1. Você já pode saber qual comprimento de mensagens esperar desse programa. Em caso afirmativo, especifique um buffer deste tamanho.

   No entanto, é possível usar a opção MQGMO_ACCEPT_TRUNCATED_MSG na estrutura MQGMO se desejar que a chamada MQGET seja concluída mesmo que a mensagem seja muito grande para o buffer. Neste caso:

   - O buffer é preenchido com o máximo da mensagem que ele pode manter
   - A chamada retorna um código de conclusão de aviso
   - A mensagem é removida da fila (descartando o restante da mensagem) ou o cursor de navegação é avançado (se você estiver navegando na fila)
   - O comprimento real da mensagem é retornado em *DataLength*

   Sem esta opção, a chamada ainda é concluída com um aviso, mas não remove a mensagem da fila (ou avança o cursor de navegação).
2. Faça uma estimativa de um tamanho para o buffer (ou mesmo especifique um tamanho de zero bytes) e *não* use a opção MQGMO_ACCEPT_TRUNCATED_MSG. Se a chamada MQGET falhar (por exemplo, porque o buffer é muito pequeno), o comprimento da mensagem será retornado no parâmetro **DataLength** da chamada. (O buffer ainda é preenchido com o máximo da mensagem que ele pode manter, mas o processamento da chamada não é concluído.) Armazene o *MsgId* desta mensagem e, em seguida, repita a chamada MQGET, especificando uma área de buffer do tamanho correto e o *MsgId* que você anotou da primeira chamada.

   Se seu programa estiver atendendo uma fila que também está sendo atendida por outros programas, um desses outros programas poderá remover a mensagem desejada antes que seu programa possa emitir outra chamada MQGET. Seu programa poderia perder tempo procurando por uma mensagem que não existe mais. Para evitar isso, primeiro procure na fila até localizar a mensagem desejada, especificando um *BufferLength* igual a zero e usando a opção MQGMO_ACCEPT_TRUNCATED_MSG. Isso posiciona o cursor de navegação sob a mensagem que você deseja. É possível, então, recuperar a mensagem chamando MQGET novamente, especificando a opção MQGMO_MSG_UNDER_CURSOR. Se outro programa remover a mensagem entre suas chamadas de navegação e remoção, seu segundo MQGET falhará imediatamente (sem procurar na fila inteira), porque não há nenhuma mensagem sob o cursor de navegação.
3. O atributo *MaxMsgLength queue* determina o comprimento máximo de mensagens aceitas por essa fila; o atributo *MaxMsgLength queue manager* determina o comprimento máximo de mensagens aceitas para esse gerenciador de filas. Se você não souber qual comprimento de mensagem esperar,

poderá pesquisar sobre o atributo **MaxMsgLength** (usando a chamada MQINQ) e, em seguida, especificar um buffer desse tamanho.

Tente tornar o tamanho do buffer o mais próximo possível do tamanho da mensagem real para evitar o desempenho reduzido.

Para obter informações adicionais sobre o atributo **MaxMsgLength**, consulte "Aumentar o comprimento máximo da mensagem" na página 804.

A ordem em que as mensagens são recuperadas de uma fila

É possível controlar a ordem na qual se recuperam as mensagens de uma fila. Esta seção analisa as opções.

Priority

Um programa pode designar uma prioridade a uma mensagem quando ela coloca a mensagem em uma fila (consulte "Prioridades de mensagens" na página 27). As mensagens de prioridade igual são armazenadas em uma fila na ordem de chegada, não a ordem na qual elas são confirmadas.

O gerenciador de filas mantém as filas em sequência estrita FIFO (primeiro a entrar, primeiro a sair) ou em FIFO em sequência de prioridade. Isso depende da configuração do atributo **MsgDeliverySequence** da fila. Quando uma mensagem chega em uma fila, ela é inserida imediatamente após a última mensagem com a mesma prioridade.

Programas podem obter a primeira mensagem de uma fila ou eles podem obter uma mensagem específica de uma fila, ignorando a prioridade dessas mensagens. Por exemplo, um programa pode desejar processar a resposta a uma determinada mensagem que enviou anteriormente. Para obter mais informações, consulte "Obtendo uma mensagem específica" na página 797.

Se um aplicativo coloca uma sequência de mensagens em uma fila, outro aplicativo pode recuperar essas mensagens na mesma ordem em que foram colocadas, desde que:

- Todas as mensagens tenham a mesma prioridade
- As mensagens foram todas colocadas na mesma unidade de trabalho ou todas colocadas fora de uma unidade de trabalho
- A fila é local para o aplicativo de colocação

Se essas condições não forem atendidas e os aplicativos dependerem das mensagens serem recuperadas em uma determinada ordem, os aplicativos devem incluir informações de sequenciamento ou nos dados da mensagem ou estabelecer um meio de reconhecer o recebimento de uma mensagem antes que a próxima seja enviada.

z/OS No IBM MQ for z/OS, é possível usar o atributo da fila, *IndexType*, para aumentar a velocidade das operações do MQGET na fila. Para obter mais informações, consulte "Type of index" na página 803.

Ordenação lógica e física

Dentro de cada nível de prioridade, as mensagens nas filas podem ocorrer na ordem *física* ou *lógica*

Ordem física é a ordem na qual as mensagens chegam a uma fila. Ordem lógica é quando todas as mensagens e segmentos em um grupo estão em sua sequência lógica, próximos uns dos outros, na posição determinada pela posição física do primeiro item pertencente ao grupo.

Para obter uma descrição dos grupos, mensagens e segmentos, consulte "Grupos de mensagens" na página 45. Essas ordens físicas e lógicas podem diferir, pois:

- Os grupos podem chegar a um destino em momentos semelhantes a partir de diferentes aplicativos e, portanto, perder uma ordem física distinta.
- Mesmo dentro de um único grupo, as mensagens podem ficar fora de ordem por causa de roteamento ou atraso de algumas das mensagens no grupo.

Por exemplo, a ordem lógica poderá ser parecida com a da Figura Figura 61 na página 787:

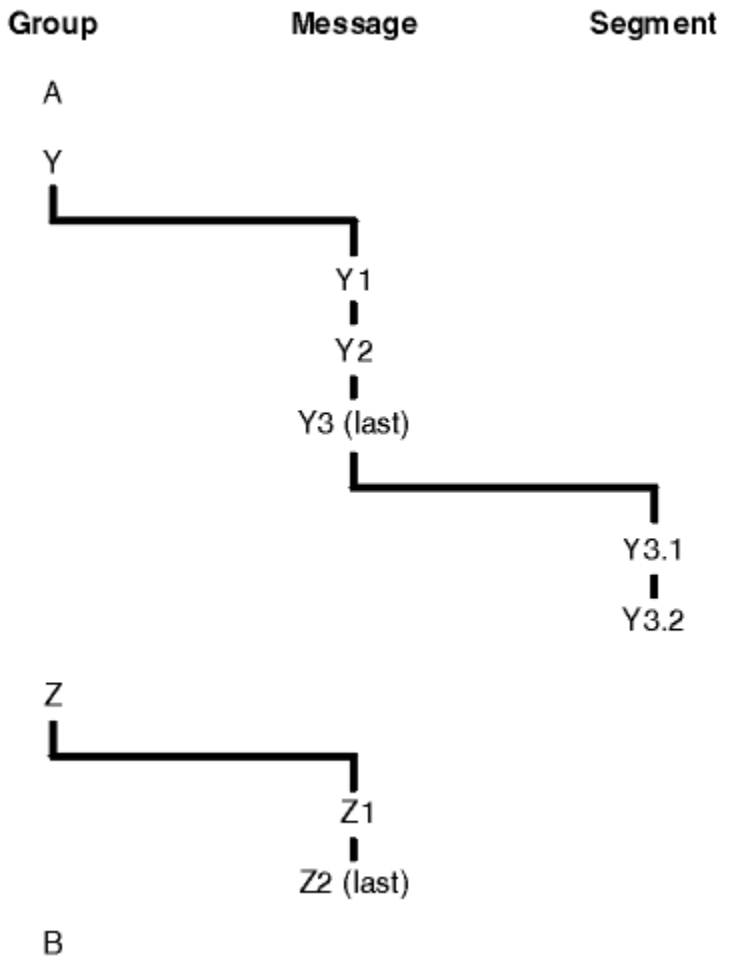


Figura 61. Ordem lógica em uma fila

Essas mensagens ocorreriam na ordem lógica a seguir em uma fila:

1. Mensagem A (não em um grupo)
2. Mensagem lógica 1 do grupo Y
3. Mensagem lógica 2 do grupo Y
4. Segmento 1 da (última) mensagem lógica 3 do grupo Y
5. (Último) segmento 2 da (última) mensagem lógica 3 do grupo Y
6. Mensagem lógica 1 do grupo Z
7. (Última) mensagem lógica 2 do grupo Z
8. Mensagem B (não em um grupo)

A ordem física, no entanto, pode ser totalmente diferente. A posição física do *primeiro* item dentro de cada grupo determina a posição lógica do grupo inteiro. Por exemplo, se os grupos Y e Z chegarem em momentos semelhantes e a mensagem 2 do grupo Z alcançar a mensagem 1 do mesmo grupo, a ordem física seria semelhante à da Figura Figura 62 na página 788:

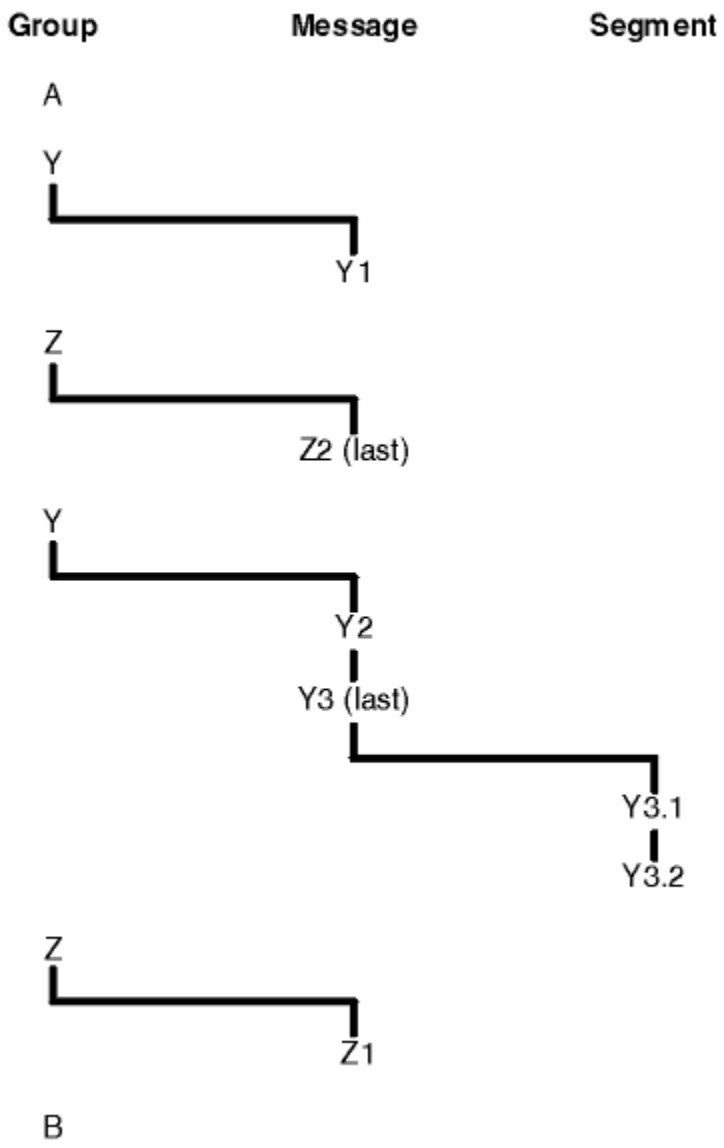


Figura 62. Ordem física em uma fila

Essas mensagens ocorrem na seguinte ordem física na fila:

1. Mensagem A (não em um grupo)
2. Mensagem lógica 1 do grupo Y
3. Mensagem lógica 2 do grupo Z
4. Mensagem lógica 2 do grupo Y
5. Segmento 1 da (última) mensagem lógica 3 do grupo Y
6. (Último) segmento 2 da (última) mensagem lógica 3 do grupo Y
7. Mensagem lógica 1 do grupo Z
8. Mensagem B (não em um grupo)

Nota: z/OS No IBM MQ for z/OS, a ordem física de mensagens na fila não será garantida se a fila for indexada por GROUPID.

Ao obter mensagens, é possível especificar MQGMO_LOGICAL_ORDER para recuperar mensagens em ordem lógica em vez da ordem física.

Se você emitir uma chamada MQGET com MQGMO_BROWSE_FIRST e MQGMO_LOGICAL_ORDER, as chamadas MQGET com MQGMO_BROWSE_NEXT subsequentes também deverão especificar MQGMO_LOGICAL_ORDER. De modo inverso, se MQGET com MQGMO_BROWSE_FIRST não especifica MQGMO_LOGICAL_ORDER, os seguintes MQGETs com MQGMO_BROWSE_NEXT também não precisarão.

As informações de grupo e segmento que o gerenciador de filas retém para chamadas MQGET que pesquisam mensagens na fila são separadas do grupo e informações de segmento que o gerenciador de filas retém para chamadas MQGET que removem da fila. Quando você especifica MQGMO_BROWSE_FIRST, o gerenciador de filas ignora as informações de grupo e segmento que devem ser pesquisadas e varre a fila se não houver nenhum grupo atual e nenhuma mensagem lógica atual.

Nota: Não use uma chamada MQGET para pesquisar *além do fim* de um grupo de mensagens (ou mensagem lógica não em um grupo) sem especificar MQGMO_LOGICAL_ORDER. Por exemplo, se a última mensagem no grupo *precede* a primeira mensagem no grupo na fila, usando MQGMO_BROWSE_NEXT para pesquisar além do final do grupo, especificando MQMO_MATCH_MSG_SEQ_NUMBER com o *MsgSeqNumber* configurado como 1 (para localizar a primeira mensagem do próximo grupo) retorna novamente a primeira mensagem no grupo já pesquisado. Isso pode acontecer imediatamente ou um número de chamadas MQGET posterior (se houver grupos de intervenção).

Evite a possibilidade de um loop infinito abrindo a fila *duas vezes* para pesquisar:

- Use o primeiro identificador para pesquisar apenas a primeira mensagem em cada grupo.
- Use o segundo identificador para pesquisar apenas as mensagens em um grupo específico.
- Use as opções MQMO_* para mover o segundo cursor de navegação para a posição do primeiro cursor de navegação, antes de pesquisar as mensagens no grupo.
- Não use a pesquisa MQGMO_BROWSE_NEXT além do fim de um grupo.

Para obter informações adicionais sobre isso, consulte MQGET, MQMD e Regras para validar opções de MQI.

Para a maioria dos aplicativos, você provavelmente escolherá ordem lógica ou física ao pesquisar. No entanto, se você desejar alternar entre esses modos, lembre-se de que quando você emite pela primeira vez uma pesquisa com MQGMO_LOGICAL_ORDER, sua posição dentro da sequência lógica é estabelecida.

Se o primeiro item dentro do grupo não estiver presente neste momento, o grupo que você está não é considerado parte da sequência lógica.

Depois que o cursor de navegação estiver em um grupo, ele poderá continuar dentro do mesmo grupo, mesmo se a primeira mensagem for removida. Inicialmente, no entanto, não é possível fazer movimentos em um grupo usando MQGMO_LOGICAL_ORDER, no qual o primeiro item não está presente.

MQPMO_LOGICAL_ORDER

A opção MQPMO instrui o gerenciador de filas sobre como o aplicativo coloca mensagens em grupos e segmentos de mensagens lógicas. Ela só pode ser especificada na chamada MQPUT. Ela não é válida na chamada MQPUT1.

Se MQPMO_LOGICAL_ORDER é especificado, ele indica que o aplicativo utiliza chamadas MQPUT sucessivas para:

1. Colocar os segmentos em cada mensagem lógica na ordem crescente de deslocamento de segmento, iniciando a partir de 0, sem lacunas.
2. Colocar todos os segmentos em uma mensagem lógica antes de colocar os segmentos na próxima mensagem lógica.
3. Colocar as mensagens lógicas em cada grupo de mensagens na ordem crescente de número de sequência da mensagem, iniciando a partir de 1, sem lacunas. IBM MQ incrementa o número de sequências da mensagem automaticamente.
4. Colocar todas as mensagens lógicas em um grupo de mensagens antes de colocar mensagens lógicas no próximo grupo de mensagens.

Como o aplicativo informou o gerenciador de filas como ele coloca mensagens em grupos e segmentos de mensagens lógicas, o aplicativo não precisa manter e atualizar as informações do

grupo e do segmento sobre cada chamada MQPUT, pois o gerenciador de filas mantém e atualiza essas informações. Especificamente, isso significa que o aplicativo não precisa configurar os campos *GroupId*, *MsgSeqNumber* e *Offset* no MQMD, porque o gerenciador de filas configura esses campos para os valores apropriados. O aplicativo deve configurar apenas o campo *MsgFlags* no MQMD, para indicar quando as mensagens pertencem a grupos ou são segmentos de mensagens lógicas e para indicar a última mensagem em um grupo ou último segmento de uma mensagem lógica.

Depois de um grupo de mensagens ou mensagem lógica ser iniciado, chamadas MQPUT subsequentes devem especificar os sinalizadores MQMF_* apropriados em *MsgFlags* no MQMD. Se o aplicativo tentar colocar uma mensagem que não está em um grupo quando há um grupo de mensagens não terminado ou colocar uma mensagem que não é um segmento quando houver uma mensagem lógica não terminada, a chamada falhará com o código de razão MQRC_INCOMPLETE_GROUP ou MQRC_INCOMPLETE_MSG, conforme apropriado. No entanto, o gerenciador de filas retém as informações sobre o grupo de mensagens atual ou de mensagens lógicas atual e o aplicativo pode finalizá-las enviando uma mensagem (possivelmente sem dados da mensagem do aplicativo) especificando MQMF_LAST_MSG_IN_GROUP ou MQMF_LAST_SEGMENT conforme apropriado, antes de emitir novamente a chamada MQPUT para colocar a mensagem que não está no grupo ou que não é um segmento.

Figura 62 na página 788 mostra as combinações de opções e sinalizações que são válidas e os valores dos campos *GroupId*, *MsgSeqNumber* e *Offset* que o gerenciador de filas usa em cada caso. Combinações de opções e sinalizadores que não são mostrados na tabela não são válidos. As colunas na tabela possuem os seguintes significados; Qualquer um significa Sim ou Não:

LOG ORD
Se a opção MQPMO_LOGICAL_ORDER é especificada na chamada.

MIG
Se a opção MQMF_MSG_IN_GROUP ou MQMF_LAST_MSG_IN_GROUP é especificada na chamada.

SEG
Se a opção MQMF_SEGMENT ou MQMF_LAST_SEGMENT é especificada na chamada.

SEG OK
Se a opção MQMF_SEGMENTATION_ALLOWED é especificada na chamada.

Cur grp
Se um grupo de mensagens atual existe antes da chamada.

Cur log msg
Se uma mensagem lógica atual existe antes da chamada.

Outras colunas
Mostram os valores que o gerenciador de filas usa. Anterior indica o valor usado para o campo na mensagem anterior para o manipulador de filas.

Tabela 116. Opções MQPUT relacionadas às mensagens em grupos e segmentos de mensagens lógicas

Opções que você especifica	Opções que você especifica	Opções que você especifica	Opções que você especifica	Grupo e status log-msg antes da chamada	Grupo e status log-msg antes da chamada	Valores que o gerenciador de filas usa	Valores que o gerenciador de filas usa	Valores que o gerenciador de filas usa
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	**GroupI**d	**MsgSeqNumber**	**Offset**
Sim	No	No	No	No	No	MQGI_NONE	1	0
Sim	No	No	Sim	No	No	Nova ID de grupo	1	0

Tabela 116. Opções MQPUT relacionadas às mensagens em grupos e segmentos de mensagens lógicas (continuação)

Opções que você especifica	Opções que você especifica	Opções que você especifica	Opções que você especifica	Grupo e status log-msg antes da chamada	Grupo e status log-msg antes da chamada	Valores que o gerenciador de filas usa	Valores que o gerenciador de filas usa	Valores que o gerenciador de filas usa
Sim	No	Sim	Qualquer um	No	No	Nova ID de grupo	1	0
Sim	No	Sim	Qualquer um	No	Sim	ID de grupo anterior	1	Deslocamento anterior + comprimento do segmento anterior
Sim	Sim	Qualquer um	Qualquer um	No	No	Nova ID de grupo	1	0
Sim	Sim	Qualquer um	Qualquer um	Sim	No	ID de grupo anterior	Número da sequência anterior + 1	0
Sim	Sim	Sim	Qualquer um	Sim	Sim	ID de grupo anterior	Número de sequência anterior	Deslocamento anterior + comprimento do segmento anterior
No	No	No	No	Qualquer um	Qualquer um	MQGI_NONE	1	0
No	No	No	Sim	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	1	0
No	No	Sim	Qualquer um	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	1	Valor no campo
No	Sim	No	Qualquer um	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	Valor no campo	0
No	Sim	Sim	Qualquer um	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	Valor no campo	Valor no campo

Nota:

- MQPMO_LOGICAL_ORDER não é válido na chamada MQPUT1.

- Para o campo *MsgId*, o gerenciador de filas gerará um novo identificador de mensagem, se MQPMO_NEW_MSG_ID ou MQMI_NONE for especificado e usará o valor no campo, caso contrário.
- Para o campo *CorrelId*, o gerenciador de filas gera um novo identificador de correlação, se MQPMO_NEW_CORREL_ID for especificado e usa o valor no campo, caso contrário.

Quando você especificar MQPMO_LOGICAL_ORDER, o gerenciador de filas exigirá que todas as mensagens em um grupo e os segmentos de uma mensagem lógica sejam colocados com o mesmo valor no campo *Persistence* no MQMD, ou seja, todos devem ser persistentes ou todos devem ser não persistentes. Se essa condição não for satisfeita, a chamada MQPUT falhará com o código de razão MQRC_INCONSISTENT_PERSISTENCE.

A opção MQPMO_LOGICAL_ORDER afeta as unidades de trabalho conforme segue:

- Se a primeira mensagem física em um grupo ou mensagem lógica for colocada em uma unidade de trabalho, todas as outras mensagens físicas no grupo ou mensagens lógicas devem ser colocadas em uma unidade de trabalho, se o mesmo manipulador de filas for usado. No entanto, elas não precisam ser colocadas na mesma unidade de trabalho, permitindo que um grupo de mensagens ou mensagem lógica que consiste em várias mensagens físicas seja dividida em duas ou mais unidades de trabalho consecutivas para o manipulador de filas.
- Se a primeira mensagem física em um grupo ou mensagem lógica não for colocada em uma unidade de trabalho, nenhuma das outras mensagens físicas no grupo ou mensagem lógica poderá ser colocada em uma unidade de trabalho se o mesmo manipulador de filas for usado.

Se estas condições não forem satisfeitas, a chamada MQPUT falhará com o código de razão MQRC_INCONSISTENT_UOW.

Quando MQPMO_LOGICAL_ORDER é especificado, o MQMD fornecido na chamada MQPUT não deve ser menor que MQMD_VERSION_2. Se essa condição não for satisfeita, a chamada falhará com o código de razão MQRC_WRONG_MD_VERSION.

Se MQPMO_LOGICAL_ORDER não for especificado, as mensagens em grupos e segmentos de mensagens lógicas poderão ser colocados em qualquer ordem e não é necessário colocar grupos de mensagens completas ou mensagens lógicas completas. É responsabilidade do aplicativo assegurar que os campos *GroupId*, *MsgSeqNumber*, *Offset*e *MsgFlags* tenham valores apropriados..

Use essa técnica para reiniciar um grupo de mensagens ou mensagem lógica no meio, após ocorrer uma falha do sistema. Quando o sistema é reiniciado, o aplicativo pode configurar os campos *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags* e *Persistence* para os valores apropriados e, em seguida, emitir a chamada MQPUT com MQPMO_SYNCPOINT ou MQPMO_NO_SYNCPOINT configurado como necessário, mas sem especificar MQPMO_LOGICAL_ORDER. Se esta chamada for bem-sucedida, o gerenciador de filas retém as informações sobre o grupo e o segmento e chamadas MQPUT subsequentes usando esse manipulador de filas poderão especificar MQPMO_LOGICAL_ORDER como normal.

As informações de grupo e segmento que o manipulador de filas retém para a chamada MQPUT são separadas das informações de grupo e segmento que ele retém para a chamada MQGET.

Para qualquer manipulador de filas, o aplicativo pode misturar chamadas MQPUT que especificam MQPMO_LOGICAL_ORDER com chamadas MQPUT que não especificam, mas observe os seguintes pontos:

- Se MQPMO_LOGICAL_ORDER não for especificado, cada chamada MQPUT bem-sucedida faz com que o gerenciador de filas configure as informações de grupo e segmento para o manipulador de filas com os valores especificados pelo aplicativo, substituindo as informações de grupo e segmento existentes retidas pelo gerenciador de filas para o manipulador de filas.
- Se MQPMO_LOGICAL_ORDER não for especificado, a chamada não falhará se houver um grupo de mensagens atuais ou mensagens lógicas. A chamada pode ter êxito com um código de conclusão MQCC_WARNING. O Tabela 117 na página 793 mostra os diferentes casos que podem surgir. Nesses casos, se o código de conclusão não for MQCC_OK, o código de razão é um dos seguintes (conforme apropriado):
  - MQRC_INCOMPLETE_GROUP

– MQRC_INCOMPLETE_MSG
– MQRC_INCONSISTENT_PERSISTENCE
– MQRC_INCONSISTENT_UOW

Nota: O gerenciador de filas não verifica as informações de grupo e segmento para a chamada MQPUT1.

Tabela 117. Resultado quando a chamada MQPUT ou MQCLOSE não é consistente com as informações de grupo e segmento

A chamada atual é	A chamada anterior era MQPUT com MQPMO_LOGICAL_ORDER	A chamada anterior era MQPUT sem MQPMO_LOGICAL_ORDER
MQPUT com MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQPUT sem MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE com um grupo ou mensagem lógica não terminada	MQCC_WARNING	MQCC_OK

Para aplicativos que colocam mensagens e segmentos em ordem lógica, especifique MQPMO_LOGICAL_ORDER, pois é a opção mais simples de usar. Esta opção livra o aplicativo da necessidade de gerenciar as informações de grupo e segmento, pois o gerenciador de filas gerencia essa informação. No entanto, os aplicativos especializados podem precisar de mais controle do que fornecido pela opção MQPMO_LOGICAL_ORDER, que pode ser obtido não especificando essa opção; se isso é feito, deve-se assegurar que os campos *GroupId*, *MsgSeqNumber*, *Offset* e *MsgFlags* no MQMD sejam configurados corretamente, antes de cada chamada MQPUT ou MQPUT1.

Por exemplo, um aplicativo que deseja redirecionar mensagens físicas que recebe, sem considerar a finalidade dessas mensagens nos grupos ou segmentos de mensagens lógicas, não deve especificar MQPMO_LOGICAL_ORDER por duas razões:

- Se as mensagens são recuperadas e colocadas em ordem, especificar MQPMO_LOGICAL_ORDER designa um novo identificador de grupo para as mensagens, que pode tornar difícil ou impossível para o originador das mensagens correlacionar quaisquer mensagens de resposta ou de relatório que resultem do grupo de mensagens.
- Em uma rede complexa com vários caminhos entre os gerenciadores de filas de envio e recebimento, as mensagens físicas podem chegar fora de ordem. Ao não especificar MQPMO_LOGICAL_ORDER e MQGMO_LOGICAL_ORDER na chamada MQGET, o aplicativo de redirecionamento pode recuperar e encaminhar cada mensagem física assim que ela chegar, sem aguardar a próxima na ordem lógica chegar.

Os aplicativos que geram mensagens de relatório para mensagens em grupos ou segmentos de mensagens lógicas também não devem especificar MQPMO_LOGICAL_ORDER ao colocar a mensagem de relatório.

MQPMO_LOGICAL_ORDER pode ser especificado com qualquer uma das outras opções MQPMO_*.

Colocando grupos ordenados de forma lógica em uma fila em cluster (MQOO_BIND_ON_GROUP)

A opção MQOO_BIND_ON_OPEN assegura que todas as mensagens deste aplicativo e, portanto, todos os grupos, sejam roteadas para uma única instância. A desvantagem é que o tráfego do aplicativo não tem a carga balanceada nas várias instâncias de uma fila de clusters. Para ativar o balanceamento de carga de trabalho enquanto mantém os grupos de mensagens intactos, deve-se configurar as opções a seguir:

- A chamada MQPUT deve especificar MQPMO_LOGICAL_ORDER
- A chamada MQOPEN deve especificar uma das duas opções a seguir:

- MQOO_BIND_ON_GROUP
- MQOO_BIND_AS_Q_DEF e a definição de fila deve especificar DEFBIND(GROUP)

O balanceamento de carga de trabalho é, então, direcionado *entre grupos* de mensagens sem precisar de um MQCLOSE e MQOPEN da fila. *Entre grupos* significa que MQMF_MSG_IN_GROUP é definido no MQMD(v2) ou MQMDE, e não há grupo parcialmente concluído em andamento. Quando um grupo está em andamento, o gerenciador de filas resolvido e o nome da fila na manipulação de objetos são reutilizados.

Se a mensagem anterior foi MO_LOGICAL_ORDER e/ou MQMF_MSG_IN_IN_GROUP foi definida mas a mensagem atual não faz parte do grupo, então a chamada PUT falhará com MQRC_INCOMPLETE_GROUP.

Se um MQPUT individual não especifica MQPMO_LOGICAL_ORDER e nenhum grupo atual está ativo, então o balanceamento de carga de trabalho é direcionado para essa mensagem (como se a chamada MQOPEN tivesse especificado MQOO_BIND_NOT_FIXED).

A não realocação ocorre para mensagens vinculadas a um destino usando MQOO_BIND_ON_GROUP. Para obter mais informações sobre a realocação, consulte “Grupos de mensagens” na página 45.

Agrupando mensagens lógicas

Há duas razões principais para usar mensagens lógicas em um grupo:

- Talvez seja necessário processar as mensagens em uma ordem específica.
- Talvez seja necessário processar cada mensagem em um grupo de forma relacionada.

Em qualquer caso, recupere o grupo inteiro com a mesma instância do aplicativo de obtenção.

Por exemplo, suponha que o grupo consista em quatro mensagens lógicas. O aplicativo de colocação será semelhante ao seguinte:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

O aplicativo de obtenção especifica a opção MQGMO_ALL_MSGS_AVAILABLE para a primeira mensagem no grupo. Isso assegura que o processamento não seja iniciado até que todas as mensagens no grupo tenham chegado. A opção MQGMO_ALL_MSGS_AVAILABLE é ignorada para mensagens subsequentes dentro do grupo.

Quando a primeira mensagem lógica do grupo é recuperada, é possível usar MQGMO_LOGICAL_ORDER para assegurar que as mensagens lógicas restantes do grupo sejam recuperadas em ordem.

Assim, o aplicativo de obtenção é semelhante ao seguinte:

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

Para obter exemplos adicionais de como agrupar mensagens, consulte “Segmentação de mensagens lógicas do aplicativo” na página 806 e “Colocando e obtendo um grupo que abrange unidades de trabalho” na página 795.

Atenção: Ao usar publicar / assinar para enviar mensagens para um tópico (ou colocar mensagens em um alias de tópico), o agrupamento de mensagens e a segmentação não são permitidos.

Como as assinaturas podem ser criadas e removidas independentemente da atividade de publicação, não é possível assegurar que um assinante receba um grupo de mensagens completo ou todos os segmentos de uma mensagem; consulte RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP

Para obter informações sobre como permitir que um aplicativo solicite que um grupo de mensagens sejam todas alocadas para a mesma instância de destino para filas de clusters, consulte DefBind.

Colocando e obtendo um grupo que abrange unidades de trabalho

No caso anterior, mensagens ou segmentos não podem começar a deixar o nó (se seu destino for remoto) ou começar a ser recuperados até que o grupo inteiro tenha sido colocado e a unidade de trabalho estiver confirmada. Isto pode não ser o desejado se levar muito tempo para colocar o grupo inteiro ou se o espaço da fila estiver limitado no nó. Para superar isso, coloque o grupo em várias unidades de trabalho.

Se o grupo for colocado em diversas unidades de trabalho, é possível que parte do grupo confirme mesmo quando o aplicativo de colocação falhar. O aplicativo deve, portanto, salvar informações de status, confirmada em cada unidade de trabalho, que ele poderá usar após um reinício para continuar um grupo incompleto. O local mais simples para registrar essas informações é em uma fila STATUS. Se um grupo completo tiver sido colocado com êxito, a fila STATUS estará vazia.

Se a segmentação estiver envolvida, a lógica será semelhante. Nesse caso, o **StatusInfo** deve incluir o *Offset*.

Aqui está um exemplo de colocação do grupo em diversas unidades de trabalho:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

Se todas as unidades de trabalho forem confirmadas, o grupo inteiro foi colocado com êxito e a fila STATUS está vazia. Se não, o grupo deve ser continuado do ponto indicado pelas informações de status. MQPMO_LOGICAL_ORDER não pode ser usado para a primeira colocação, mas pode posteriormente.

O processamento de reinicialização é semelhante a este:

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
```

```
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

No aplicativo de obtenção, você talvez queira iniciar o processamento das mensagens em um grupo antes que o grupo inteiro tenha chegado. Isso melhora tempos de resposta nas mensagens dentro do grupo e também significa que o armazenamento não é requerido para o grupo inteiro. Para realizar os benefícios, use diversas unidades de trabalho para cada grupo de mensagens. Por razões de recuperação, deve-se recuperar cada mensagem em uma unidade de trabalho.

Como com o aplicativo de colocação correspondente, isto requer que informações de status sejam registradas em algum lugar automaticamente conforme cada unidade de trabalho é confirmada. Novamente, o local mais simples para registrar estas informações estão em uma fila STATUS. Se um grupo completo foi processado com êxito, a fila STATUS estará vazia.

Nota: Para unidades de trabalho intermediárias, é possível evitar as chamadas MQGET a partir da fila STATUS especificando que cada chamada MQPUT para a fila de status é um segmento de uma mensagem (ou seja, configurando a sinalização MQMF_SEGMENT) em vez de colocar uma mensagem nova completa para cada unidade de trabalho. Na última unidade de trabalho, um segmento final é colocado na fila de status especificando MQMF_LAST_SEGMENT e, em seguida, as informações de status são limpas com um MQGET especificando MQGMO_COMPLETE_MSG.

Durante o processamento de reinicialização, em vez de usar um único MQGET para obter uma mensagem de status possível, navegue na fila de status com MQGMO_LOGICAL_ORDER até que você atinja o último segmento (ou seja, até que nenhum segmentos adicionais sejam retornados). Na primeira unidade de trabalho após a reinicialização, especifique também o deslocamento explicitamente ao colocar o segmento de status.

No exemplo a seguir, consideramos apenas as mensagens em um grupo, assumindo que buffer do aplicativo é sempre grande o suficiente para conter a mensagem inteira, seja ela segmentada ou não. Portanto, MQGMO_COMPLETE_MSG é especificado em cada MQGET. Os mesmos princípios se aplicam se a segmentação estiver envolvida (nesse caso, StatusInfo deve incluir o *Offset*).

Para simplificar, assumimos que um máximo de 4 mensagens são recuperadas dentro de uma única UOW:

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

Se todas as unidades de trabalho tiverem sido confirmadas, o grupo inteiro foi recuperado com êxito e a fila STATUS está vazia. Se não, o grupo deve ser continuado do ponto indicado pelas informações de status. MQGMO_LOGICAL_ORDER não pode ser usado para o primeiro recuperar, mas pode posteriormente.

O processamento de reinicialização é semelhante a este:

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   /* The next message on the group must be retrieved by matching
      the sequence number and group ID with those retrieved from the
      status information. */
   GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
   MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
         MQMD.GroupId      = value from Status message,
         MQMD.MsgSeqNumber = value from Status message plus 1
   msgs = 1
   /* Process this message */
   ...

   /* Now normal processing is resumed */
   /* Retrieve remaining messages in the group */
   do while ( GroupStatus == MQGS_MSG_IN_GROUP )

      /* Process up to 4 messages in the group */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                  | MQGMO_LOGICAL_ORDER
      do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
         MQGET
         msgs = msgs + 1
         /* Process this message */
         ...

      /* Have retrieved last message or 4 messages  */
      /* Update status message if not last in group */
      MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
      if ( GroupStatus == MQGS_MSG_IN_GROUP )
         StatusInfo = GroupId,MsgSeqNumber from MQMD
         MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
      MQCMIT
      msgs = 0
```

Obtendo uma mensagem específica

Existem diversas formas de obter uma mensagem específica de uma fila. São elas: selecionando em MsgId e CorrelId, selecionando em GroupId, MsgSeqNumber e Offset e selecionando em MsgToken. É possível também usar uma sequência de seleção ao abrir a fila.

Para obter uma mensagem específica de uma fila, use os campos `MsgId` e `CorrelId` da estrutura MQMD. No entanto, os aplicativos podem configurar explicitamente esses campos, portanto, os valores que você especifica podem não identificar uma mensagem exclusiva. O Tabela 118 na página 797 mostra qual mensagem é recuperada para as configurações possíveis desses campos. Esses campos serão ignorados na entrada se você especificar MQGMO_MSG_UNDER_CURSOR no parâmetro **GetMsgOpts** da chamada MQGET.

Tabela 118. Usando identificadores de mensagem e de correlação

Para recuperar...	**MsgId**	**CorrelId**
Primeira mensagem na fila	MQMI_NONE	MQCI_NONE
Primeira mensagem que corresponde a *MsgId*	Diferente de zero	MQCI_NONE
Primeira mensagem que corresponde a *CorrelId*	MQMI_NONE	Diferente de zero
Primeira mensagem que corresponde a *MsgId* e *CorrelId*	Diferente de zero	Diferente de zero

Em cada caso, *primeiro* significa a primeira mensagem que satisfaz os critérios de seleção (a menos que MQGMO_BROWSE_NEXT seja especificado, quando significa a *próxima* mensagem na sequência que satisfaz os critérios de seleção).

No retorno, a chamada MQGET configura os campos *MsgId* e *CorrelId* como os identificadores de mensagem e de correlação da mensagem retornada, se houver.

Se você configurar o campo *Version* da estrutura MQMD como 2, poderá usar os campos *GroupId*, *MsgSeqNumber* e *Offset*. O Tabela 119 na página 798 mostra qual mensagem é recuperada para as configurações possíveis desses campos.

Tabela 119. Usando o identificador de grupo

Para recuperar...	**Opções de correspondência**
Primeira mensagem na fila	MQMO_NONE
Primeira mensagem que corresponde a `MsgId`	MQMO_MATCH_MSG_ID
Primeira mensagem que corresponde a `CorrelId`	MQMO_MATCH_CORREL_ID
Primeira mensagem que corresponde a `GroupId`	MQMO_MATCH_GROUP_ID
Primeira mensagem que corresponde a `MsgSeqNumber`	MQMO_MATCH_MSG_SEQ_NUMBER
Primeira mensagem que corresponde a `MsgToken`	MQMO_MATCH_MSG_TOKEN
Primeira mensagem que corresponde a `Offset`	MQMO_MATCH_OFFSET

Notas:

1. MQMO_MATCH_XXX implica que o campo *XXX* na estrutura MQMD é configurado como o valor a ser correspondido.
2. Os sinalizadores MQMO podem ser usados em combinação. Por exemplo, MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER e MQMO_MATCH_OFFSET podem ser usados juntos para fornecer o segmento identificado pelos campos `GroupId`, `MsgSeqNumber` e `Offset`.
3. Se especificar MQGMO_LOGICAL_ORDER, a mensagem que você está tentando recuperar será afetada porque a opção depende das informações de estado controladas para o manipulador de filas. Para obter informações sobre isso, consulte “Ordenação lógica e física” na página 786 e Opções.

A chamada MQGET, geralmente, recupera a primeira mensagem de uma fila. Se você especificar uma determinada mensagem ao usar a chamada MQGET, o gerenciador de filas deverá procurar na fila até localizar essa mensagem. Isso pode afetar o desempenho do seu aplicativo.

Se você estiver usando a Versão 2 ou posterior da estrutura MQGMO e não especificar os sinalizadores MQMO_MATCH_MSG_ID ou MQMO_MATCH_CORREL_ID, não será necessário reconfigurar os campos `MsgId` ou `CorrelId` entre MQGETs.

z/OS No IBM MQ for z/OS, o atributo da fila `IndexType` pode ser usado para aumentar a velocidade de operações MQGET na fila. Para obter informações adicionais, consulte “Type of index” na página 803.

É possível obter uma mensagem específica de uma fila especificando seu MsgToken e o MatchOption MQMO_MATCH_MSG_TOKEN na estrutura MQGMO. O MsgToken é retornado pela chamada MQPUT que originalmente colocou essa mensagem na fila ou por operações MQGET anteriores e permanece constante, a menos que o gerenciador de filas seja reiniciado.

Se você estiver interessado em apenas um subconjunto de mensagens na fila, poderá especificar quais mensagens deseja processar usando uma sequência de seleção com a chamada MQOPEN ou MQSUB. MQGET, então, recupera a próxima mensagem que satisfaz essa sequência de seleção. Para obter informações adicionais sobre sequências de seleção, consulte “Seletores” na página 31.

Melhorando o desempenho de mensagens não persistentes

Quando um cliente requer uma mensagem de um servidor, ele envia uma solicitação ao servidor. Ele envia uma solicitação separada para cada uma das mensagens que consome. Para melhorar o desempenho de um cliente que consome mensagens não persistentes evitando a necessidade de enviar essas mensagens de solicitação, um cliente pode ser configurado para usar *leia mais adiante*. Leia mais adiante permite que mensagens sejam enviadas a um cliente sem que um aplicativo precise solicitá-las.

Quando a opção leia mais adiante estiver ativada, as mensagens são enviadas para um buffer de memória no cliente chamado de buffer de *leia mais adiante*. O cliente terá um buffer de leia mais adiante para cada fila que tiver aberto com leia mais adiante ativado. As mensagens no buffer de leia mais adiante não são persistidas. O cliente atualiza periodicamente o servidor com informações sobre a quantia de dados que consumiu.

Ao chamar MQOPEN com MQOO_READ_AHEAD, o cliente IBM MQ somente ativará o modo leia mais adiante se determinadas condições forem atendidas. Essas condições incluem:

- O aplicativo cliente deve ser compilado e vinculado em relação as bibliotecas encadeadas do cliente IBM MQ MQI.
- O canal do cliente deve estar utilizando o protocolo TCP/IP
- O canal deve ter a configuração de SharingConversations diferente de zero (SHARECNV) em ambas as definições de canal, do cliente e do servidor.

Usar leia mais adiante pode melhorar o desempenho ao consumir mensagens não persistentes a partir de um aplicativo cliente. Essa melhoria de desempenho está disponível para MQI e aplicativos JMS. Aplicativos clientes que usam MQGET ou consumo assíncrono se beneficiarão das melhorias de desempenho ao consumir mensagens não persistentes.

Nem todos os designs de aplicativo cliente são adequados para usar leia mais adiante, pois nem todas as opções são suportadas para uso com leia mais adiante e algumas opções precisam ser consistentes entre chamadas MQGET quando leia mais adiante está ativado. Se um cliente alterar seus critérios de seleção entre chamadas MQGET, as mensagens que estão sendo armazenadas no buffer de leia mais adiante permanecerão conectadas ao buffer de leia mais adiante do cliente.

Se uma lista não processada de mensagens conectadas com os critérios de seleção anteriores não for mais necessária, um intervalo de limpeza configurável pode ser configurado no cliente para limpar automaticamente essas mensagens do cliente. O intervalo de limpeza é uma de um grupo de opções de ajuste de leia mais adiante determinadas pelo cliente. É possível ajustar essas opções para atender seus requisitos.

Se um aplicativo cliente for reiniciado, as mensagens no buffer de leia mais adiante poderão ser perdidas. Por outro lado, uma mensagem que foi movida para um buffer de leia mais adiante poderia, então, ser excluída da fila subjacente; isso não resulta em sua remoção do buffer, portanto, uma chamada MQGET usando leia mais adiante pode retornar uma mensagem que não existe mais.

Leia mais adiante é executado somente para ligações com o cliente. O atributo é ignorado para todas as outras ligações.

Leia mais adiante não tem efeito sobre o acionamento. Nenhuma mensagem do acionador é gerada quando o cliente executa leia mais adiante de uma mensagem. Leia mais adiante não gera informações de contabilidade e estatísticas quando está ativado.

Usando leia mais adiante com sistema de mensagens de assinatura de publicação

Quando um aplicativo de assinatura especifica uma fila de destino para a qual publicações são enviadas, o valor de DEFREADA da fila especificada é usado como o valor padrão de leia mais adiante.

Quando um aplicativo de assinatura solicita que o IBM MQ gerencie o destino para o qual as publicações são enviadas, uma fila gerenciada é criada como uma fila dinâmica com base em uma fila modelo predefinida. É o valor de DEFREADA da fila modelo que é usado como o valor padrão de leia mais adiante. As filas modelo padrão SYSTEM.DURABLE.PUBLICATIONS.MODEL ou SYSTEM.NONDURABLE.PUBLICATIONS.MODEL são usadas, a menos que uma fila modelo seja definido para este ou um tópico pai.

Conceitos relacionados

"Ajustando o desempenho para mensagens não persistentes em AIX" na página 802
Se você estiver usando o AIX V5.3 ou posterior, considere configurar o parâmetro de ajuste para usar o desempenho integral para mensagens não persistentes.

Tarefas relacionadas

"Ativando e desativando leia mais adiante" na página 801
Por padrão leia mais adiante está desativado. É possível ativar leia mais adiante no nível da fila ou do aplicativo.

Referências relacionadas

"Opções de MQGET e leia mais adiante" na página 800
Nem todas as opções MQGET são suportadas quando leia mais adiante está ativada; algumas opções precisam ser consistentes entre chamadas MQGET.

Opções de MQGET e leia mais adiante

Nem todas as opções MQGET são suportadas quando leia mais adiante está ativada; algumas opções precisam ser consistentes entre chamadas MQGET.

Ao chamar MQOPEN com MQOO_READ_AHEAD, o cliente IBM MQ somente ativará o modo leia mais adiante se determinadas condições forem atendidas. Essas condições incluem:

- O aplicativo cliente deve ser compilado e vinculado em relação as bibliotecas encadeadas do cliente IBM MQ MQI.
- O canal do cliente deve estar utilizando o protocolo TCP/IP
- O canal deve ter a configuração de SharingConversations diferente de zero (SHARECNV) em ambas as definições de canal, do cliente e do servidor.

A tabela a seguir indica quais opções são suportadas para uso com leia mais adiante e se elas podem ser alteradas entre chamadas MQGET.

Tabela 120. Opções de MQGET e leia mais adiante

Valores e opções de MQGET	**Permitida quando a leitura antecipada está ativada e pode ser alterada entre chamadas MQGET** [5]	**Permitido quando a leitura antecipada está ativada e não pode ser alterada entre chamadas MQGET**[1]	**Opções MQGET que não são permitidas quando a leitura antecipada está ativada** [2]
Valores de MQGET MQMD	MsgId [3] CorrelId [3]	Codificação CodedCharSetId	
Opções de MQGET MQGMO	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED_M SG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR [4] • MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

Notas:

1. Se essas opções forem mudadas entre chamadas MQGET, um código de razão MQRC_OPTIONS_CHANGED será retornado.
2. Se estas opções forem especificadas na primeira chamada MQGET, a leitura antecipada é desativada. Se essas opções forem especificadas em uma chamada MQGET subsequente, um código de razão MQRC_OPTIONS_ERROR será retornado.

3. Se um aplicativo cliente alterar os valores de MsgId e CorrelId entre chamadas MQGET, mensagens com os valores anteriores poderão já ter sido enviadas ao cliente e permanecerão no buffer de leita mais adiante do cliente até serem consumidas (ou limpas automaticamente).
4. MQGMO_MSG_UNDER_CURSOR não é possível com a leitura antecipada. Leia mais adiante está desativada quando MQOO_BROWSE e uma das opções MQOO_INPUT_SHARED ou MQOO_INPUT_EXCLUSIVE são especificadas ao abrir a fila.
5. Quando a leitura antecipada está ativada, a primeira MQGET determina se mensagens devem ser procuradas ou obtidas de uma fila. Se o aplicativo cliente usar então MQGET com opções mudadas, como uma tentativa de procura após um get inicial ou tentar efetuar get após uma procura inicial, um código de razão MQRC_OPTIONS_CHANGED será retornado.

Se um cliente alterar seus critérios de seleção entre chamadas MQGET, as mensagens que estão sendo armazenadas no buffer de leia mais adiante que correspondem aos critérios de seleção iniciais não serão consumidas pelo aplicativo cliente e permanecerão conectadas ao buffer de leia mais adiante do cliente. Em situações em que o buffer de leia mais adiante do cliente contém muitas mensagens conectadas, os benefícios associados a leia mais adiante serão perdidos e uma solicitação separada para o servidor será necessária para cada mensagem consumida. Para determinar se leia mais adiante está sendo usada de forma eficiente, é possível usar o parâmetro do status da conexão, READA.

Leia mais adiante pode ser inibida quando solicitado por um aplicativo devido às opções incompatíveis especificadas na primeira chamada MQGET. Nessa situação, o status da conexão mostra leia mais adiante como sendo inibida.

Se, devido a essas restrições em MQGET, você decidir que o design de um aplicativo cliente não é adequado para leia mais adiante, especifique a opção MQOO_READ_AHEAD_NO de MQOPEN. Como alternativa, configure o valor de leia mais adiante padrão da fila que está sendo aberta alterado para NO ou DISABLED.

Ativando e desativando leia mais adiante
Por padrão leia mais adiante está desativado. É possível ativar leia mais adiante no nível da fila ou do aplicativo.

Sobre esta tarefa

Ao chamar MQOPEN com MQOO_READ_AHEAD, o cliente IBM MQ somente ativará o modo leia mais adiante se determinadas condições forem atendidas. Essas condições incluem:

- O aplicativo cliente deve ser compilado e vinculado em relação as bibliotecas encadeadas do cliente IBM MQ MQI.
- O canal do cliente deve estar utilizando o protocolo TCP/IP
- O canal deve ter a configuração de SharingConversations diferente de zero (SHARECNV) em ambas as definições de canal, do cliente e do servidor.

Para ativar leia mais adiante:

- Para configurar leia mais adiante no nível da fila, configure o atributo da fila, DEFREADA para YES.
- Para configurar leia mais adiante no nível do aplicativo:
  - para usar leia mais adiante sempre que possível, use a opção MQOO_READ_AHEAD na chamada de função MQOPEN. Não é possível para o aplicativo cliente usar leia mais adiante se o atributo da fila DEFREADA tiver sido configurado para DISABLED.
  - para usar leia mais adiante somente quando leia mais adiante estiver ativado em uma fila, use a opção MQOO_READ_AHEAD_AS_Q_DEF na chamada de função MQOPEN.

Se um design do aplicativo cliente não for adequado para leia mais adiante, é possível desativá-lo:

- no nível da fila, configurando o atributo da fila DEFREADA para NO, se não desejar que leia mais adiante seja usado, a menos que seja solicitado por um aplicativo cliente ou DISABLED, se não desejar que leia mais adiante seja usado independentemente de se leia mais adiante é necessário para um aplicativo cliente.

- no nível do aplicativo, usando a opção MQOO_NO_READ_AHEAD na chamada de função MQOPEN.

Duas opções MQCLOSE permitem configurar o que acontece com as mensagens que estão sendo armazenadas no buffer de leia mais adiante se a fila for fechada.

- Use MQCO_IMMEDIATE para descartar as mensagens no buffer de leia mais adiante.
- Use MQCO_QUIESCE para assegurar que as mensagens no buffer de leia mais adiante sejam consumidas pelo aplicativo antes da fila ser fechada. Quando MQCLOSE com MQCO_QUIESCE é emitido e há mensagens restantes no buffer de leia mais adiante, MQRC_READ_AHEAD_MSGS retorna com MQCC_WARNING.

Ajustando o desempenho para mensagens não persistentes em AIX
Se você estiver usando o AIX V5.3 ou posterior, considere configurar o parâmetro de ajuste para usar o desempenho integral para mensagens não persistentes.

Para configurar o parâmetro de ajuste de modo que ele entre em vigor imediatamente, emita o comando a seguir como um usuário raiz:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Para configurar o parâmetro de ajuste de modo que ele entra em vigor imediatamente e persista sobre as reinicializações, emita o comando a seguir como um usuário raiz:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Normalmente, as mensagens não persistentes são mantidas apenas na memória, mas há circunstâncias em que o AIX pode planejar que as mensagens não persistentes sejam gravadas no disco. As mensagens planejadas para serem gravadas em disco estão indisponíveis para MQGET até que a gravação do disco esteja concluída. O comando de ajuste sugerido varia esse limite; em vez de planejar que as mensagens sejam gravadas no disco quando 16 kilobytes de dados são enfileirados, a gravação em disco ocorre apenas quando o armazenamento real na máquina se torna quase cheio. Esta é uma alteração global e pode afetar outros componentes de software.

No AIX, ao usar aplicativos multiencadeados e especialmente ao executar em máquinas com vários processadores, recomenda-se fortemente configurar AIXTHREAD_SCOPE=S no ID do `.profile` mqm ou configurar AIXTHREAD_SCOPE=S no ambiente antes de iniciar o aplicativo, para melhor desempenho e planejamento mais sólido. Por exemplo:

```
export AIXTHREAD_SCOPE=S
```

Configurar AIXTHREAD_SCOPE=S significa que os encadeamentos de usuário criados com atributos padrão são colocados no escopo de contenção de todo o sistema. Se um encadeamento de usuários for criado com o escopo de contenção para todo o sistema, ele será ligado a um encadeamento kernel e planejado pelo kernel. O encadeamento kernel subjacente não é compartilhado com nenhum encadeamento de usuário.

Descritores de Arquivos

Ao executar um processo de encadeamento múltiplo, como o processo do agente, você pode alcançar o limite flexível para descritores de arquivos. Esse limite lhe dá o código de razão IBM MQ `MQRC_UNEXPECTED_ERROR (2195)` e, se houver descritores de arquivos suficientes, um arquivo IBM MQ FFST™.

Para evitar esse problema, é possível aumentar o limite de processo para o número de descritores de arquivo. Para isso, altere o atributo `nofiles` em `/etc/security/limits` para 10.000 para o ID do usuário mqm ou na sub-rotina padrão.

Limites de recursos do sistema

Defina o limite de recursos do sistema para segmento de dados e segmento de pilha como ilimitado utilizando os seguintes comandos em um prompt de comandos:

```
ulimit -d unlimited
ulimit -s unlimited
```

Type of index

The queue attribute, *IndexType*, specifies the type of index that the queue manager maintains to increase the speed of MQGET operations on the queue.

Note: Supported only on IBM MQ for z/OS.

You have five options:

Value	Description
NONE	No index is maintained. Use this when retrieving messages sequentially (see “Priority” on page 786).
GROUPID	An index of group identifiers is maintained. You must use this index type if you want logical ordering of message groups (see “Ordenação lógica e física” on page 786).
MSGID	An index of message identifiers is maintained. Use this when retrieving messages using the *MsgId* field as a selection criterion on the MQGET call (see “Obtendo uma mensagem específica” on page 797).
MSGTOKEN	An index of message tokens is maintained.
CORRELID	An index of correlation identifiers is maintained. Use this when retrieving messages using the *CorrelId* field as a selection criterion on the MQGET call (see “Obtendo uma mensagem específica” on page 797).

Note:

1. If you are indexing using the MSGID option or CORRELID option, set the relative **MsgId** or **CorrelId** parameters in the MQMD. It is not beneficial to set both.
2. Browse uses the index mechanism to find a message if a queue matches all the following conditions:
   - It has index type MSGID, CORRELID, or GROUPID
   - It is browsed with the same type of id
   - It has messages of only one priority
3. Avoid queues (indexed by *MsgId* or *CorrelId*) containing thousands of messages because this affects restart time. (This does not apply to nonpersistent messages as they are deleted at restart.)
4. MSGTOKEN is used to define queues managed by the z/OS workload manager.

For a full description of the **IndexType** attribute, see IndexType. For further information on the **IndexType** attribute, see “Design and performance considerations for z/OS applications” on page 67.

Manipulando mensagens com mais de 4 MB de comprimento

As mensagens podem ser muito grandes para o aplicativo, fila ou gerenciador de filas. Dependendo do ambiente, o IBM MQ fornece diversas maneiras para lidar com mensagens que são mais longas do que 4 MB.

É possível aumentar o atributo **MaxMsgLength** até 100 MB em todos os sistemas IBM MQ na V6 ou posterior. Configure esse valor para refletir o tamanho das mensagens usando a fila. No IBM MQ for Multiplatforms, também é possível:

1. Usar mensagens segmentadas. (As mensagens podem ser segmentadas pelo aplicativo ou pelo gerenciador de filas.)

2. Usar mensagens de referência.

Cada uma dessas abordagens é descrita no restante desta seção.

Aumentar o comprimento máximo da mensagem

O atributo **MaxMsgLength** do gerenciador de filas define o comprimento máximo de uma mensagem que pode ser manipulada por um gerenciador de filas. De forma semelhante, o atributo da fila **MaxMsgLength** é o comprimento máximo de uma mensagem que pode ser manipulada por uma fila. O comprimento máximo padrão da mensagem suportado depende do ambiente no qual você está trabalhando.

Multi No IBM MQ for Multiplatforms, é possível configurar esses dois atributos manualmente É possível configurar o valor de atributo do gerenciador de filas no intervalo de 32768 bytes até 100 MB.

Atenção: z/OS No IBM MQ for z/OS , o atributo do gerenciador de filas **MaxMsgLength** é codificado permanentemente em 100 MB

Após mudar um ou ambos os atributos **MaxMsgLength**, reinicie seus aplicativos e canais para assegurar que as mudanças entrem em vigor.

Quando estas mudanças forem feitas, o comprimento da mensagem deve ser menor ou igual aos atributos **MaxMsgLength** da fila e do gerenciador de filas. No entanto, mensagens existentes podem ser mais longas do que qualquer um desses dois atributos.

Se a mensagem for muito grande para a fila, MQRC_MSG_TOO_BIG_FOR_Q será retornado.
De forma semelhante, se a mensagem for muito grande para o gerenciador de filas, MQRC_MSG_TOO_BIG_FOR_Q_MGR será retornado.

Esse método de manipulação de mensagens grandes é fácil e conveniente. No entanto, considere os fatores a seguir antes de usá-lo:

- A uniformidade entre os gerenciadores de filas é reduzida. O tamanho máximo de dados da mensagem é determinado por *MaxMsgLength* para cada fila (incluindo filas de transmissão) em que a mensagem será colocada. Esse valor geralmente é padronizado para o *MaxMsgLength* do gerenciador de filas, principalmente, para filas de transmissão. Isso dificulta prever se uma mensagem é muito grande quando for viajar para um gerenciador de filas remotas.
- O uso de recursos do sistema é aumentado. Por exemplo, os aplicativos precisam de buffers maiores e, em algumas plataformas, poderá haver aumento no uso de armazenamento compartilhado. O armazenamento de fila deve ser afetado somente se realmente necessário para mensagens maiores.
- Lote do canal é afetado. Uma grande mensagem ainda conta como apenas uma mensagem para a contagem de lotes, mas precisa de mais tempo para transmissão, aumentando assim os tempos de resposta para outras mensagens.

Multi *Segmentação de mensagem*
Use estas informações para aprender sobre a segmentação de mensagens. Esse recurso não é suportado em IBM MQ for z/OS ou por aplicativos usando IBM MQ classes for JMS.

Aumentar o comprimento máximo da mensagem, conforme explicado no tópico "Aumentar o comprimento máximo da mensagem" na página 804 tem algumas implicações negativas. Além disso, ainda pode resultar na mensagem ser muito grande para a fila ou para o gerenciador de filas. Nesses casos, é possível segmentar uma mensagem. Para obter informações sobre segmentos, consulte "Grupos de mensagens" na página 45.

As próximas seções verificam usos comuns para segmentação de mensagens. Para put e get destrutivo, supõe-se que as chamadas MQPUT ou MQGET *sempre* operam dentro de uma unidade de trabalho. Sempre considere usar essa técnica para reduzir a possibilidade de grupos incompletos estarem presentes na rede. Single-phase commit pelo gerenciador de filas é assumido, mas outras técnicas de coordenação são igualmente válidas.

Além disso, nos aplicativos de get, supõe-se que se vários servidores estiverem processando a mesma fila, cada servidor executa um código semelhante, de forma que um servidor nunca deixe

de localizar uma mensagem ou um segmento que espera que esteja lá (porque havia especificado MQGMO_ALL_MSGS_AVAILABLE ou MQGMO_ALL_SEGMENTS_AVAILABLE anteriormente).

Atenção: Ao usar publicar / assinar para enviar mensagens para um tópico (ou colocar mensagens em um alias de tópico), o agrupamento de mensagens e a segmentação não são permitidos.

Como as assinaturas podem ser criadas e removidas independentemente da atividade de publicação, não é possível assegurar que um assinante receba um grupo de mensagens completo ou todos os segmentos de uma mensagem; consulte RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP

Efetuando put e get de uma mensagem segmentada que se estende por unidades de trabalho

É possível efetuar put e get de uma mensagem segmentada que se estende por uma unidade de trabalho de maneira semelhante a "Colocando e obtendo um grupo que abrange unidades de trabalho" na página 795.

Não é possível, no entanto, efetuar put e get de mensagens segmentadas em uma unidade de trabalho global.

Multi *Segmentação e remontagem pelo gerenciador de filas*

Esse é o cenário mais simples, no qual um aplicativo coloca uma mensagem a ser recuperada por outro. A mensagem pode ser grande: não muito grande para que o aplicativo put ou get manipule em um único buffer, mas muito grande para o gerenciador de filas ou uma fila na qual a mensagem deve ser colocada.

As únicas mudanças necessárias para esses aplicativos são para que o aplicativo put autorize o gerenciador de filas a executar a segmentação se necessário:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

e para o aplicativo get solicite ao gerenciador de filas que remonte a mensagem se ela foi segmentada:

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

Neste cenário mais simples, o aplicativo deve reconfigurar o campo GroupId para MQGI_NONE antes da chamada MQPUT, para que o gerenciador de filas possa gerar um identificador de grupo exclusivo para cada mensagem. Se isso não for feito, mensagens não relacionadas poderão ter o mesmo identificador de grupo, que pode, subsequentemente, levar ao processamento incorreto.

O buffer do aplicativo deve ser grande o suficiente para conter a mensagem remontada (a menos que você inclua a opção MQGMO_ACCEPT_TRUNCATED_MSG).

Se o atributo MAXMSGLEN de uma fila tiver de ser modificado para acomodar a segmentação de mensagens, considere:

- O segmento mínimo da mensagem suportado em uma fila local é 16 bytes.
- Para uma fila de transmissão, MAXMSGLEN também deve incluir o espaço necessário para os cabeçalhos. Considere usar um valor pelo menos 4000 bytes maior que o comprimento máximo esperado de dados do usuário em qualquer segmento de mensagem que poderia ser colocado em uma fila de transmissão.

Se a conversão de dados for necessária, o aplicativo get pode ter que fazer isso especificando MQGMO_CONVERT. Isso deve ser simples porque a saída de conversão de dados é apresentada com a mensagem completa. Não tente converter os dados em um canal do emissor se a mensagem estiver segmentada e o formato dos dados forem de tal forma que a saída de conversão de dados não puder fazer a conversão em dados incompletos.

Multi *Segmentação do aplicativo*

A segmentação do aplicativo é usada quando a segmentação do gerenciador de filas não é adequada ou quando os aplicativos requerem conversão de dados com limites de segmentação específicos.

A segmentação do aplicativo é usada por duas razões principais:

1. A segmentação do gerenciador de filas sozinha não é adequada porque a mensagem é muito grande para ser manipulada em um único buffer pelos aplicativos.
2. A conversão de dados deve ser executada pelos canais emissores e o formato é tal que o aplicativo de put deve estipular onde devem ser os limites dos segmentos para que a conversão de um segmento individual seja possível.

No entanto, se a conversão de dados não for um problema ou se o aplicativo de obtenção sempre usar MQGMO_COMPLETE_MSG, a segmentação do gerenciador de filas também poderá ser permitida especificando-se MQMF_SEGMENTATION_ALLOWED. Em nosso exemplo, o aplicativo segmenta a mensagem em quatro segmentos:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

Se não usar MQPMO_LOGICAL_ORDER, o aplicativo deverá configurar *Offset* e o comprimento de cada segmento. Neste caso, o estado lógico não é mantido automaticamente.

O aplicativo de get não pode garantir ter um buffer grande o suficiente para conter qualquer mensagem remontada. Deve, portanto, estar preparado para processar os segmentos individualmente.

Para mensagens que são segmentadas, esse aplicativo não deseja iniciar o processamento de um segmento até que todos os segmentos que constituem a mensagem lógica estejam presentes. MQGMO_ALL_SEGMENTS_AVAILABLE é, portanto, especificado para o primeiro segmento. Se você especificar MQGMO_LOGICAL_ORDER e houver uma mensagem lógica atual, MQGMO_ALL_SEGMENTS_AVAILABLE será ignorado.

Após o primeiro segmento de uma mensagem lógica ter sido recuperado, use MQGMO_LOGICAL_ORDER para assegurar que os segmentos restantes da mensagem lógica serão recuperados em ordem.

As mensagens dentro de grupos diferentes não são consideradas. Se essas mensagens ocorrerem, elas serão processadas na ordem em que o primeiro segmento de cada mensagem ocorre na fila.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi *Segmentação de mensagens lógicas do aplicativo*

As mensagens devem ser mantidas em ordem lógica em um grupo e algumas ou todas elas podem ser tão grandes que requerem segmentação de aplicativos.

Em nosso exemplo, um grupo de quatro mensagens lógicas devem ter put efetuado. Todas, exceto a terceira mensagem, são grandes e precisam de segmentação, que é executada pelo aplicativo de put:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
```

```
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

No aplicativo de get, MQGMO_ALL_MSGS_AVAILABLE é especificado no primeiro MQGET. Isso significa que nenhuma mensagem ou segmento de um grupo será recuperado até que todo o grupo esteja disponível. Quando a primeira mensagem física de um grupo tiver sido recuperada, MQGMO_LOGICAL_ORDER é usado para assegurar que os segmentos e as mensagens do grupo sejam recuperadas em ordem:

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
           (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

Nota: Se você especificar MQGMO_LOGICAL_ORDER e houver um grupo atual, MQGMO_ALL_MSGS_AVAILABLE será ignorado.

Multi *Mensagens de referência e transferências de objetos grandes*

As mensagens de referência permitem que um objeto grande seja transferido de um nó para outro sem armazenar o objeto nas filas do IBM MQ nos nós de origem ou de destino... Isso é especialmente benéfico quando os dados existem em outro formato, por exemplo, para aplicativos de e-mail.

Para ativar esse método de transferência, especifique uma saída de mensagem em ambas as extremidades de um canal. Para obter informações sobre como fazer isso, consulte “Programas de saída de mensagem do canal” na página 991.

O IBM MQ define o formato de um cabeçalho de mensagem de referência (MQRMH). Consulte MQRMH para obter uma descrição. Ele é reconhecido com um nome de formato definido e pode ser seguido pelos dados reais.

Para iniciar a transferência de um objeto grande, um aplicativo pode colocar uma mensagem que consiste em um cabeçalho de mensagem de referência sem dados após ele. Quando essa mensagem sai do nó, a saída de mensagem recupera o objeto de maneira apropriada e anexa-o à mensagem de referência. Em seguida, retorna a mensagem (agora maior do que antes) ao Agente do canal de mensagens de envio para transmissão ao MCA de recebimento.

Outra saída de mensagem é configurada no MCA de recebimento. Quando essa saída de mensagem recebe uma dessas mensagens, ela cria o objeto usando os dados do objeto que foram anexados e passa adiante a mensagem de referência *sem* isso. A mensagem de referência agora pode ser recebida por um aplicativo e esse aplicativo sabe que o objeto (ou pelo menos a parte dele representada por essa mensagem de referência) foi criado neste nó.

A quantia máxima de dados do objeto que uma saída de mensagem de envio pode anexar à mensagem de referência é limitada pelo comprimento máximo da mensagem negociado para o canal. A saída pode retornar somente uma única mensagem para o MCA para cada mensagem passada, portanto, o aplicativo de put pode colocar várias mensagens para fazer com que um objeto seja transferido. Cada mensagem deve identificar o comprimento *lógico* e o deslocamento do objeto que deve ser anexado a ele. No entanto, nos casos em que não é possível saber o tamanho total do objeto ou o tamanho máximo permitido pelo canal, projetar a saída de mensagem de envio para que o aplicativo de put apenas coloque

uma única mensagem e a saída em si coloque a próxima mensagem na fila de transmissão quando tiver anexado o quanto de dados puder à mensagem que foi passada a ela.

Antes de usar esse método de lidar com mensagens grandes, considere os pontos a seguir:

- O MCA e a saída de mensagem são executados sob um ID do usuário do IBM MQ. A saída de mensagem (e, portanto, o ID do usuário) precisa acessar o objeto para recuperá-lo na extremidade de envio ou criá-lo na extremidade de recebimento; isso pode ser viável somente em casos em que o objeto está acessível de forma global. Isso levanta um problema de segurança.
- Se a mensagem de referência com dados em massa anexados a ela precisar percorrer vários gerenciadores de filas antes de atingir seu destino, os dados em massa estarão presentes nas filas do IBM MQ nos nós intervenientes. No entanto, nenhum suporte ou saídas especiais precisam ser fornecidos nesses casos.
- Projetar sua saída de mensagem é dificultado se for permitido novo roteamento ou enfileiramento de filas não entregues. Nesses casos, as partes do objeto podem chegar fora de ordem.
- Quando uma mensagem de referência chega ao seu destino, a saída de mensagem de recebimento cria o objeto. No entanto, isso não é sincronizado com a unidade de trabalho do MCA, portanto, se o lote for restaurado, outra mensagem de referência que contém essa mesma parte do objeto chegará em um lote posterior e a saída de mensagem pode tentar recriar a mesma parte do objeto. Se o objeto for, por exemplo, uma série de atualizações do banco de dados, isso pode ser inaceitável. Nesse caso, a saída de mensagem deve manter um log de quais atualizações foram aplicadas; isso pode requerer o uso de uma fila do IBM MQ.
- Dependendo das características do tipo de objeto, as saídas de mensagem e os aplicativos podem precisar cooperar para manter contagens de uso, de forma que o objeto possa ser excluído quando não for mais necessário. Um identificador de instância também pode ser necessário; um campo é fornecido para isso no cabeçalho da mensagem de referência (consulte MQRMH).
- Se uma mensagem de referência for colocada como uma lista de distribuição, o objeto deverá ser recuperável para cada lista de distribuição ou destino individual resultante nesse nó. Pode ser necessário manter contagens de uso. Além disso, considere a possibilidade de que um nó pode ser o nó final para alguns dos destinos na lista, mas um nó intermediário para outros.
- Dados em massa não são geralmente convertidos. Isso ocorre porque a conversão ocorre *antes* que a saída de mensagem seja chamada. Por essa razão, a conversão não deve ser solicitada no canal emissor original. Se a mensagem de referência passar por um nó intermediário, os dados em massa serão convertidos quando enviados do nó intermediário, se solicitado.
- Mensagens de referência não podem ser segmentadas.

Usando as estruturas MQRMH e MQMD

Consulte MQRMH e MQMD para obter uma descrição dos campos no cabeçalho da mensagem de referência e no descritor de mensagens.

Na estrutura do MQMD, configure o campo *Format* para MQFMT_REF_MSG_HEADER. O formato MQHREF, quando solicitado em MQGET, é convertido automaticamente pelo IBM MQ juntamente com quaisquer dados em massa que seguirem.

Aqui está um exemplo do uso dos campos *DataLogicalOffset* e *DataLogicalLength* de MQRMH:

Um aplicativo de put pode colocar uma mensagem de referência com:

- Nenhum dado físico
- *DataLogicalLength* = 0 (esta mensagem representa o objeto inteiro)
- *DataLogicalOffset* = 0.

Supondo que o objeto tenha 70.000 bytes de comprimento, a saída de mensagem de envio envia os primeiros 40.000 bytes juntamente com o canal em uma mensagem de referência que contém:

- 40.000 bytes de dados físicos após o MQRMH
- *DataLogicalLength* = 40000

- *DataLogicalOffset* = 0 (a partir do início do objeto).

Em seguida, coloca uma outra mensagem na fila de transmissão contendo:

- Nenhum dado físico
- *DataLogicalLength* = 0 (até o final do objeto). Você poderia especificar um valor de 30.000 aqui.
- *DataLogicalOffset* = 40000 (a partir deste ponto).

Quando essa saída de mensagem for vista pela saída de mensagem de envio, os 30.000 bytes de dados restantes são anexados e os campos são configurados para:

- 30.000 bytes de dados físicos após o MQRMH
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (a partir deste ponto).

A sinalização MQRMHF_LAST também é configurada.

Para obter uma descrição dos programas de amostra fornecidos para o uso de mensagens de referência, consulte "Usando os programas de amostra em multiplataformas" na página 1071.

Esperando mensagens

Se desejar que um programa espere até que uma mensagem chegue em uma fila, especifique a opção MQGMO_WAIT no campo *Options* da estrutura MQGMO.

Use o campo *WaitInterval* da estrutura MQGMO para especificar o tempo máximo (em milissegundos) que deseja que uma chamada MQGET espere a chegada de uma mensagem em uma fila.

Se a mensagem não chegar dentro desse tempo, a chamada MQGET será concluída com o código de razão MQRC_NO_MSG_AVAILABLE.

É possível especificar um intervalo de espera ilimitado usando a constante MQWI_UNLIMITED no campo *WaitInterval*. No entanto, eventos fora de seu controle poderiam fazer seu programa esperar muito tempo, portanto, use essa constante com cuidado. AplicativosIMS não devem especificar um intervalo de espera ilimitado porque isso impediria a finalização do sistema IMS. (Quando o IMS é finalizado, ele requer que todas as regiões dependentes sejam finalizadas.) Em vez disso, os aplicativos IMS podem especificar um intervalo de espera finito; então, se a chamada for concluída sem recuperar uma mensagem após esse intervalo, emita outra chamada MQGET com a opção de espera.

Nota: Se mais de um programa estiver esperando na mesma fila compartilhada para *remover* uma mensagem, somente um programa será ativado por uma mensagem que chega. No entanto, se mais de um programa estiver esperando para procurar uma mensagem, todos os programas poderão ser ativados. Para obter mais informações, consulte a descrição do campo *Options* da estrutura MQGMO em MQGMO.

Se o estado da fila ou do gerenciador de filas mudar antes que o intervalo de espera expire, ocorrem as ações a seguir:

- Se o gerenciador de filas entrar no estado quiesce e você usou a opção MQGMO_FAIL_IF_QUIESCING, a espera será cancelada e a chamada MQGET será concluída com o código de razão MQRC_Q_MGR_QUIESCING. Sem essa opção, a chamada permanece em espera.
- z/OS No z/OS, se a conexão (para um aplicativo CICS ou IMS) entrar no estado quiesce e você usou a opção MQGMO_FAIL_IF_QUIESCING, a espera será cancelada e a chamada MQGET será concluída com o código de razão MQRC_CONN_QUIESCING. Sem essa opção, a chamada permanece em espera.
- Se o gerenciador de filas for forçado a parar ou for cancelado, a chamada MQGET será concluída com o código de razão MQRC_Q_MGR_STOPPING ou MQRC_CONNECTION_BROKEN.
- Se os atributos da fila (ou uma fila para a qual o nome da fila é resolvido) forem mudados de forma que solicitações get agora são inibidas, a espera será cancelada e a chamada MQGET será concluída com o código de razão MQRC_GET_INHIBITED.

- Se os atributos da fila (ou uma fila para a qual o nome da fila é resolvido) forem mudados de tal forma que a opção FORCE seja necessária, a espera será cancelada e a chamada MQGET será concluída com o código de razão MQRC_OBJECT_CHANGED.

z/OS Se deseja que seu aplicativo espere em mais de uma fila, use o recurso de sinal do IBM MQ for z/OS (consulte “Signaling” na página 810). Para obter mais informações sobre as circunstâncias nas quais essas ações ocorrem, consulte MQGMO.

Signaling

Signaling is supported only on IBM MQ for z/OS.

Signaling is an option on the MQGET call to allow the operating system to notify (or *signal*) a program when an expected message arrives on a queue. This is like the *get with wait* function described in topic “Esperando mensagens” on page 809 because it allows your program to continue with other work while waiting for the signal. However, if you use signaling, you can free the application thread and rely on the operating system to notify the program when a message arrives.

To set a signal

To set a signal, do the following in the MQGMO structure that you use on your MQGET call:

1. Set the MQGMO_SET_SIGNAL option in the *Options* field.
2. Set the maximum life of the signal in the *WaitInterval* field. This sets the length of time (in milliseconds) for which you want IBM MQ to monitor the queue. Use the MQWI_UNLIMITED value to specify an unlimited life.

   Note: IMS applications must not specify an unlimited wait interval because this would prevent the IMS system from terminating. (When IMS terminates, it requires all dependent regions to end.) Instead, IMS applications can examine the state of the ECB at regular intervals (see step 3). A program can have signals set on several queue handles at the same time:
3. Specify the address of the *Event Control Block* (ECB) in the *Signal1* field. This notifies you of the result of your signal. The ECB storage must remain available until the queue is closed.

Note: You cannot use the MQGMO_SET_SIGNAL option with the MQGMO_WAIT option.

When the message arrives

When a suitable message arrives, a completion code is returned to the ECB.

The completion code describes one of the following:

- The message that you set the signal for has arrived on the queue. The message is not reserved for the program that requested a signal, so the program must issue an MQGET call again to get the message.

  Note: Another application could get the message in the time between your receiving the signal and issuing another MQGET call.
- The wait interval you set has expired and the message you set the signal for did not arrive on the queue. IBM MQ has canceled the signal.
- The signal has been canceled. This happens, for example, if the queue manager stops, or the attribute of the queue is changed, so that MQGET calls are no longer allowed.

When a suitable message is already on the queue, the MQGET call completes in the same way as an MQGET call without signaling. Also, if an error is detected immediately, the call completes and the return codes are set.

When the call is accepted and no message is immediately available, control is returned to the program so that it can continue with other work. None of the output fields in the message descriptor are set, but the **CompCode** parameter is set to MQCC_WARNING and the **Reason** parameter is set to MQRC_SIGNAL_REQUEST_ACCEPTED.

For information about what IBM MQ can return to your application when it makes an MQGET call using signaling, see MQGET.

If the program has no other work to do while it is waiting for the ECB to be posted, it can wait for the ECB using:

- For a CICS Transaction Server for z/OS program, the EXEC CICS WAIT EXTERNAL command
- For batch and IMS programs, the z/OS WAIT macro

If the state of the queue or the queue manager changes while the signal is set (that is, the ECB has not yet been posted), the following actions occur:

- If the queue manager enters the quiescing state, and you used the MQGMO_FAIL_IF_QUIESCING option, the signal is canceled. The ECB is posted with the MQEC_Q_MGR_QUIESCING completion code. Without this option, the signal remains set.
- If the queue manager is forced to stop, or is canceled, the signal is canceled. The signal is delivered with the MQEC_WAIT_CANCELED completion code.
- If the attributes of the queue (or a queue to which the queue name resolves) are changed so that get requests are now inhibited, the signal is canceled. The signal is delivered with the MQEC_WAIT_CANCELED completion code.

Note:

1. If more than one program has set a signal on the same shared queue to remove a message, only one program is activated by a message arriving. However, if more than one program is waiting to browse a message, all the programs can be activated. The rules that the queue manager follows when deciding which applications to activate are the same as those for waiting applications: for more information, see the description of the *Options* field of the MQGMO structure in MQGMO - Get-message options.
2. If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. For more information, see the description of the *Options* field of the MQGMO structure in MQGMO - Get-message options.
3. Under some conditions, it is possible both for an MQGET call to retrieve a message and for a signal (resulting from the arrival of the same message) to be delivered. This means that when your program issues another MQGET call (because the signal was delivered), there could be no message available. Design your program to test for this situation.

For information about how to set a signal, see the description of the MQGMO_SET_SIGNAL option and the *Signal1* field in Signal1.

z/OS *Ignorando restauração*

É possível evitar que um programa de aplicativo entre em um loop *MQGET-error-backout* especificando a opção **MQGMO_MARK_SKIP_BACKOUT** na chamada MQGET.

Como parte de uma unidade de trabalho, um programa de aplicativo pode emitir uma ou mais chamadas MQGET para obter mensagens de uma fila. Se o programa de aplicativo detectar um erro, ele pode restaurar a unidade de trabalho. Isso restaura todos os recursos atualizados durante essa unidade de trabalho para o estado em que estavam antes da unidade de trabalho ser iniciada e restabelece as mensagens recuperadas pelas chamadas MQGET.

Após serem restabelecidas, essas mensagens estarão disponíveis para chamadas MQGET subsequentes emitidas pelo programa de aplicativo. Em muitos casos, isso não causa um problema para o programa de aplicativo. No entanto, em casos em que o erro que leva à restauração não pode ser contornado, ter a mensagem restabelecida na fila poderá fazer o programa de aplicativo entrar em um loop *MQGET-error-backout*.

Para evitar esse problema, especifique a opção MQGMO_MARK_SKIP_BACKOUT na chamada MQGET. Isso marca a solicitação MQGET como não estando envolvida na restauração iniciada pelo aplicativo; ou seja, ela não deve ser restaurada. O uso dessa opção significa que quando uma restauração ocorrer, atualizações em outros recursos são recuperadas conforme necessário, mas a mensagem marcada será tratada como se tivesse sido recuperada sob uma nova unidade de trabalho.

O programa de aplicativo deve emitir uma chamada do IBM MQ para confirmar a nova unidade de trabalho ou para restaurar a nova unidade de trabalho. Por exemplo, o programa pode executar a manipulação de exceção, como informar o originador que a mensagem foi descartada e confirmar a unidade de trabalho removendo a mensagem da fila. Se a nova unidade de trabalho for restaurada (por qualquer razão) a mensagem será restabelecida na fila.

Dentro de uma unidade de trabalho, pode haver apenas uma solicitação MQGET marcada para ignorar restauração; no entanto, pode haver várias outras mensagens que não estão marcadas para ignorar restauração. Após uma mensagem ser marcada para ignorar restauração, quaisquer chamadas MQGET adicionais dentro da unidade de trabalho que especifiquem MQGMO_MARK_SKIP_BACKOUT falharão com o código de razão MQRC_SECOND_MARK_NOT_ALLOWED.

Nota:

1. A mensagem marcada ignora a restauração somente se a unidade de trabalho que a contém for finalizada por uma solicitação de aplicativo para restaurá-la. Se a unidade de trabalho for restaurada por qualquer outra razão, a mensagem será restaurada na fila da mesma maneira que seria se não estivesse marcada para ignorar a restauração.
2. Ignorar a recuperação não é suportado nos procedimentos armazenados do Db2 participantes de unidades de trabalho controladas por RRS. Por exemplo, uma chamada MQGET com a opção MQGMO_MARK_SKIP_BACKOUT falhará com o código de razão MQRC_OPTION_ENVIRONMENT_ERROR.

Figura 63 na página 812 ilustra uma sequência típica de etapas que um programa de aplicativo pode conter quando uma solicitação MQGET é necessária para ignorar a restauração.

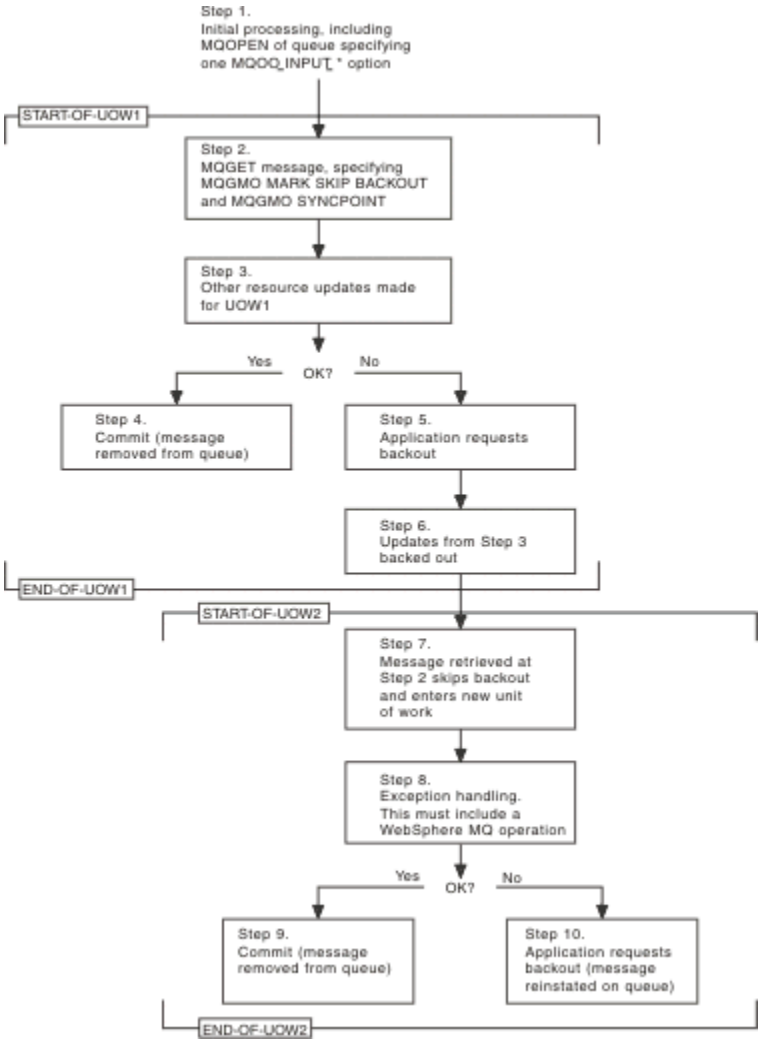


Figura 63. Ignorando a restauração usando MQGMO_MARK_SKIP_BACKOUT

As etapas em Figura 63 na página 812 são:

Etapa 1
O processamento inicial ocorre dentro da transação, incluindo uma chamada MQOPEN para abrir a fila (especificando uma das opções MQOO_INPUT_* para obter mensagens da fila na Etapa 2).

Etapa 2
MQGET é chamado, com MQGMO_SYNCPOINT e MQGMO_MARK_SKIP_BACKOUT. MQGMO_SYNCPOINT é necessário porque MQGET deve estar em uma unidade de trabalho para MQGMO_MARK_SKIP_BACKOUT ser efetivo. Em Figura 63 na página 812, essa unidade de trabalho é referida como UOW1.

Etapa 3
Outras atualizações de recursos são feitas como parte de UOW1. Elas podem incluir outras chamadas MQGET (emitidas sem MQGMO_MARK_SKIP_BACKOUT).

Etapa 4
Todas as atualizações das Etapas 2 e 3 concluídas conforme necessário. O programa de aplicativo confirma as atualizações e a UOW1 é finalizada. A mensagem recuperada na Etapa 2 é removida da fila.

Etapa 5
Algumas das atualizações das Etapas 2 e 3 não são concluídas conforme necessário. O programa de aplicativo solicita que as atualizações feitas durante essas etapas sejam restauradas.

Etapa 6
As atualizações feitas na Etapa 3 são restauradas.

Etapa 7
A solicitação MQGET feita na Etapa 2 ignora a restauração e se torna parte de uma nova unidade de trabalho, UOW2.

Etapa 8
A UOW2 executa a manipulação de exceção em resposta à restauração da UOW1. (Por exemplo, uma chamada MQPUT para outra fila, indicando que ocorreu um problema que causou a restauração da UOW1.)

Etapa 9
A Etapa 8 é concluída conforme necessário, o programa de aplicativo confirma a atividade e a UOW2 é finalizada. Como a solicitação MQGET faz parte da UOW2 (consulte a Etapa 7), essa confirmação faz com que a mensagem seja removida da fila.

Etapa 10
A Etapa 8 não é concluída conforme necessário e o programa de aplicativo restaura a UOW2. Como a solicitação get message faz parte da UOW2 (consulte a Etapa 7), ela também é restaurada e restabelecida na fila. Agora, esta está disponível para novas chamadas MQGET emitidas por este ou outro programa de aplicativo (da mesma maneira que qualquer outra mensagem na fila).

Conversão de Dados do Aplicativo

Quando necessário, MCAs convertem o descritor de mensagens e os dados de cabeçalho no conjunto de caracteres e na codificação necessários. Qualquer uma das extremidades do link (ou seja, o MCA local ou o MCA remoto) pode fazer a conversão.

Quando um aplicativo coloca mensagens em uma fila, o gerenciador de filas local inclui informações de controle nos descritores de mensagens para facilitar o controle das mensagens quando elas são processadas pelos gerenciadores de filas e MCAs. Dependendo do ambiente, os campos de dados do cabeçalho da mensagem são criados no conjunto de caracteres e na codificação do sistema local.

Ao mover mensagens entre sistemas, você, às vezes, precisa converter os dados do aplicativo no conjunto de caracteres e na codificação requeridos pelo sistema de recebimento. Isso pode ser feito a partir de programas de aplicativos no sistema de recebimento ou pelos MCAs no sistema de envio. Se a conversão de dados for suportada no sistema de recebimento, use os programas de aplicativo para converter os dados do aplicativo, em vez de depender de a conversão já ter ocorrido no sistema de envio.

Os dados do aplicativo são convertidos em um programa de aplicativo quando você especifica a opção MQGMO_CONVERT no campo *Options* da estrutura MQGMO passada para uma chamada MQGET e quando *todas* as instruções a seguir são verdadeiras:

- Os campos *CodedCharSetId* ou *Encoding* configurados na estrutura MQMD associada à mensagem na fila são diferentes dos campos *CodedCharSetId* ou *Encoding* configurados na estrutura MQMD especificada na chamada MQGET.
- O campo *Format* na estrutura MQMD associada à mensagem não for MQFMT_NONE.
- O *BufferLength* especificado na chamada MQGET não for zero.
- O comprimento dos dados da mensagem não for zero.
- O gerenciador de filas suporta a conversão entre os campos *CodedCharSetId* e *Encoding* especificados nas estruturas MQMD associadas à mensagem e à chamada MQGET. Consulte CodedCharSetId e Codificação para obter detalhes sobre os identificadores do conjunto de caracteres codificados e as codificações de máquina suportados.
- O gerenciador de filas suporta a conversão do formato da mensagem. Se o campo *Format* da estrutura MQMD associada à mensagem for um dos formatos integrados, o gerenciador de filas poderá converter a mensagem. Se o *Format* não for um dos formatos integrados, será necessário gravar uma saída de conversão de dados para converter a mensagem.

Se o MCA de envio for para converter os dados, especifique a palavra-chave CONVERT(YES) na definição de cada emissor ou canal do servidor para o qual a conversão é necessária. Se a conversão de dados falhar, a mensagem será enviada para o DLQ no gerenciador de filas de envio e o campo *Feedback* da estrutura MQDLH indicará a razão. Se a mensagem não puder ser colocada no DLQ, o canal será fechado e a mensagem não convertida permanecerá na fila de transmissão. A conversão de dados nos aplicativos em vez de nos MCAs de envio evita esta situação.

Como regra, os dados na mensagem que são descritos como *dados de caracteres* pelo formato integrado ou saída de conversão de dados são convertidos do conjunto de caracteres codificados usado pela mensagem para esse solicitado e os campos *numéricos* são convertidos para a codificação solicitada.

Para obter detalhes adicionais das convenções de processamento de conversão usadas ao converter os formatos integrados e para obter informações sobre como gravar suas próprias saídas de conversão de dados, consulte “Escrevendo saídas de conversão de dados” na página 995. Consulte também Idiomas nacionais e Codificações da máquina para obter informações sobre as tabelas de suporte ao idioma e sobre as codificações de máquina suportadas.

Conversão de caracteres de nova linha EBCDIC

Se precisar assegurar que os dados que você envia de uma plataforma EBCDIC para um ASCII são idênticos aos dados que você recebe de volta, deve-se controlar a conversão de caracteres de nova linha EBCDIC.

É possível fazer isso usando um comutador dependente da plataforma que força o IBM MQ a usar as tabelas de conversão não modificadas, mas deve-se estar ciente do comportamento inconsistente que pode resultar.

O problema surge porque o caractere de nova linha EBCDIC não é convertido de forma consistente em plataformas ou tabelas de conversão. Como resultado, se os dados forem exibidos em uma plataforma ASCII, a formatação pode estar incorreta. Isso dificultaria, por exemplo, administrar um sistema IBM i remotamente a partir de uma plataforma ASCII usando RUNMQSC.

Consulte Conversão de dados para obter informações adicionais sobre como converter dados de formato EBCDIC no formato ASCII.

Procurando mensagens em uma fila

Use estas informações para descobrir sobre como procurar mensagens em uma fila usando a chamada MQGET.

Para usar a chamada MQGET para procurar as mensagens em uma fila:

1. Chame o MQOPEN para abrir a fila para navegar, especificando a opção MQOO_BROWSE.
2. Para navegar para a primeira mensagem na fila, chame MQGET com a opção MQGMO_BROWSE_FIRST. Para localizar a mensagem que você deseja, chame MQGET repetidamente com a opção MQGMO_BROWSE_NEXT para percorrer várias mensagens.

   Deve-se configurar os campos *MsgId* e *CorrelId* da estrutura MQMD como nulo após cada chamada MQGET para ver todas as mensagens.
3. Chame MQCLOSE para fechar a fila.

O cursor de navegação
Ao abrir (MQOPEN) uma fila para procura, a chamada estabelece um cursor de procura para ser usado com chamadas MQGET que usem uma das opções de procura. É possível considerar o cursor de procura como um ponteiro lógico posicionado antes da primeira mensagem na fila.

É possível haver mais de um cursor de procura ativo (a partir de um único programa) emitindo várias solicitações MQOPEN para a mesma fila.

Ao chamar MQGET para procurar, use uma das opções a seguir em sua estrutura MQGMO:

MQGMO_BROWSE_FIRST
Obtém uma cópia da primeira mensagem que satisfaz as condições especificadas em sua estrutura MQMD.

MQGMO_BROWSE_NEXT
Obtém uma cópia da próxima mensagem que satisfaz as condições especificadas em sua estrutura MQMD.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Obtém uma cópia da mensagem atualmente apontada pelo cursor, ou seja, aquela que foi recuperada por último usando a opção MQGMO_BROWSE_FIRST ou MQGMO_BROWSE_NEXT.

Em todos os casos, a mensagem permanece na fila.

Ao abrir uma fila, o cursor de procura é posicionado logicamente antes da primeira mensagem na fila. Isso significa que se você fizer sua chamada MQGET imediatamente após a sua chamada MQOPEN, será possível usar a opção MQGMO_BROWSE_NEXT para procurar a primeira mensagem; não é necessário usar a opção MQGMO_BROWSE_FIRST.

A ordem na qual as mensagens são copiadas da fila é determinada pelo atributo **MsgDeliverySequence** da fila. (Para obter mais informações, consulte "A ordem em que as mensagens são recuperadas de uma fila" na página 786.)

- "Filas na sequência FIFO (primeira a entrar, primeira a sair)" na página 815
- "Filas em sequência de prioridade" na página 816
- "Mensagens não confirmadas" na página 816
- "Mudança na sequência da fila" na página 816
- z/OS "Usando o índice da fila." na página 816

Filas na sequência FIFO (primeira a entrar, primeira a sair)

A primeira mensagem em uma fila nessa sequência é a mensagem que está na fila há mais tempo.

Use MQGMO_BROWSE_NEXT para ler as mensagens sequencialmente na fila. Você verá quaisquer mensagens colocadas na fila enquanto estiver procurando, pois uma fila nessa sequência tem mensagens colocadas no final. Quando o cursor reconhece que alcançou o fim da fila, o cursor de procura permanece onde está e retorna com MQRC_NO_MSG_AVAILABLE. É possível então deixá-lo lá esperando mensagens adicionais ou reconfigurá-lo para o início da fila com uma chamada MQGMO_BROWSE_FIRST.

Filas em sequência de prioridade

A primeira mensagem em uma fila nessa sequência é a mensagem que está na fila há mais tempo e que tem a prioridade mais alta no momento em que a chamada MQOPEN é emitida.

Use MQGMO_BROWSE_NEXT para ler as mensagens na fila.

O cursor de procura aponta para a próxima mensagem, trabalhando a partir da prioridade da primeira mensagem para terminar com a mensagem com a prioridade mais baixa. Ele procura quaisquer mensagens colocadas na fila durante esse tempo, contanto que elas tenham prioridade igual a ou menor que a mensagem identificada pelo cursor de procura atual.

Quaisquer mensagens colocadas na fila com prioridade mais alta poderão ser procuradas somente da seguinte forma:

- Abrindo a fila para procurar novamente, quando um novo cursor de procura é estabelecido
- Usando a opção MQGMO_BROWSE_FIRST

Mensagens não confirmadas

Uma mensagem não confirmada nunca está visível para uma procura; o cursor de procura a ignora.

As mensagens dentro de uma unidade de trabalho não podem ser procuradas até a unidade de trabalho ser confirmada. As mensagens não mudam sua posição na fila quando confirmadas, portanto, mensagens não confirmadas ignoradas não serão vistas, mesmo quando *estiverem* confirmadas, a menos que você use a opção MQGMO_BROWSE_FIRST e trabalha ao longo da fila novamente.

Mudança na sequência da fila

Se a sequência de entrega de mensagem for mudada de prioridade para FIFO enquanto houver mensagens na fila, a ordem das mensagens que já estão na fila não será mudada. As mensagens incluídas na fila posteriormente, usam a prioridade padrão da fila.

Usando o índice da fila.

z/OS

No IBM MQ for z/OS, ao procurar uma fila indexada que contém apenas mensagens de uma única prioridade (persistente ou não persistente ou ambos), o gerenciador de filas usa o índice para procurar quando determinadas formas de procura são usadas.

Qualquer uma das formas de procura a seguir é usada quando uma fila indexada contém somente mensagens de prioridade única:

1. Se a fila estiver indexada por MSGID, as solicitações de procura que passam um MSGID na estrutura MQMD são processadas usando o índice para localizar a mensagem de destino.
2. Se a fila for indexada por CORRELID, solicitações de procura que passam um CORRELID na estrutura MQMD são processadas usando o índice para localizar a mensagem de destino.
3. Se a fila for indexada por GROUPID, solicitações de procura que passam um GROUPID na estrutura MQMD são processadas usando o índice para localizar a mensagem de destino.

Se a solicitação de procura não passar um MSGID, CORRELID ou GROUPID na estrutura MQMD, a fila será indexada e uma mensagem será retornada, a entrada de índice para a mensagem deve ser localizada e as informações da mesma usadas para atualizar o cursor de procura. Se você usar uma ampla seleção de valores de índice, isso não inclui processamento extra significativo à solicitação de procura.

Procurando mensagens quando o comprimento da mensagem é desconhecido

Para procurar uma mensagem quando não souber o tamanho da mensagem e se não desejar usar os campos *MsgId*, *CorrelId* ou *GroupId* para localizar a mensagem, será possível usar a opção MQGMO_BROWSE_MSG_UNDER_CURSOR:

1. Emita uma chamada MQGET com:

- A opção MQGMO_BROWSE_FIRST ou MQGMO_BROWSE_NEXT
- A opção MQGMO_ACCEPT_TRUNCATED_MSG
- Buffer de comprimento zero

Nota: Se outro programa provavelmente obtiver a mesma mensagem, considere usar a opção MQGMO_LOCK também. MQRC_TRUNCATED_MSG_ACCEPTED deve ser retornado.

2. Use *DataLength* retornado para alocar o armazenamento necessário.
3. Emita uma chamada MQGET com a MQGMO_BROWSE_MSG_UNDER_CURSOR.

A mensagem apontada é a última que foi recuperada; o cursor não terá se movido. É possível optar por bloquear a mensagem usando a opção MQGMO_LOCK ou desbloquear uma mensagem bloqueada usando a opção MQGMO_UNLOCK.

A chamada falha se nenhuma MQGET com as opções MQGMO_BROWSE_FIRST ou MQGMO_BROWSE_NEXT tiver sido emitida com sucesso desde a abertura da fila.

Removendo uma mensagem que você procurou
É possível remover da fila uma mensagem que já procurou desde que tenha aberto a fila para remover mensagens, assim como para procura. (Deve-se especificar uma das opções MQOO_INPUT_*, assim como a opção MQOO_BROWSE, em sua chamada MQOPEN.)

Para remover a mensagem, ligue para o MQGET novamente, mas no campo *Options* da estrutura MQGMO, especifique MQGMO_MSG_UNDER_CURSOR.Nesse caso, a chamada de MQGET ignora os campos *MsgId* *CorrelId*e *GroupId* da estrutura MQMD.

No tempo entre suas etapas de procura e remoção, outro programa pode ter removido mensagens da fila, inclusive a mensagem sob seu cursor de procura. Nesse caso, sua chamada MQGET retorna um código de razão para informar que a mensagem não está disponível.

Procurando mensagens em ordem lógica

"Ordenação lógica e física" na página 786 explica a diferença entre a ordem lógica e física de mensagens em uma fila. Essa distinção é importante principalmente ao procurar em uma fila, porque, em geral, as mensagens não estão sendo excluídas e operações de procura não começam necessariamente no início da fila.

Se um aplicativo procurar nas diversas mensagens de um grupo (usando ordem lógica), é importante que a ordem lógica deve ser seguida para chegar ao início do próximo grupo, porque a última mensagem de um grupo pode ocorrer fisicamente *após* a primeira mensagem do próximo grupo. A opção MQGMO_LOGICAL_ORDER assegura que a ordem lógica seja seguida ao fazer a varredura de uma fila.

Use MQGMO_ALL_MSGS_AVAILABLE (ou MQGMO_ALL_SEGMENTS_AVAILABLE) com cuidado para operações de procura. Considere o caso de mensagens lógicas com MQGMO_ALL_MSGS_AVAILABLE. O efeito disso é que uma mensagem lógica estar disponível somente se todas as mensagens restantes no grupo também estiverem presentes. Se não estiverem, a mensagem será saltada. Isso pode significar que quando as mensagens ausentes chegarem subsequentemente, elas não serão observadas por uma operação browse-next.

Por exemplo, se as mensagens lógicas a seguir estiverem presentes,

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

e uma função de procura for emitido com MQGMO_ALL_MSGS_AVAILABLE, a primeira mensagem lógica do grupo 456 será retornada primeiro, deixando o cursor de procura nessa mensagem lógica. Se a segunda (última) mensagem do grupo 123 chegar agora:

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
```

```
   Logical message 1 (not last) of group 456  <=== browse cursor
   Logical message 2 (last)     of group 456
```

e a mesma função browse-next for emitida, não será percebido que o grupo 123 agora está completo, pois a primeira mensagem desse grupo está *antes* do cursor de procura.

Em alguns casos (por exemplo, se mensagens forem recuperadas destrutivamente quando o grupo estiver presente em sua totalidade), é possível usar MQGMO_ALL_MSGS_AVAILABLE juntamente com MQGMO_BROWSE_FIRST. Caso contrário, deve-se repetir a varredura de procura para observar as novas mensagens que chegaram que passaram despercebidas; simplesmente emitindo MQGMO_WAIT junto com MQGMO_BROWSE_NEXT e MQGMO_ALL_MSGS_AVAILABLE não as leva em consideração. (Isso também acontece com mensagens de prioridade mais alta que podem chegar após a varredura de mensagens ser concluída.)

As próximas seções verificam exemplos de procura que lidam com mensagens não segmentadas; as mensagens segmentadas seguem princípios semelhantes.

Procurando mensagens em grupos
Neste exemplo, o aplicativo procura em cada mensagem na fila, em ordem lógica.

Mensagens na fila podem estar agrupadas. Para mensagens agrupadas, o aplicativo não deseja iniciar o processamento de qualquer grupo até que todas as mensagens dele tenham chegado. MQGMO_ALL_MSGS_AVAILABLE, portanto, é especificado para a primeira mensagem no grupo; para mensagens subsequentes no grupo, essa opção é desnecessária.

MQGMO_WAIT é usado neste exemplo. No entanto, embora a espera possa ser satisfeita se um novo grupo chegar, pelas razões em “Procurando mensagens em ordem lógica” na página 817, ela não será satisfeita e o cursor de procura já tiver passado a primeira mensagem lógica de um grupo e as mensagens restantes chegarem agora. No entanto, esperar um intervalo adequado assegura que o aplicativo não entre em loop constantemente enquanto espera novas mensagens ou segmentos.

MQGMO_LOGICAL_ORDER é usado de forma geral para assegurar que a varredura esteja em ordem lógica. Isso contrasta com o exemplo MQGET destrutivo, em que como cada grupo está sendo removido, MQGMO_LOGICAL_ORDER não é usado ao procurar a primeira (ou única) mensagem em um grupo.

Supõe-se que o buffer do aplicativo sempre seja grande o suficiente para conter a mensagem inteira, independentemente de se a mensagem foi segmentada ou não. Portanto, MQGMO_COMPLETE_MSG é especificado em cada MQGET.

Segue um exemplo de como procurar mensagens lógicas em um grupo:

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

O grupo é repetido até MQRC_NO_MSG_AVAILABLE ser retornado.

Procurando e recuperando de forma destrutiva
Neste exemplo, o aplicativo procura cada uma das mensagens lógicas dentro de um grupo, antes de decidir se recupera esse grupo de forma destrutiva.

A primeira parte deste exemplo é semelhante à anterior. No entanto, nesse caso, tendo navegado em um grupo inteiro, decidimos voltar e recuperá-lo de forma destrutiva.

Conforme cada grupo é removido neste exemplo, MQGMO_LOGICAL_ORDER não é usado ao procurar a primeira ou única mensagem em um grupo.

Segue um exemplo de procura e recuperação de forma destrutiva:

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...

if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

Evitando entrega repetida de mensagens procuradas
Usando determinadas opções open e opções get-message, é possível marcar mensagens como tendo sido procuradas para que não sejam recuperadas novamente pelo aplicativo atual ou outros de cooperação. As mensagens podem ser desmarcadas explícita ou automaticamente para torná-las disponíveis novamente para procura.

Se você procurar mensagens em uma fila, poderá recuperá-las em uma ordem diferente da ordem na qual as recuperaria tivesse obtido as mesmas destrutivamente. Especificamente, é possível procurar a mesma mensagem diversas vezes, o que não será possível se ela for removida da fila. Para evitar isso, é possível *marcar* mensagens conforme são procuradas e evitar recuperar mensagens marcadas. Isso às vezes é referido como *procurar com marca*. Para marcar mensagens procuradas, use a opção get message MQGMO_MARK_BROWSE_HANDLE e, para recuperar somente mensagens não marcadas, use MQGMO_UNMARKED_BROWSE_MSG. Se usar uma combinação de opções MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_HANDLE e emitir repetidos MQGETs, irá recuperar cada mensagem na fila da vez. Isso evita entrega repetida de mensagens, embora, MQGMO_BROWSE_FIRST seja usado para assegurar que as mensagens não sejam ignoradas. Essa combinação de opções pode ser representada pela constante única MQGMO_BROWSE_HANDLE. Quando não há mensagens na fila que não foram procuradas, MQRC_NO_MSG_AVAILABLE será retornado.

Se vários aplicativos estiverem procurando na mesma fila, eles podem abrir a fila com as opções MQOO_CO_OP e MQOO_BROWSE. A manipulação de objetos retornada por cada chamada MQOPEN é considerada como parte de um grupo de cooperação. Qualquer mensagem retornada por uma chamada MQGET especificando a opção MQGMO_MARK_BROWSE_CO_OP é considerada como marcada para esse conjunto de identificadores de cooperação.

Se uma mensagem tiver sido marcada por algum tempo, ela poderá ser automaticamente desmarcada pelo gerenciador de filas e disponibilizada para procura novamente. O atributo MsgMarkBrowseInterval do gerenciador de filas fornece o tempo em milissegundos que uma mensagem deve permanecer

marcada para o conjunto de identificadores de cooperação. Um MsgMarkBrowseInterval igual a -1 significa que as mensagens nunca são automaticamente desmarcadas.

Quando o processo único ou o conjunto de processos de cooperação de marcação de mensagens para, quaisquer mensagens marcadas serão desmarcadas.

Exemplos de procura cooperativa

Você pode executar várias cópias de um aplicativo dispatcher para procurar mensagens em uma fila e iniciar um consumidor com base no conteúdo de cada mensagem. Em cada dispatcher, abra a fila com MQOO_CO_OP. Isso indica que os dispatchers estão cooperando e estarão cientes das mensagens marcadas uns dos outros. Cada dispatcher faz, então, repetidas chamadas MQGET, especificando as opções MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_CO_OP (é possível usar a constante única MQGMO_BROWSE_CO_OP para representar essa combinação de opções). Cada aplicativo dispatcher recupera então somente as mensagens que ainda não foram marcadas por outros dispatchers em cooperação. O dispatcher inicializa um consumidor e passa o MsgToken retornado por MQGET para o consumidor, que destrutivamente obtém a mensagem da fila. Se o consumidor restaurar MQGET da mensagem, então, a mensagem estará disponível para um dos navegadores para novo dispatch, porque ela não está mais marcada. Se o consumidor não executar um MQGET na mensagem, então, após o MsgMarkBrowseInterval ter passado, o gerenciador de filas desmarca a mensagem para o conjunto de identificadores em cooperação e poderá ser despachada novamente.

Em vez de várias cópias do mesmo aplicativo dispatcher, você pode ter diversos aplicativos dispatcher diferentes procurando na fila, cada um apropriado para processar um subconjunto de mensagens na fila. Em cada dispatcher, abra a fila com MQOO_CO_OP. Isso indica que os dispatchers estão cooperando e estarão cientes das mensagens marcadas uns dos outros.

- Se a ordem de processamento de mensagens de um único dispatcher for importante, cada dispatcher faz chamadas MQGET repetidas, especificando as opções MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_HANDLE (ou MQGMO_BROWSE_HANDLE). Se a mensagem procurada for adequada para esse dispatcher processar, ele então faz uma chamada MQGET especificando MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP e o MsgToken retornado pela chamada MQGET anterior. Se a chamada for bem-sucedida, o dispatcher inicializa o consumidor, passando o MsgToken para ele.
- Se a ordem de processamento de mensagens não for importante e o for esperado que dispatcher processe a maioria das mensagens que encontrar, use as opções MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG e MQGMO_MARK_BROWSE_CO_OP (ou MQGMO_BROWSE_CO_OP). Se o dispatcher procurar uma mensagem que ele não pode processar, ele desmarca a mensagem chamando MQGET com a opção MQMO_MATCH_MSG_TOKEN, MQGMO_UNMARK_BROWSE_CO_OP e o MsgToken retornado anteriormente.

Alguns casos em que a chamada MQGET falha

Se determinados atributos de uma fila forem alterados usando a opção FORCE em um comando entre a emissão de um MQOPEN e uma chamada MQGET, a chamada MQGET falha e retorna o código de razão MQRC_OBJECT_CHANGED.

O gerenciador de filas marca a manipulação de objetos como não sendo mais válida. Isso também acontece se as mudanças se aplicarem a qualquer fila para a qual o nome da fila é resolvido. Os atributos que afetam a manipulação dessa forma são listados na descrição da chamada MQOPEN no MQOPEN. Se a sua chamada retornar o código de razão MQRC_OBJECT_CHANGED, feche a fila, reabra-a e, em seguida, tente obter uma mensagem novamente.

Se as operações get forem inibidas para uma fila a partir da qual você está tentando obter mensagens (ou qualquer fila para a qual o nome da fila é resolvido), a chamada MQGET falhará e retornará o código de razão MQRC_GET_INHIBITED. Isso acontece mesmo se você estiver usando a chamada MQGET para navegação. É possível obter uma mensagem com êxito se você tentar a chamada MQGET posteriormente, se o design do aplicativo for tal que outros programas mudarem os atributos de filas regularmente.

Se uma fila dinâmica (temporária ou permanente) foi excluída, chamadas MQGET que usam uma manipulação de objetos adquirida anteriormente falham e retornam o código de razão MQRC_Q_DELETED.

Escrevendo aplicativos de publicar/assinar

Inicie a escrever aplicativos de publicar/assinar IBM MQ.

Para uma visão geral dos conceitos de publicar/assinar, consulte o sistema de mensagens Publicar/assinar.

Consulte os tópicos a seguir para obter informações sobre como escrever diferentes tipos de aplicativos de publicar/assinar:

- "Gravando aplicativos de publicador" na página 822
- "Escrevendo aplicativos de assinante" na página 828
- "Ciclos de vida de publicar/assinar" na página 845
- "Propriedades de mensagem de publicação/assinatura" na página 850
- "Ordenação de mensagens" na página 851
- "Interceptando publicações" na página 852
- "Opções de publicação" na página 860
- "Opções de Assinatura" na página 860

Conceitos relacionados

"Conceitos de desenvolvimento de aplicativos" na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

"Desenvolvendo aplicativos para o IBM MQ" na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

"Considerações de design para aplicativos IBM MQ" na página 50
Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

"Escrevendo um aplicativo processual para enfileiramento" na página 732
Use essas informações para aprender como escrever aplicativos de enfileiramento, conectar e desconectar de um gerenciador de filas, publicar/assinar e abrir e fechar objetos.

"Escrevendo aplicativos clientes processuais" na página 924
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

"Construindo um aplicativo processual" na página 1012
É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

"Manipulando erros de programa processual" na página 1050
Estas informações explicarão os erros associados às chamadas MQI de aplicativos, ao realizar uma chamada ou quando sua mensagem for entregue para seu destino final.

Tarefas relacionadas

"Usando os programas processuais de amostra do IBM MQ" na página 1070
Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

Gravando aplicativos de publicador

Comece a escrever aplicativos do publicador estudando dois exemplos. O primeiro é modelado o mais próximo possível em um aplicativo ponto a ponto colocando mensagens em uma fila e o segundo demonstra a criação de tópicos dinamicamente - um padrão mais comum para aplicativos publicadores.

Escrever um simples aplicativo do publicador IBM MQ é exatamente como escrever um aplicativo de ponto a ponto IBM MQ que coloca mensagens em uma fila (Tabela 121 na página 822). A diferença é que as mensagens são colocadas em um tópico e não em uma fila.

Tabela 121. Padrão do programa IBM MQ ponto a ponto versus de publicação/assinatura.

Etapa	**Ponto a ponto de Chamada MQ**	**Publicação Chamada MQ**
Conecta-se a um gerenciador de filas	MQCONN	MQCONN
Abrir fila	MQOPEN	
Abra tópico		MQOPEN
Coloca mensagem(ns)	MQPUT	MQPUT
tópico Fechar		MQCLOSE
Fechar fila	MQCLOSE	
Desconecte a partir do gerenciador de filas	MQDISC	MQDISC

Para concretizar isso, há dois exemplos de aplicativos para publicar preços de ações. No primeiro exemplo (“Exemplo 1: publicador em um tópico fixo” na página 822), que é modelado de perto colocando mensagens em uma fila, o administrador cria uma definição de tópico de maneira semelhante à criação de uma fila. O programador codifica MQPUT para gravar mensagens no tópico, em vez de gravá-las em uma fila. No segundo exemplo (“Exemplo 2: publicador em um tópico de variável” na página 825), o padrão de interação do programa com o IBM MQ é semelhante. A diferença é que o programador fornece o tópico ao qual a mensagem é gravada, em vez do administrador. Na prática, isto geralmente significa que a sequência de tópicos é conteúdo definido ou fornecido por outra origem, como entrada humana através de um navegador.

Conceitos relacionados

“Escrevendo aplicativos de assinante” na página 828
Comece o gravar aplicativos de assinante estudando três exemplos: um aplicativo IBM MQ consumindo mensagens de uma fila, um aplicativo que cria uma assinatura e não requer conhecimento de enfileiramento e, finalmente, um exemplo que usa tanto assinaturas quanto enfileiramento.

Referências relacionadas

DEFINE TOPIC
DISPLAYTOPIC
DISPLAYTPSTATUS

Exemplo 1: publicador em um tópico fixo

Um programa IBM MQ para ilustrar a publicação em um tópico definido administrativamente.

Nota: O estilo de codificação compacto destina-se à capacidade de leitura e não ao uso na produção.

Consulte a saída em Figura 65 na página 823

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
                publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figura 64. Publicador simples do IBM MQ para um tópico fixo.

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Figura 65. Saída de amostra do primeiro exemplo de publicador

As linhas de código selecionadas a seguir ilustram aspectos de escrever um aplicativo publicador para o IBM MQ.

char topicNameDefault[] = "IBMSTOCKPRICE";
Um nome de tópico padrão é definido no programa. É possível substituí-lo ao fornecer o nome de um objeto do tópico diferente como o primeiro argumento para o programa.

MQCHAR resTopicStr[151];
`resTopicStr` é apontado por `td.ResObjectString.VSPtr` e é usado por `MQOPEN` para retornar a sequência de tópicos resolvida. Torne o comprimento de `resTopicStr` um maior do que o comprimento passado em `td.ResObjectString.VSBufSize` para fornecer espaço para a finalização em nulo.

memset (resTopicStr, 0, sizeof(resTopicStr));
Inicialize `resTopicStr` para nulos para assegurar que a sequência de tópicos resolvida retornada em um `MQCHARV` seja finalizada em nulo.

td.ObjectType = MQOT_TOPIC
Há um novo tipo de objeto para publicar/assinar: o *objeto do tópico*.

td.Version = MQOD_VERSION_4;
Para usar o novo tipo de objeto, deve-se usar pelo menos a *versão 4* do descritor de objeto.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
O `topicName` é o nome de um objeto do tópico, às vezes chamado de um objeto do tópico administrativo. No exemplo, o objeto do tópico precisa ser criado com antecedência, usando o IBM MQ Explorer ou este comando do MQSC:

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
A sequência de tópicos resolvida é ecoada no `printf` final no programa. Configure a estrutura `MQCHARV ResObjectString` para o IBM MQ retornar a sequência resolvida ao programa.

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
Abra o tópico para saída; exatamente como abrir uma fila para saída.

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
Você deseja que novos assinantes possam receber a publicação e, especificando `MQPMO_RETAIN` no publicador, ao iniciar um assinante, ele recebe a publicação mais recente, publicada antes do assinante ter sido iniciado, como sua primeira publicação correspondente. A alternativa é fornecer aos assinantes publicações feitas somente após eles terem sido iniciados. Além disso, um assinante tem a opção de recusar a receber uma publicação retida especificando `MQSO_NEW_PUBLICATIONS_ONLY` em sua assinatura.

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
Some 1 ao comprimento da sequência passada para `MQPUT` para passar o caractere de finalização nulo para o IBM MQ como parte do buffer de mensagem.

O que o primeiro exemplo demonstra? O exemplo imita o mais próximo possível o padrão tradicional experimentado e testado para escrever programa IBM MQ de ponto a ponto. Uma característica importante do padrão de programação do IBM MQ é que o programador não está preocupado para onde as mensagens são enviadas. A tarefa do programador é conectar a um gerenciador de filas e passar a ele as mensagens que devem ser distribuídas a destinatários. No paradigma ponto a ponto, o programador abre uma fila (provavelmente uma fila de alias) que o administrador configurou. A fila de alias roteia mensagens para uma fila de destino, seja no gerenciador de filas locais ou para um gerenciador de filas remotas. Enquanto as mensagens estão esperando para serem entregues, elas são armazenadas em filas em algum lugar entre a origem e o destino.

No padrão de publicar/assinar, em vez de abrir uma fila, o programador abre um tópico. Em nosso exemplo, o tópico é associado a uma sequência de tópicos por um administrador. O gerenciador de filas encaminha a publicação, usando filas, para assinantes locais ou remotos que têm assinaturas que correspondem à sequência de tópicos da publicação. Se as publicações forem retidas, o gerenciador de filas mantém a cópia mais recente da publicação, mesmo se não tiver assinantes agora. A publicação retida está disponível para encaminhamento para futuros assinantes. O aplicativo publicador não

desempenha qualquer papel na seleção ou no roteamento da publicação para um destino; sua tarefa é criar e colocar publicações nos tópicos definidos pelo administrador.

Este exemplo de tópico fixo é atípico de muitos aplicativos de publicar/assinar: é estático. Ele requer que um administrador defina as sequências de tópicos e mude os tópicos nos quais são publicadas. Aplicativos de publicar/assinar geralmente precisam conhecer parte ou toda a árvore de tópicos. Possivelmente, tópicos mudem com frequência ou, embora os tópicos não mudem muito, o número de combinações de tópicos é grande e é demasiado oneroso para um administrador definir um nó de tópico para cada sequência de tópicos nas quais possa ser preciso publicar. Possivelmente, as sequências de tópicos não são conhecidas antes da publicação; um aplicativo publicador pode usar informações do conteúdo de publicação para especificar uma sequência de tópicos ou pode ter informações sobre sequências de tópicos para publicar a partir de outra origem, como inserção manual a partir de um navegador. Para fornecer estilos mais dinâmicos de publicação, o exemplo a seguir mostra como criar tópicos dinamicamente, como parte do aplicativo publicador.

Tópicos colocam publicadores e assinantes em pares. Projetar as regras, ou da arquitetura, para denominar tópicos e organizá-los em árvores de tópicos é uma etapa importante no desenvolvimento de uma solução de publicar/assinar. Verifique cuidadosamente até que ponto a organização da árvore de tópicos junta programas de publicador e assinante e liga os mesmos ao conteúdo da árvore de tópicos. Pergunte a si mesmo se mudanças na árvore de tópicos afetam aplicativos de publicador e assinante e como é possível minimizar o efeito. Integrada à arquitetura do modelo publicar/assinar do IBM MQ está a noção de um objeto do tópico administrativo que fornece a parte raiz, ou subárvore raiz, de um tópico. O objeto do tópico fornece a opção de definir a parte raiz da árvore de tópicos administrativamente que simplifica a programação de aplicativos e operações, e, consequentemente, melhora a sustentabilidade. Por exemplo, se você estiver implementando diversos aplicativos de publicar/assinar que tenham árvores de tópicos isoladas, então, ao definir administrativamente a parte raiz da árvore de tópicos, será possível garantir o isolamento de árvores de tópicos, mesmo que não houver consistência nas convenções de nomenclatura de tópicos adotadas pelos diferentes aplicativos.

Na prática, os aplicativos do publicador cobrem um espectro de usar exclusivamente tópicos fixos, como neste exemplo, e tópicos variáveis, como no próximo. “Exemplo 2: publicador em um tópico de variável” na página 825 também demonstra combinar o uso de tópicos e as sequências de tópicos.

Conceitos relacionados

“Exemplo 2: publicador em um tópico de variável” na página 825
Um programa do WebSphere MQ para ilustrar a publicação em um tópico definido programaticamente.

“Escrevendo aplicativos de assinante” na página 828
Comece o gravar aplicativos de assinante estudando três exemplos: um aplicativo IBM MQ consumindo mensagens de uma fila, um aplicativo que cria uma assinatura e não requer conhecimento de enfileiramento e, finalmente, um exemplo que usa tanto assinaturas quanto enfileiramento.

Exemplo 2: publicador em um tópico de variável
Um programa do WebSphere MQ para ilustrar a publicação em um tópico definido programaticamente.

Nota: O estilo de codificação compacto destina-se à capacidade de leitura e não ao uso na produção.

Consulte a saída em Figura 67 na página 826.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figura 66. Publicador IBM MQ simples para um tópico de variável.

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Figura 67. Saída de amostra do exemplo do segundo publicador

Há alguns pontos a serem observados sobre este exemplo.

char topicNameDefault[] = "STOCKS";
O padrão de nome de tópico STOCKS define parte da sequência de tópicos. É possível substituir esse nome de tópico fornecendo-o como o primeiro argumento ao programa ou eliminar o uso do nome do tópico fornecendo / como o primeiro parâmetro.

char topicString[101] = "IBM/PRICE";
IBM/PRICE é a sequência de tópicos padrão. É possível substituir essa sequência de tópico fornecendo-a como o segundo argumento ao programa.

O gerenciador de filas combina a sequência de tópicos fornecida pelo objeto do tópico STOCKS, "NYSE", com a sequência de tópicos fornecida pelo programa "IBM/PRICE" e insere um "/" entre as duas sequências de tópicos. O resultado é a sequência de tópicos resolvidos "NYSE/IBM/PRICE". A sequência de tópicos resultante é a mesma que a definida no objeto do tópico IBMSTOCKPRICE e tem precisamente o mesmo efeito.

O objeto do tópico administrativo associado à sequência de tópico resolvida não é necessariamente o mesmo objeto do tópico que foi passado ao MQOPEN pelo publicador. O IBM MQ usa a árvore implícita na sequência de tópicos resolvida para descobrir qual objeto do tópico administrativo define os atributos associados à publicação.

Digamos que existam dois objetos do tópico A e B e A define tópico "a" e B define tópico "a/b" (Figura 68 na página 827). Se o programa do publicador se referir ao objeto do tópico A e fornecer sequência de tópicos "b", resolvendo o tópico para a sequência de tópicos "a/b", então a publicação herda suas propriedades do objeto do tópico B porque o tópico corresponde à sequência de tópicos "a/b" definida para B.

if (strcmp(argv[1],"/"))
argv[1] é o topicName fornecido opcionalmente. "/" é inválido como um nome de tópico; aqui, isso significa que não há nome de tópico e a sequência de tópicos é fornecida totalmente pelo programa. A saída em Figura 67 na página 826 mostra a sequência de tópicos inteira a ser fornecida dinamicamente pelo programa.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
Para o caso padrão, o topicName opcional precisa ser criado com antecedência usando o IBM MQ Explorer ou esse comando MQSC:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
A sequência de tópicos é um campo MQCHARV no descritor de tópicos

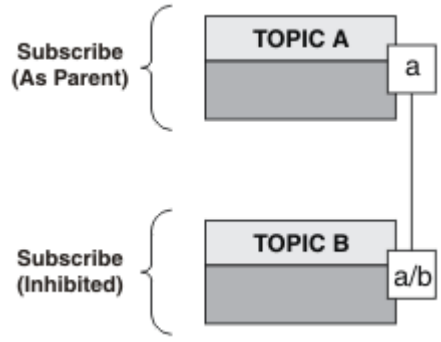


Figura 68. Associações de objeto de tópico

O que o segundo exemplo demonstra? Embora o código seja muito semelhante ao primeiro exemplo - efetivamente, há somente duas linhas de diferença - o resultado é um programa significativamente diferente do primeiro. O programador controla os destinos aos quais publicações são enviadas. Em

conjunto com a entrada do administrador mínimo usada para projetar aplicativos de assinante, tópicos ou filas não precisam ser predefinidos para rotear publicações de publicadores a assinantes.

No paradigma do sistema de mensagens ponto a ponto, as filas têm de ser definidas antes que as mensagens sejam capazes de fluir. Para publicar/assinar, elas não têm, embora IBM MQ implemente publicação/assinatura usando seu sistema de enfileiramento subjacente; os benefícios de entrega garantida, transacionalidade e acoplamento fraco associados ao sistema de mensagens e enfileiramento são herdados por aplicativos de publicação/assinatura.

Um designer tem que decidir se programas, tanto publicadores como assinantes, devem estar cientes da árvore de tópicos subjacente ou não e também se os programas assinantes estão cientes de enfileiramento ou não. Estude os aplicativos de exemplo de assinante em seguida. Eles são projetados para serem usados com os exemplos de publicador, geralmente publicando e assinando ao `NYSE/IBM/PRICE`.

Conceitos relacionados

“Exemplo 1: publicador em um tópico fixo” na página 822
Um programa IBM MQ para ilustrar a publicação em um tópico definido administrativamente.

“Escrevendo aplicativos de assinante” na página 828
Comece o gravar aplicativos de assinante estudando três exemplos: um aplicativo IBM MQ consumindo mensagens de uma fila, um aplicativo que cria uma assinatura e não requer conhecimento de enfileiramento e, finalmente, um exemplo que usa tanto assinaturas quanto enfileiramento.

Escrevendo aplicativos de assinante

Comece o gravar aplicativos de assinante estudando três exemplos: um aplicativo IBM MQ consumindo mensagens de uma fila, um aplicativo que cria uma assinatura e não requer conhecimento de enfileiramento e, finalmente, um exemplo que usa tanto assinaturas quanto enfileiramento.

No Tabela 122 na página 828 os três estilos de consumidor ou assinante são listados, juntamente com as sequências de chamadas de função IBM MQ que os caracterizam.

1. O primeiro estilo, MQ Publicação Consumidor, é idêntico a um programa MQ ponto a ponto que executa apenas MQGET. O aplicativo não tem conhecimento de que está consumindo as publicações – está simplesmente lendo mensagens de uma fila. A assinatura que faz com que as publicações sejam encaminhadas para a fila é criada administrativamente usando o IBM MQ Explorer ou um comando.
2. O segundo estilo é o padrão preferido para a maioria dos aplicativos de assinantes. O aplicativo do assinante cria a assinatura e, em seguida, obtém publicações. O de gerenciamento de filas são todas executadas pelo gerenciador de filas. Isso é conhecido como um *assinante gerenciado*.
3. No terceiro estilo, o aplicativo de assinante é responsável pela especificação da fila que será usada para reter publicações, abrir e fechar essa fila e emitir assinaturas para preencher a fila com publicações. Isso é conhecido como um *assinante não gerenciado*.

Uma maneira de entender esses estilos é estudar os programas C de exemplo listados em Tabela 122 na página 828 para cada um dos estilos. Os exemplos são projetados para serem executados em conjunto com o exemplo publicador localizado em “Gravando aplicativos de publicador” na página 822.

Tabela 122. Padrões de programa IBM MQ ponto a ponto vs. de assinatura.

Etapa	**consumidor de mensagens do MQ**	**“Exemplo 1: consumidor de publicação do MQ” na página 829**	**“Exemplo 2: assinante do MQ gerenciado” na página 831**	**“Exemplo 3: assinante do MQ não gerenciado” na página 836**
Conecta-se a um gerenciador de filas	MQCONN	MQCONN	MQCONN	MQCONN
Abrir fila	MQOPEN	MQOPEN		MQOPEN
Assinar			MQSUB	MQSUB

Tabela 122. Padrões de programa IBM MQ ponto a ponto vs. de assinatura. (continuação)				
Etapa	**consumidor de mensagens do MQ**	**"Exemplo 1: consumidor de publicação do MQ" na página 829**	**"Exemplo 2: assinante do MQ gerenciado" na página 831**	**"Exemplo 3: assinante do MQ não gerenciado" na página 836**
Obtém mensagem(ns)	MQGET	MQGET	MQGET	MQGET
Fechar fila	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Fechar assinatura			MQCLOSE	MQCLOSE
Desconecte a partir do gerenciador de filas	MQDISC	MQDISC	MQDISC	MQDISC

Usar MQCLOSE sempre é opcional, seja para liberar recurso, para passar opções do MQCLOSE ou apenas para simetria com MQOPEN. Como é improvável que você precise especificar as opções MQCLOSE quando a fila de assinatura for fechada no caso assinante Managed MQ e o argumento simetria não é relevante, a fila de assinatura não é explicitamente fechada em Exemplo 2: assinante Managed MQ.

Outra maneira de entender os padrões de aplicativos de publicação/assinatura é analisar as interações entre as diferentes entidades envolvidas. Linha de vida ou diagramas de sequência UML são uma boa maneira de estudar interações. Três exemplos de linha de vida são descritos em "Ciclos de vida de publicar/assinar" na página 845.

Exemplo 1: consumidor de publicação do MQ
O consumidor da publicação MQ é um consumidor de mensagens IBM MQ que não assina tópicos em si.

Para criar a fila de assinatura e publicação para este exemplo, execute os seguintes comandos ou defina os objetos usando o IBM MQ Explorer.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

A assinatura `IBMSTOCKPRICESUB` faz referência ao objeto do tópico `IBMSTOCK` criado para o exemplo de publicador e a fila local `STOCKTICKER`. O objeto do tópico `IBMSTOCK` define a sequência de tópicos que é usada na assinatura, `NYSE/IBM/PRICE`. Observe que o objeto do tópico e a fila usada para receber publicações precisam ser definidos antes de a assinatura ser criada.

Existem várias máscaras valiosas para o padrão do consumidor de publicação MQ:

1. Multiprocessamento: compartilhar do trabalho de publicações de leitura. Todas as publicações vão para a única fila associada ao tópico de assinatura. Vários consumidores podem abrir a fila usando `MQOO_INPUT_SHARED`.
2. Assinaturas gerenciadas centralmente. Os aplicativos não constroem sua própria assinatura ou assinaturas de tópicos; o administrador é responsável por aonde as publicações são enviadas.
3. Concentração de assinatura: várias assinaturas diferentes podem ser enviadas para uma única fila.
4. Durabilidade da assinatura: a fila recebe todas as publicações estando ou não os consumidores ativos.
5. Migração e coexistência: o código do consumidor trabalha igualmente bem para um ponto a ponto e para um cenário de publicação/assinatura.

A assinatura cria um relacionamento entre a sequência de tópicos `NYSE/IBM/PRICE` e a fila `STOCKTICKER`. As publicações, incluindo qualquer publicação retida atualmente, são encaminhadas para `STOCKTICKER` a partir do momento em que a assinatura for criada.

Uma assinatura administrativamente criada pode ser gerenciada ou não gerenciada. Uma assinatura gerenciada entre em vigor assim que ela tiver sido criada, exatamente como uma assinatura não

gerenciada. Nem todas as máscaras padrão estão disponíveis para uma assinatura gerenciada. Consulte “Exemplo 3: assinante do MQ não gerenciado” na página 836

Nota: O estilo de codificação compacto destina-se à capacidade de leitura e não ao uso na produção.

Os resultados são mostrados em Figura 70 na página 830.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figura 69. Consumidor de publicação MQ.

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

Figura 70. A saída do consumidor de publicação MQ

Tem algumas dicas padrão de linguagem de programação IBM MQ *C* dicas para que esteja ciente:

memset(publication, 0, sizeof(publicationBuffer));
: Assegure-se de que a mensagem tenha um nulo final para fácil formatação usando `printf`. O exemplo do publicador inclui o nulo final no buffer de mensagem passado para o `MQPUT` incluindo 1 ao `strlen(publication)`. Configurar `MQCHAR` buffers para nulo é bom para o estilo de programação de programas C IBM MQ que usam os buffers para armazenar sequências, garantindo que um nulo siga uma matriz de caracteres que não preenche totalmente o buffer.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
: Reserve um nulo ao final do buffer de mensagem para assegurar que a mensagem retornada tem final nulo em caso de `if (messlen == strlen(publication));` ser true. Essa dica complementa a anterior e assegura que haja pelo menos uma nula no `publicationBuffer` que não será sobrescrita pelo conteúdo de `publication`.

Conceitos relacionados

“Exemplo 2: assinante do MQ gerenciado” na página 831
O assinante do MQ gerenciado é o padrão preferencial para a maioria dos aplicativos de assinante. Uma assinatura gerenciada é uma em que o IBM MQ manipula a assinatura e faz o registro e a remoção do registro para você. O exemplo *não* requer nenhuma definição administrativa de filas, tópicos ou assinaturas.

“Exemplo 3: assinante do MQ não gerenciado” na página 836
O assinante não gerenciado é uma classe importante de aplicativo de assinante. Com ele, você combina os benefícios de publicar/assinar com *controle* de enfileiramento e consumo de publicações. Uma assinatura não gerenciada é onde o aplicativo é responsável. para especificar a fila na qual as assinaturas são armazenadas O exemplo demonstra maneiras diferentes de combinar assinaturas e filas.

“Gravando aplicativos de publicador” na página 822
Comece a escrever aplicativos do publicador estudando dois exemplos. O primeiro é modelado o mais próximo possível em um aplicativo ponto a ponto colocando mensagens em uma fila e o segundo demonstra a criação de tópicos dinamicamente - um padrão mais comum para aplicativos publicadores.

Exemplo 2: assinante do MQ gerenciado

O assinante do MQ gerenciado é o padrão preferencial para a maioria dos aplicativos de assinante. Uma assinatura gerenciada é uma em que o IBM MQ manipula a assinatura e faz o registro e a remoção do registro para você. O exemplo *não* requer nenhuma definição administrativa de filas, tópicos ou assinaturas.

Esse tipo mais simples de assinante gerenciado geralmente usa uma assinatura *não durável*. O exemplo foca uma assinatura não durável. A assinatura perdura somente durante a existência do identificador de assinatura de `MQSUB`. Quaisquer publicações que correspondam à sequência de tópicos durante a existência da assinatura são enviadas para a fila de assinaturas (e possivelmente uma publicação retida se a sinalização `MQSO_NEW_PUBLICATIONS_ONLY` não estiver configurada ou padronizada, uma publicação anterior correspondente à sequência de tópicos foi retida e a publicação foi persistente ou o gerenciador de filas não foi finalizado desde que a criação da publicação).

Também é possível usar uma assinatura *durável* com este padrão. Normalmente, se uma assinatura durável gerenciada for usada, isso é feito por razões de confiabilidade, em vez de para estabelecer uma assinatura que, sem a ocorrência de qualquer erro, poderia sobreviver além do assinante. Para obter mais informações sobre diferentes ciclos de vida associados a assinaturas gerenciadas, não gerenciadas, duráveis e não duráveis, consulte a seção de tópicos relacionada.

Assinaturas duráveis são frequentemente associadas a publicações persistentes e assinaturas não duráveis a publicações não persistentes, mas não há relacionamento necessário entre durabilidade de assinatura e persistência de publicação. Todas as quatro combinações de persistência e durabilidade são possíveis.

Para o caso de não duráveis gerenciadas considerado, o gerenciador de filas cria uma fila de assinaturas que é limpa e excluída quando a fila é fechada. As publicações são removidas da fila quando a assinatura não durável é fechada.

Os aspectos importantes do padrão não durável gerenciada exemplificado por este código são os seguintes:

1. Assinatura on demand: a sequência de tópicos da assinatura é dinâmica. É fornecida pelo aplicativo quando ele é executado.
2. Fila autogerenciada: a fila de assinatura é autodefinida e gerenciada.
3. Ciclo de vida de assinatura autogerenciada: as assinaturas *não duráveis* existem somente para a duração do aplicativo de assinante.
   - Se você definir uma assinatura gerenciada *durável*, então, ela resulta em uma fila de assinaturas permanente e as publicações continuam a ser armazenadas nela sem nenhum programa de assinante ativo. O gerenciador de filas exclui a fila (e limpa quaisquer publicações não recuperadas da mesma) somente após o aplicativo ou o administrador ter optado por excluir a assinatura. A assinatura pode ser excluída usando um comando administrativo ou fechando a assinatura com a opção `MQCO_REMOVE_SUB`.
   - Considere configurar `SubExpiry` para assinaturas duráveis de forma que as publicações deixem de ser enviadas para a fila e o assinante possa consumir quaisquer publicações restantes antes de remover a assinatura e fazer com que o gerenciador de filas exclua a fila e todas as publicações restantes na mesma.
4. Implementação de sequência de tópicos flexível: o gerenciamento de tópicos de assinatura é simplificado definindo-se a parte raiz da assinatura usando um tópico definido administrativamente. A parte raiz da árvore de tópicos é então ocultada do aplicativo. Ocultando a parte raiz, um aplicativo pode ser implementado sem que o aplicativo inadvertidamente crie uma árvore de tópicos que se sobreponha a outra árvore de tópicos criada por outra instância ou outro aplicativo.
5. Tópicos administrados: usando uma sequência de tópicos na qual a primeira parte corresponde a um objeto de tópico definido administrativamente, as publicações são gerenciadas de acordo com os atributos do objeto do tópico.
   - Por exemplo, se a primeira parte da sequência de tópicos corresponder à sequência de tópicos associada a um objeto do tópico em cluster, então, a assinatura poderá receber publicações de outros membros do cluster
   - A correspondência seletiva de objetos do tópico definidos administrativamente e assinaturas definidas programaticamente permite combinar os benefícios de ambos. O administrador fornece atributos para tópicos e o programador define subtópicos dinamicamente sem se preocupar com o gerenciamento de tópicos.
   - É a sequência de tópicos resultante que é usada para corresponder o objeto do tópico que fornece os atributos associados ao tópico, não necessariamente o objeto do tópico denominado em `sd.Objectname`, embora eles geralmente acabam sendo um único e o mesmo. Consulte “Exemplo 2: publicador em um tópico de variável” na página 825.

Ao tornar a assinatura durável no exemplo, as publicações continuam a ser enviadas para a fila de assinaturas depois que o assinante tiver fechado a assinatura com a opção `MQCO_KEEP_SUB`. A fila continua a receber publicações quando o assinante não está ativo. É possível substituir esse comportamento ao criar a assinatura com a opção `MQSO_PUBLICATIONS_ON_REQUEST` e usar `MQSUBRQ` para solicitar a publicação retida.

A assinatura pode ser continuada posteriormente abrindo a assinatura com a opção `MQCO_RESUME`.

É possível usar o identificador de fila, `Hobj`, retornado por `MQSUB` de várias maneiras. O identificador de fila é usado no exemplo para consultar sobre o nome da fila de assinaturas. Filas gerenciadas são abertas usando as filas modelo padrão `SYSTEM.NDURABLE.MODEL.QUEUE` ou `SYSTEM.DURABLE.MODEL.QUEUE`. É possível substituir os padrões fornecendo suas próprias filas modelo duráveis e não duráveis tópico a tópico como propriedades do objeto do tópico associado à assinatura.

Independentemente dos atributos herdados das filas modelo, não é possível reutilizar um identificador de fila gerenciado para criar uma assinatura adicional. Nem é possível obter outro identificador para a fila gerenciada abrindo a fila gerenciada pela segunda vez usando o nome da fila retornado. A fila se comporta como se tivesse sido aberta para entrada exclusiva.

Filas não gerenciadas são mais flexíveis do que filas gerenciadas. É possível, por exemplo, compartilhar filas não gerenciadas ou definir várias assinaturas em uma fila. O próximo exemplo demonstra como combinar assinaturas com uma fila de assinaturas não gerenciada.

Nota: O estilo de codificação compacto destina-se à capacidade de leitura e não ao uso na produção.

Os resultados são mostrados em Figura 73 na página 834.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){               /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
}
```

Figura 71. Assinante do MQ gerenciado – parte 1: declarações e manipulação de parâmetro.

Há alguns comentários adicionais para fazer sobre as declarações neste exemplo.

MQHOBJ Hobj = MQHO_NONE;
Não é possível abrir explicitamente uma fila de assinaturas gerenciada não durável para receber publicações, mas você precisa alocar armazenamento para a identificação de objeto que o gerenciador de filas retorna ao abrir a fila para você. É importante inicializar o identificador para MQHO_OBJECT. Isso indica ao gerenciador de filas que ele precisa retornar um identificador de fila para a fila de assinaturas.

MQSD sd = {MQSD_DEFAULT};
O novo descritor de assinatura, usado em MQSUB.

MQCHAR48 qName;
Embora o exemplo não requeira conhecimento da fila de assinaturas, ele consulta o nome da fila de assinaturas - a ligação MQINQ é um pouco estranha na linguagem C, então você pode achar essa parte do exemplo útil para um estudo.

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

Figura 72. Assinante do MQ gerenciado – parte 2: corpo do código.

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020    "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

Figura 73. Assinante do MQ

Há alguns comentários adicionais para fazer sobre o código neste exemplo.

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
Se topicName for nulo ou estiver em branco *(valor padrão)*, o nome do tópico não será usado para calcular a sequência de tópicos resolvida.

sd.ObjectString.VSPtr = topicString;
Em vez de usar exclusivamente um objeto do tópico predefinido, neste exemplo, o programador fornece um objeto do tópico e uma sequência de tópicos, que são combinados por MQSUB. Observe que a sequência de tópicos é uma estrutura MQCHARV.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Uma alternativa para configurar o comprimento de um campo MQCHARV.

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
Após definir a sequência de tópicos, as sinalizações sd.Options precisam de muita atenção. Há muitas opções, o exemplo especifica somente as mais comumente usadas. As outras opções usam os valores padrão.

1. Como a assinatura é *não durável*, ou seja, tem um tempo de vida da assinatura aberta no aplicativo, configure a sinalização MQSO_CREATE. Também é possível configurar a sinalização *(padrão)* MQSO_NON_DURABLE para capacidade de leitura.
2. Complementando MQSO_CREATE há MQSO_RESUME. Ambas as sinalizações podem ser configuradas juntas; o gerenciador de filas cria uma nova assinatura ou continua uma assinatura existente, o que for apropriado. No entanto, se você especificar MQSO_RESUME, deve-se também inicializar a estrutura MQCHARV para sd.SubName, mesmo se não houver nenhuma assinatura para continuar. Falha ao inicializar SubName resulta em um código de retorno 2440: MQRC_SUB_NAME_ERROR de MQSUB.

   Nota: MQSO_RESUME sempre e ignorado para uma assinatura gerenciada não durável, mas especificá-lo sem inicializar a estrutura MQCHARV para sd.SubName causa o erro.
3. Além disso, há uma terceira sinalização que afeta como a assinatura é aberta, MQSO_ALTER. Dadas as permissões corretas, as propriedades de uma assinatura continuada são mudadas para corresponder a outros atributos especificados em MQSUB.

   Nota: Pelo menos uma das sinalizações MQSO_CREATE, MQSO_RESUME e MQSO_ALTER deve ser especificada. Consulte Opções (MQLONG). Há exemplos de uso de todas as três sinalizações em “Exemplo 3: assinante do MQ não gerenciado” na página 836.
4. Configure MQSO_MANAGED para o gerenciador de filas gerenciar a assinatura para você automaticamente.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Como opção, omita a configuração do comprimento de MQCHARV para sequências finalizadas em nulo e use a sinalização de terminador nulo.

sd.ResObjectString.VSPtr = resTopicStr;
A sequência de tópicos resultante é ecoada no primeiro printf no programa. Configure MQCHARV ResObjectString para o IBM MQ retornar a sequência resolvida de volta para o programa.

Nota: resTopicStringBuffer é inicializado para nulos em memset(resTopicStr, 0, sizeof(resTopicStrBuffer)). As sequências de tópicos retornadas não terminam com nulos à direita.

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
Configure o tamanho do buffer de sd.ResObjectString para um a menos que seu tamanho real. Isso evita sobrescrever o terminador nulo que é fornecido, caso a sequência de tópicos resolvida preencha todo o buffer.

Nota: Nenhum erro será retornado se a sequência de tópicos for mais longa que sizeof(resTopicStrBuffer)-1. Mesmo se VSLength > VSBufSiz, o comprimento retornado em sd.ResObjectString.VSLength será o comprimento da sequência completa e não necessariamente o comprimento da sequência retornada. Teste sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz para confirmar a sequência de tópicos concluída.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
A função MQSUB cria uma assinatura. Se for não durável, provavelmente você não está interessado em seu nome, embora seja possível inspecionar seu status no IBM MQ Explorer. É possível fornecer o parâmetro sd.SubName como entrada para que saiba o nome a procurar; é evidente que precisa evitar conflitos de nomes com outras assinaturas.

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
Fechar a assinatura e a fila de assinaturas é opcional. No exemplo, a assinatura é fechada, mas não a fila. A opção MQCLOSE MQCO_REMOVE_SUB é o padrão nesse caso de qualquer forma, já que a assinatura é não durável. Usar MQCO_KEEP_SUB é um erro.

Nota: a *fila* de assinaturas não é fechada por MQSUB e seu identificador, Hobj, permanece válido até que a fila seja fechada por MQCLOSE ou MQDISC. Se o aplicativo for finalizado de forma prematura, a fila e a assinatura são limpas pelo gerenciador de filas algum tempo após a finalização do aplicativo.

Conceitos relacionados

"Exemplo 1: consumidor de publicação do MQ" na página 829
O consumidor da publicação MQ é um consumidor de mensagens IBM MQ que não assina tópicos em si.

"Exemplo 3: assinante do MQ não gerenciado" na página 836
O assinante não gerenciado é uma classe importante de aplicativo de assinante. Com ele, você combina os benefícios de publicar/assinar com *controle* de enfileiramento e consumo de publicações. Uma assinatura não gerenciada é onde o aplicativo é responsável. para especificar a fila na qual as assinaturas são armazenadas O exemplo demonstra maneiras diferentes de combinar assinaturas e filas.

"Gravando aplicativos de publicador" na página 822
Comece a escrever aplicativos do publicador estudando dois exemplos. O primeiro é modelado o mais próximo possível em um aplicativo ponto a ponto colocando mensagens em uma fila e o segundo demonstra a criação de tópicos dinamicamente - um padrão mais comum para aplicativos publicadores.

Exemplo 3: assinante do MQ não gerenciado
O assinante não gerenciado é uma classe importante de aplicativo de assinante. Com ele, você combina os benefícios de publicar/assinar com *controle* de enfileiramento e consumo de publicações. Uma assinatura não gerenciada é onde o aplicativo é responsável. para especificar a fila na qual as assinaturas são armazenadas O exemplo demonstra maneiras diferentes de combinar assinaturas e filas.

O padrão não gerenciado é mais comumente associado a assinaturas *duráveis* do que *não duráveis*. Geralmente, o ciclo de vida de uma assinatura criada por um assinante não gerenciado é independente do ciclo de vida do aplicativo de assinatura em si. Ao tornar a assinatura durável, a assinatura recebe publicações mesmo quando nenhum aplicativo de assinatura estiver ativo.

É possível criar assinaturas duráveis *gerenciadas* para atingir o mesmo resultado, mas alguns aplicativos requerem mais flexibilidade e controle sobre filas e mensagens do que é possível com uma assinatura gerenciada. Para uma assinatura gerenciada durável, o gerenciador de filas cria uma fila permanente para as publicações que correspondem ao tópico de assinatura. Ele exclui a fila e as publicações associadas quando a assinatura é excluída.

Geralmente, assinaturas *gerenciadas* duráveis são usadas se o ciclo de vida do aplicativo e da assinatura for essencialmente o mesmo, mas é difícil garantir. Ao tornar a assinatura durável e fazer com que o publicador crie publicações persistentes, não haverá mensagens perdidas caso o gerenciador de filas ou o assinante seja finalizado prematuramente e precise ser recuperado.

Para aplicativos não JMS ou aplicativos JMS que não estão usando uma assinatura compartilhada, o gerenciador de filas abrirá implicitamente a fila de assinaturas gerenciadas duráveis para um assinante de tal forma que o processamento compartilhado da fila não seja possível. Além disso, a menos que seu aplicativo esteja usando assinaturas compartilhadas JMS, não é possível criar mais de uma assinatura para cada fila gerenciada e é possível que você ache as filas mais difíceis de gerenciar por ter menos controle sobre os nomes delas. Por esses motivos, considere se o assinante *não gerenciado* do MQ é mais adequado para aplicativos que requerem assinaturas duráveis do que o assinante *gerenciado* do MQ.

O código em Figura 76 na página 842 demonstra um padrão de assinatura durável não gerenciada. Para ilustração, o código também cria assinaturas não gerenciadas, não duráveis. Este exemplo ilustra as máscaras padrão a seguir:

- Assinaturas on demand: as sequências de tópicos de assinatura são dinâmicas. Elas são fornecidas pelo aplicativo quando ele é executado.
- Gerenciamento de tópicos de assinatura simplificado: o gerenciamento de tópicos de assinatura é simplificado definindo-se a parte raiz da sequência de tópicos de assinatura usando um tópico definido administrativamente. Isso oculta a parte raiz da árvore de tópicos do aplicativo. Ao ocultar a parte raiz, um assinante pode ser implementado em diferentes árvores de tópicos.
- Gerenciamento de assinatura flexível: é possível definir uma assinatura administrativamente ou criá-la on demand em um programa do assinante. Não há diferença entre assinaturas criadas administrativamente e programaticamente, exceto um atributo que mostra como a assinatura foi criada. Há um terceiro tipo de assinatura que é criado automaticamente pelo gerenciador de filas para distribuição de assinaturas. Todas as assinaturas são exibidas no IBM MQ Explorer.
- Associação flexível de assinaturas com filas: uma fila local predefinida é associada a uma assinatura pela função MQSUB. Há maneiras diferentes de usar MQSUB para associar assinaturas a filas:
  - Associe uma assinatura a uma fila que não tenha *nenhuma* assinatura existente, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Associe uma *nova* assinatura a uma fila que tenha assinaturas existentes, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Mova uma assinatura existente para uma fila diferente, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Continue uma assinatura existente associada a uma fila existente, `MQSO_RESUME + (Hobj = MQHO_NONE)` ou `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - Ao combinar `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` em diferentes combinações, é possível atender diferentes estados de entrada da assinatura e da fila sem precisar codificar várias versões do MQSUB com diferentes valores de `sd.Options`.
  - Como alternativa, codificando uma opção específica do `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER`, o gerenciador de filas retorna um erro (Tabela 123 na página 838) se os estados da assinatura e da fila fornecidos como entrada para MQSUB forem inconsistentes com o valor de `sd.Options`. Figura 82 na página 845 mostra os resultados de emitir MQSUB para a Assinatura X com diferentes configurações individuais da sinalização `sd.Options` e passá-lo para três identificadores de objetos diferentes.

  Explore entradas diferentes para o programa de exemplo em Figura 75 na página 841 para se familiarizar com esses tipos diferentes de erros. Um erro comum, `RC = 2440`, que não está incluído nos casos listados na tabela, é um erro de nome da assinatura. é comumente causado ao se passar um nome de assinatura nulo ou inválido com `MQSO_RESUME` ou `MQSO_ALTER`.
- Multiprocessamento: é possível compartilhar entre muitos consumidores o trabalho de leitura de publicações. Todas as publicações vão para a única fila associada ao tópico de assinatura. Consumidores têm a opção de abrir a fila diretamente usando MQOPEN ou continuar a assinatura usando MQSUB.
- Concentração de assinaturas: várias assinaturas podem ser criadas na mesma fila. Tenha cuidado com este recurso, pois ele pode à sobreposição de assinaturas e ao recebimento da mesma publicação várias vezes. A opção `MQSO_GROUP_SUB` elimina publicações duplicadas causadas pela sobreposição de assinaturas.
- Separação de assinante e consumidor: assim como os três modelos de consumidor ilustrados nos exemplos, outro modelo é separar o consumidor do assinante. É uma variação do MQ Subscriber não gerenciado, mas em vez de emitir MQOPEN e MQSUB no mesmo programa, um programa assina publicações e outro programa as consome. Por exemplo, o assinante pode fazer parte de um cluster de publicação/assinatura e o consumidor estar anexado a um gerenciador de filas fora do cluster de gerenciador de filas. O consumidor recebe publicações por meio do enfileiramento distribuído padrão definindo a fila de assinaturas como uma definição de fila remota.

Entender o comportamento de `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` é importante, principalmente se você planeja simplificar seu código usando combinações dessas opções. Estude a tabela Tabela 123 na página 838 que mostra os resultados de passar diferentes identificadores de filas

para MQSUB e os resultados de executar o programa de exemplo mostrado em Figura 77 na página 843 para Figura 82 na página 845.

O cenário usado para construir a tabela tem uma assinatura X e duas filas, A e B. O parâmetro do nome da assinatura sd.SubName é configurado como X, o nome de uma assinatura anexada na fila A. A fila B não tem assinatura anexada a ela.

Em Tabela 123 na página 838, MQSUB é transmitido para a assinatura X e o identificador de fila para a fila A. Os resultados das opções de assinatura são os seguintes:

- MQSO_CREATE falha porque o identificador de fila corresponde à fila A que já tem uma assinatura para X. Contraste esse comportamento para a chamada bem-sucedida. Aquela chamada é bem-sucedida porque a fila B não tem uma assinatura para X anexada a ela.
- MQSO_RESUME tem êxito porque o identificador de fila corresponde à fila A que já tem uma assinatura de X. Em contraste, a chamada falha onde a assinatura X não existe na fila A.
- MQSO_ALTER se comporta de maneira semelhante a MQSO_RESUME em relação a abrir a assinatura e a fila. No entanto, se os atributos contidos no descritor de assinatura passado para MQSUB forem diferentes dos atributos da assinatura, MQSO_RESUME falhará, enquanto MQSO_ALTER será bem-sucedido desde que a instância do programa tenha permissão para mudar os atributos. Observe que nunca é possível mudar a sequência de tópicos em uma assinatura; mas em vez de retornar um erro, MQSUB ignora o nome do tópico e os valores da sequência de tópicos no descritor de assinatura e usa os valores na assinatura existente.

Em seguida, analise Tabela 123 na página 838 em que MQSUB é transmitido por assinatura X e o identificador de fila para a fila B. Os resultados das opções de assinatura são os seguintes:

- MQSO_CREATE é bem-sucedido e cria a assinatura X na fila B porque esta é uma nova assinatura na fila B.
- MQSO_RESUME falha. MQSUB procura a assinatura X na fila B e não a localiza, mas em vez de retornar *RC = 2428 – assinatura X não existe*, retorna *RC = 2019 – A fila de assinaturas não corresponde à manipulação de objetos da fila*. O comportamento da terceira opção MQSO_ALTER sugere a razão para esse erro inesperado. MQSUB espera que o manipulador de filas aponte para uma fila com uma assinatura. Ele verifica isso primeiro antes de verificar se a assinatura denominada em sd.SubName existe.
- MQSO_ALTER é bem-sucedido e move a assinatura da fila A para a fila B.

Um caso que não é mostrado na tabela é se o nome da assinatura na fila A não corresponde ao nome da assinatura em sd.SubName. Essa chamada falha com *RC = 2428 – assinatura X não existe na Fila A*.

Tabela 123. Erros de MQSUB com diferentes identificadores de filas e combinações de assinaturas

Manipuladores de filas	**Fila A** **Assinatura X** **Fila B** **Nenhuma assinatura**	**Fila A** **Nenhuma assinatura** **Fila B** **Nenhuma assinatura**
Hobj para *Fila A* passado para MQSUB	**MQSO_CREATE** RC = 2432 – A assinatura X já existe na Fila A **MQSO_RESUME** Continua a assinatura X na Fila A **MQSO_ALTER** Continua a assinatura X na Fila A e faz alterações permitidas	**MQSO_CREATE** Cria a assinatura X na Fila A **MQSO_RESUME** RC = 2428 – A assinatura X não existe na Fila A **MQSO_ALTER** RC = 2428 – A assinatura X não existe na Fila A

Tabela 123. Erros de MQSUB com diferentes identificadores de filas e combinações de assinaturas (continuação)

Manipuladores de filas	**Fila A** **Assinatura X** **Fila B** **Nenhuma assinatura**	**Fila A** **Nenhuma assinatura** **Fila B** **Nenhuma assinatura**
Hobj para *Fila B* passado para MQSUB	**MQSO_CREATE** Cria nova assinatura X na Fila B **MQSO_RESUME** RC = 2019 – A fila de assinaturas não corresponde à manipulação de objetos de fila **MQSO_ALTER** Mova a assinatura X da Fila A para a Fila B	**MQSO_CREATE** Cria nova assinatura X na Fila B **MQSO_RESUME** RC = 2428 – a assinatura X não existe na Fila B **MQSO_ALTER** RC = 2428 – a assinatura X não existe na Fila B
MQHO_NONE passado para MQSUB	**MQSO_CREATE** RC = 2019 – Manipulação de objetos inválida: configure a sinalização MQSO_MANAGED para criar uma assinatura gerenciada e criar uma fila gerenciada **MQSO_RESUME** Continua a assinatura X na Fila A e retorna Hobj para a Fila A **MQSO_ALTER** Continua a assinatura X na Fila A, retorna Hobj para a Fila A e faz alterações permitidas	**MQSO_CREATE** RC = 2019 – Manipulação de objetos inválida: configure a sinalização MQSO_MANAGED para criar uma assinatura gerenciada e criar uma fila gerenciada **MQSO_RESUME** RC = 2428 – Nenhuma assinatura X **MQSO_ALTER** RC = 2019 – Manipulação de objetos inválida: nenhuma fila A ou B

Nota: O estilo de codificação compacto destina-se à capacidade de leitura e não ao uso na produção.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

Figura 74. Assinante do MQ não gerenciado – parte 1: declarações.

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

Figura 75. Assinante do MQ não gerenciado - parte 2: manipulação de parâmetros.

Comentários adicionais sobre a manipulação de parâmetros neste exemplo são os seguintes:

switch((argv[5][0]))
Você tem a opção de inserir A lter | C reate | R esume no parâmetro 5 para testar o efeito de substituir parte da configuração da opção MQSUB usada por padrão no exemplo. A configuração padrão usada pelo exemplo é MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Nota: Configurar MQSO_ALTER ou MQSO_RESUME sem configurar MQSO_DURABLE é um erro e sd.SubName deve ser configurado e fazer referência a uma assinatura que pode ser continuada ou alterada.

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
Se a fila de assinaturas padrão, STOCKTICKER, for substituída por uma sequência nula, então, contanto que MQSO_CREATE seja configurado, o exemplo configura a sinalização MQSO_MANAGED e cria uma fila de assinaturas dinâmicas. Se Alter or Resume forem configurados no quinto parâmetro, o comportamento do exemplo dependerá do valor de subscriptionName.

***subscriptionName = '\0';**
sdOptions = sdOptions - MQSO_DURABLE;

Se a assinatura padrão, IBMSTOCKPRICESUB, for substituída por uma sequência nula, o exemplo removerá o sinalizador MQSO_DURABLE. Se você executar o exemplo fornecendo os valores padrão para os outros parâmetros, uma assinatura temporária adicional destinada a STOCKTICKER será criada e receberá publicações duplicadas. Na próxima vez que executar o exemplo, sem nenhum parâmetro, você receberá somente uma publicação novamente.

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                   &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
&CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes            */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
&CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

Figura 76. Assinante do MQ não gerenciado – parte 3: corpo do código.

Comentários adicionais sobre o código neste exemplo são os seguintes:

if (strlen(subscriptionQueue))
Se não houver nenhum nome da fila de assinatura, o exemplo usará MQHO_NONE como o valor de Hobj.

MQOPEN(...);
A fila de assinatura é aberta e o identificador de fila é salvo em Hobj.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
A assinatura é aberta usando Hobj passado de MQOPEN (ou MQHO_NONE, se não houver nenhum nome da fila de assinatura). Uma fila não gerenciada pode ser continuada sem ser aberta explicitamente com um MQOPEN.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
A assinatura é fechada usando o identificador de assinatura. Dependendo de se a assinatura é durável ou não, a assinatura será fechada com um MQCO_KEEP_SUB ou MQCO_REMOVE_SUB implícito. É possível fechar uma assinatura durável com MQCO_REMOVE_SUB, mas não é possível fechar uma assinatura não durável com MQCO_KEEP_SUB. A ação de MQCO_REMOVE_SUB é remover a assinatura que para quaisquer publicações adicionais que estiverem sendo enviadas à fila de assinaturas.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
Nenhuma ação especial será tomada se a assinatura for não gerenciada. Se a fila for gerenciada e a assinatura fechada com um MQCO_REMOVE_SUB explícito ou implícito, todas as publicações serão eliminadas da fila e a fila será excluída neste ponto.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
Assegure-se de que as mensagens recebidas sejam aquelas para a nossa assinatura.

Resultados do exemplo ilustram aspectos de publicar/assinar:

Em Figura 77 na página 843, o exemplo inicia publicando 130 no tópico NYSE/IBM/PRICE.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Figura 77. Publicar 130 em NYSE/IBM/PRICE

Em Figura 78 na página 843, a execução do exemplo usando parâmetros padrão recebe a publicação retida 130. O objeto do tópico e a sequência de tópicos fornecidos são ignorados, conforme mostrado em Figura 82 na página 845. O objeto do tópico e a sequência de tópicos são sempre tomados do objeto de assinatura, quando um é fornecido, e a sequência de tópicos é imutável. O comportamento real do exemplo depende da opção ou combinação de MQSO_CREATE, MQSO_RESUME e MQSO_ALTER. Neste exemplo, MQSO_RESUME é a opção selecionada.

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

Figura 78. Receber a publicação retida

Em (Figura 79 na página 844), nenhuma publicação é recebida, porque a assinatura durável já recebeu a publicação retida. Neste exemplo, a assinatura é continuada fornecendo somente o nome da assinatura sem o nome da fila. Se o nome da fila foi fornecido, a fila poderá ser aberta primeiro e o identificador passado para MQSUB.

Nota: O erro 2038 de MQINQ é devido ao MQOPEN implícito de STOCKTICKER por MQSUB sem incluir a opção MQOO_INQUIRE. Evite o código de retorno 2038 de MQINQ abrindo a fila explicitamente.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

Figura 79. Continuar assinatura

No Figura 80 na página 844, o exemplo cria uma assinatura não gerenciada não durável usando STOCKTICKER como o destino. Como esta é uma nova assinatura, ela recebe a publicação retida.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

Figura 80. Receber publicação retida com a nova assinatura não durável não gerenciada

No Figura 81 na página 844, para demonstrar a sobreposição de assinaturas, outra publicação é enviada, mudando a publicação retida. Em seguida, uma nova assinatura não gerenciada, não durável é criada não fornecendo um nome de assinatura. A publicação retida é recebida duas vezes, uma para a nova assinatura e uma para a assinatura IBMSTOCKPRICESUB durável que ainda está ativa na fila STOCKTICKER. O exemplo é uma ilustração de que é a fila que tem assinaturas e não o aplicativo. Embora não fazendo referência à assinatura IBMSTOCKPRICESUB nesta chamada do aplicativo, o aplicativo recebe a publicação duas vezes: uma da assinatura durável que foi criada administrativamente e uma da assinatura não durável criada pelo próprio aplicativo.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

Figura 81. Assinaturas sobrepostas

No Figura 82 na página 845, o exemplo demonstra que fornecer uma nova sequência de tópicos e uma assinatura existente não resulta em uma assinatura mudada.

1. No primeiro caso, `Resume` continua a assinatura existente, como você poderia esperar e ignora a sequência de tópicos mudada.
2. No segundo caso, `Alter` causa um erro, `RC = 2510, Topic not alterable`.
3. No terceiro exemplo, `Create` causa um erro `RC = 2432, Sub already exists`.

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

Figura 82. Tópicos de Assinatura não podem ser mudados

Conceitos relacionados

“Exemplo 1: consumidor de publicação do MQ” na página 829
O consumidor da publicação MQ é um consumidor de mensagens IBM MQ que não assina tópicos em si.

“Exemplo 2: assinante do MQ gerenciado” na página 831
O assinante do MQ gerenciado é o padrão preferencial para a maioria dos aplicativos de assinante. Uma assinatura gerenciada é uma em que o IBM MQ manipula a assinatura e faz o registro e a remoção do registro para você. O exemplo *não* requer nenhuma definição administrativa de filas, tópicos ou assinaturas.

“Gravando aplicativos de publicador” na página 822
Comece a escrever aplicativos do publicador estudando dois exemplos. O primeiro é modelado o mais próximo possível em um aplicativo ponto a ponto colocando mensagens em uma fila e o segundo demonstra a criação de tópicos dinamicamente - um padrão mais comum para aplicativos publicadores.

Ciclos de vida de publicar/assinar

Considere os ciclos de vida de tópicos, assinaturas, assinantes, publicações, editores e filas ao projetar aplicativos de publicar/assinar.

O ciclo de vida de um objeto, como uma assinatura, começa com sua criação e termina com sua exclusão. Também pode incluir outros estados e mudanças pelos quais passa, como suspensão temporária, ter tópicos pais e filhos, expiração e exclusão.

Tradicionalmente, objetos do IBM MQ, como filas, são criados administrativamente ou por programas administrativos usando Formato de comando programável (PCF). Publicar/assinar é diferente, pois fornece os verbos de API `MQSUB` e `MQCLOSE` para criar e excluir assinaturas, tendo o conceito de assinaturas gerenciadas que não apenas cria e exclui filas, mas também limpa mensagens não consumidas e tendo associações entre objetos de tópicos criados administrativamente e sequências de tópicos criadas programaticamente ou administrativamente.

Essa riqueza funcional atende uma ampla variedade de requisitos de publicar/assinar e também simplifica o design de alguns padrões comuns de aplicativo de publicar/assinar. Assinaturas gerenciadas, por exemplo, simplificam tanto a programação e a administração de uma assinatura destinada a durar apenas o mesmo tempo que o programa que a criou. Assinaturas não gerenciadas simplificam a programação quando houver uma conexão mais livre entre assinatura e consumo de publicações. Assinaturas criadas centralmente são úteis quando o padrão for de roteamento de tráfego de publicação para consumidores com base em um modelo centralizado de controle, por exemplo, envio de informações de voo para portões automatizados, enquanto que as assinaturas programaticamente criadas podem ser usadas se a equipe do portão for responsável por assinar os registros de passageiros do voo, inserindo um número de voo em um portão.

Neste último exemplo, uma assinatura durável gerenciada pode ser apropriada: gerenciada, pois as assinaturas estão sendo criadas com muita frequência e possuem um terminal claro quando o portão se fecha e a assinatura pode ser removida programaticamente; durável, para evitar perder um registro de passageiro devido ao programa de assinantes do portão diminuir por uma razão ou outra[8]Para iniciar a publicação de registros de passageiros do portão, um possível design seria para o aplicativo do portão assinar tanto os registros do passageiro usando o número do portão quanto publicar o evento

[8] O publicar deve enviar os registros de passageiros como mensagens persistentes para evitar outras possíveis falhas, é claro.

de abertura do portão usando o número do portão. O publicador responde ao evento de abertura do portão publicando os registros de passageiros - que podem, então, também ir para outras partes interessadas, como faturamento, para registrar que o voo está ocorrendo e para atendimento ao cliente, para notificações de texto em celulares dos passageiros sobre o número do portão.

A assinatura gerenciada centralmente pode usar um modelo durável não gerenciado, roteando de listas de passageiros para o portão usando uma fila predefinida para cada portão.

Os três exemplos a seguir de ciclos de vida de publicar/assinar ilustram como assinantes gerenciados não duráveis, gerenciados duráveis e não gerenciados duráveis interagirem com assinaturas, tópicos, filas, publicadores e com o gerenciador de filas, e como as responsabilidades podem ser divididas entre a administração e os programas de assinante.

Assinante gerenciado não durável

Figura 83 na página 847 mostra um aplicativo criando uma assinatura gerenciada não durável, obtendo duas mensagens publicadas no tópico identificado na assinatura e finalizando. As interações com rótulo em uma fonte cinza em itálico com setas pontilhadas são implícitas.

Há alguns pontos a serem observados.

1. O aplicativo cria uma assinatura em um tópico para o qual já foi publicado duas vezes. Quando o assinante recebe sua primeira publicação, recebe a *segunda* publicação que é a publicação retida atualmente.
2. O gerenciador de filas cria uma fila de assinatura temporária, assim como cria uma assinatura para o tópico.
3. A assinatura tem uma expiração. Quando a assinatura expira, mais nenhuma publicação no tópico será enviada para essa assinatura, mas o assinante continua a obter mensagens publicadas antes da assinatura expirar. A expiração da publicação não é afetada pela expiração da assinatura.
4. A quarta publicação não é colocada na fila de assinatura e, consequentemente, o último `MQGET` não retorna uma publicação.
5. Embora o assinante fecha sua assinatura, ela não fecha sua conexão com a fila ou com o gerenciador de filas.
6. O gerenciador de filas é limpo pouco depois da finalização do aplicativo. Como a assinatura é gerenciada e não durável, a fila de assinaturas é excluída.

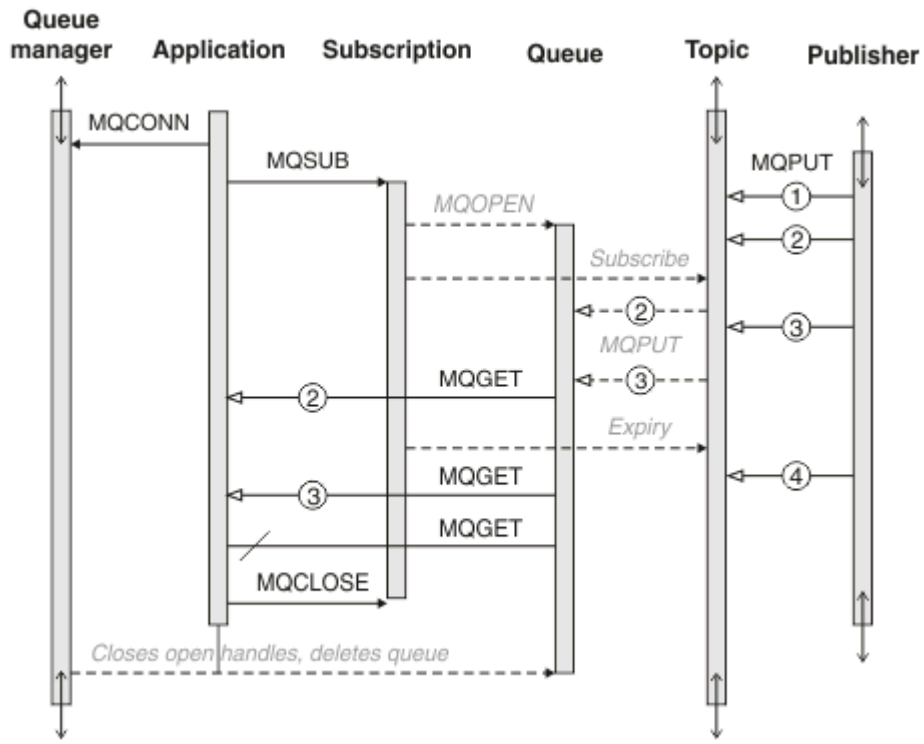


Figura 83. Linhas de vida do assinante gerenciado não durável

Assinante gerenciado durável

O assinante gerenciado durável leva o exemplo anterior um passo à frente e mostra uma assinatura gerenciada que sobrevive à finalização e à reinicialização do aplicativo de assinatura.

Há alguns novos pontos a serem observados.

1. Neste exemplo, diferentemente do anterior, o tópico de publicação não existia antes de ser definido na assinatura.
2. A primeira vez que o assinante é finalizado, ele fecha a assinatura com a opção `MQCO_KEEP_SUB`. Esse é o comportamento padrão para fechar implicitamente uma assinatura gerenciada durável.
3. Quando o assinante continua a assinatura, a fila de assinaturas é reaberta.
4. A nova publicação 2, colocada na fila antes de ser reaberta, está disponível para `MQGET`, mesmo após a assinatura ter sido removida.

   Embora a assinatura seja durável, o assinante recebe de forma confiável todas as mensagens enviadas pelo publicador somente se *ambos* forem verdadeiros, a assinatura durável e as mensagens persistentes. Persistência de mensagem depende da configuração do campo `Persistent` no `MQMD` da mensagem enviada pelo publicador. Um assinante não tem controle sobre isso.
5. Fechar a assinatura com a sinalização `MQCO_REMOVE_SUB` remove a assinatura, parando quaisquer publicações adicionais que estiverem sendo colocadas na fila de assinaturas. Quando a fila de assinatura é fechada, então, o gerenciador de filas remove a publicação 3 não lida e, em seguida, exclui a fila. A ação é equivalente a excluir a assinatura administrativamente.

   Nota: Não exclua a fila manualmente ou emita `MQCLOSE` com a opção `MQCO_DELETE` ou `MQCO_PURGE_DELETE`. Os detalhes visíveis da implementação de uma assinatura gerenciada não fazem parte da interface suportada do IBM MQ. O gerenciador de filas não pode gerenciar uma assinatura de forma confiável a menos que tenha controle completo.

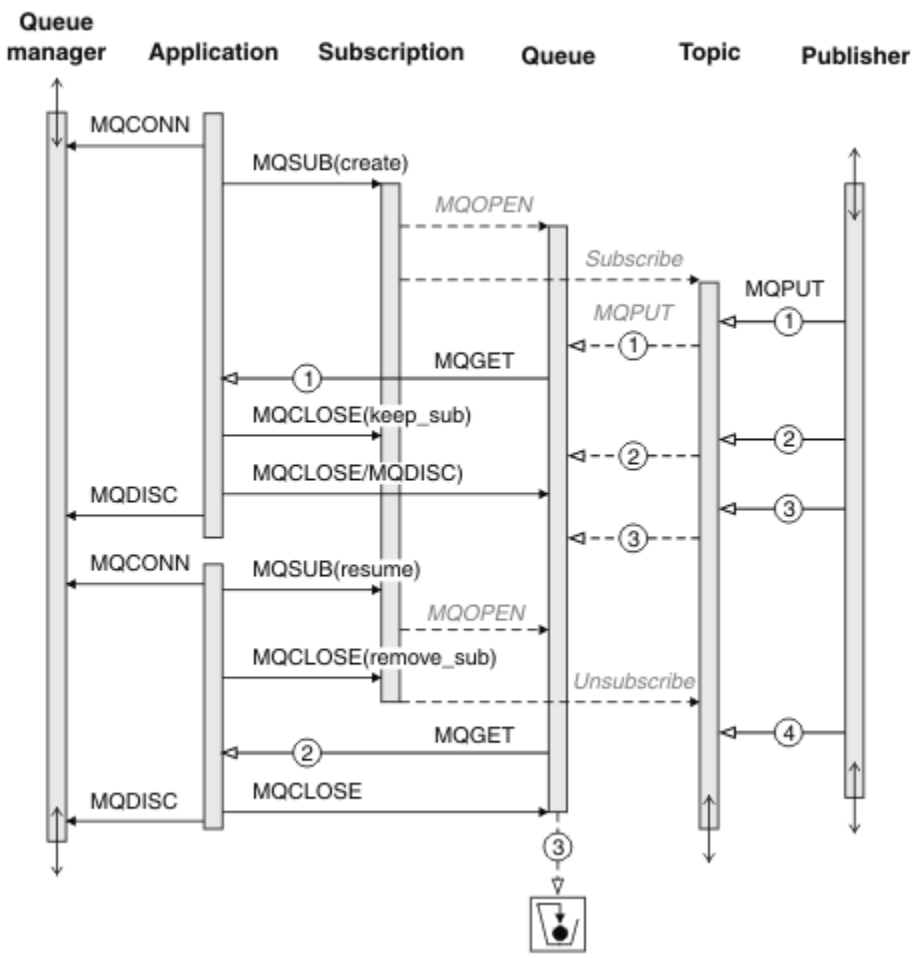


Figura 84. Linhas de vida do assinante gerenciado durável

Assinante não gerenciado durável

Um administrador é incluído no terceiro exemplo: o assinante não gerenciado durável. Este é um bom exemplo para mostrar como o administrador pode interagir com um aplicativo de publicar/assinar.

Os pontos a serem observados são listados.

1. O publicador coloca uma mensagem, 1, em um tópico que posteriormente se tornará associado ao objeto do tópico usado para assinatura. O objeto do tópico define uma sequência de tópicos que corresponde ao tópico ao qual foi publicado usando caracteres curinga.
2. O tópico possui uma publicação retida.
3. O administrador cria um objeto do tópico, uma fila e uma assinatura. O objeto do tópico e a fila precisam ser definidos antes da assinatura.
4. O aplicativo abre a fila associada à assinatura e passa a `MQSUB` o identificador da fila. Poderia, como alternativa, simplesmente abrir a assinatura, passando a ela o identificador de fila `MQHO_NONE`. O inverso não é verdadeiro, não pode continuar uma assinatura passando a ela somente o identificador de fila sem um nome de assinatura – uma fila pode ter várias assinaturas.
5. O aplicativo abre a assinatura usando a opção `MQSO_RESUME`, embora seja a primeira vez que abriu a assinatura. Está continuando uma assinatura criada administrativamente.
6. O assinante recebe a publicação retida, 1. A publicação 2, embora publicada antes que quaisquer publicações tenham sido recebidas pelo assinante, foi publicada após a assinatura ser iniciada e é a segunda publicação na fila de assinaturas.

Nota: Se a publicação retida não for publicada como uma mensagem persistente, então, será perdida após a reinicialização do gerenciador de filas.

7. Neste exemplo, a assinatura é durável. É possível que um programa crie uma assinatura não gerenciada durável; deve estar óbvio que isso não é algo que um administrador possa fazer.
8. O efeito da opção `MQCO_REMOVE_SUB` no fechamento da assinatura é remover a assinatura como se o administrador a tivesse excluído. Isso para quaisquer publicações adicionais que estiverem sendo enviadas para a fila, mas não afeta as publicações que já estão na fila, mesmo quando a fila for fechada, diferentemente de uma assinatura *gerenciada* durável.
9. O administrador posteriormente exclui a mensagem restante, 3, e exclui a fila.

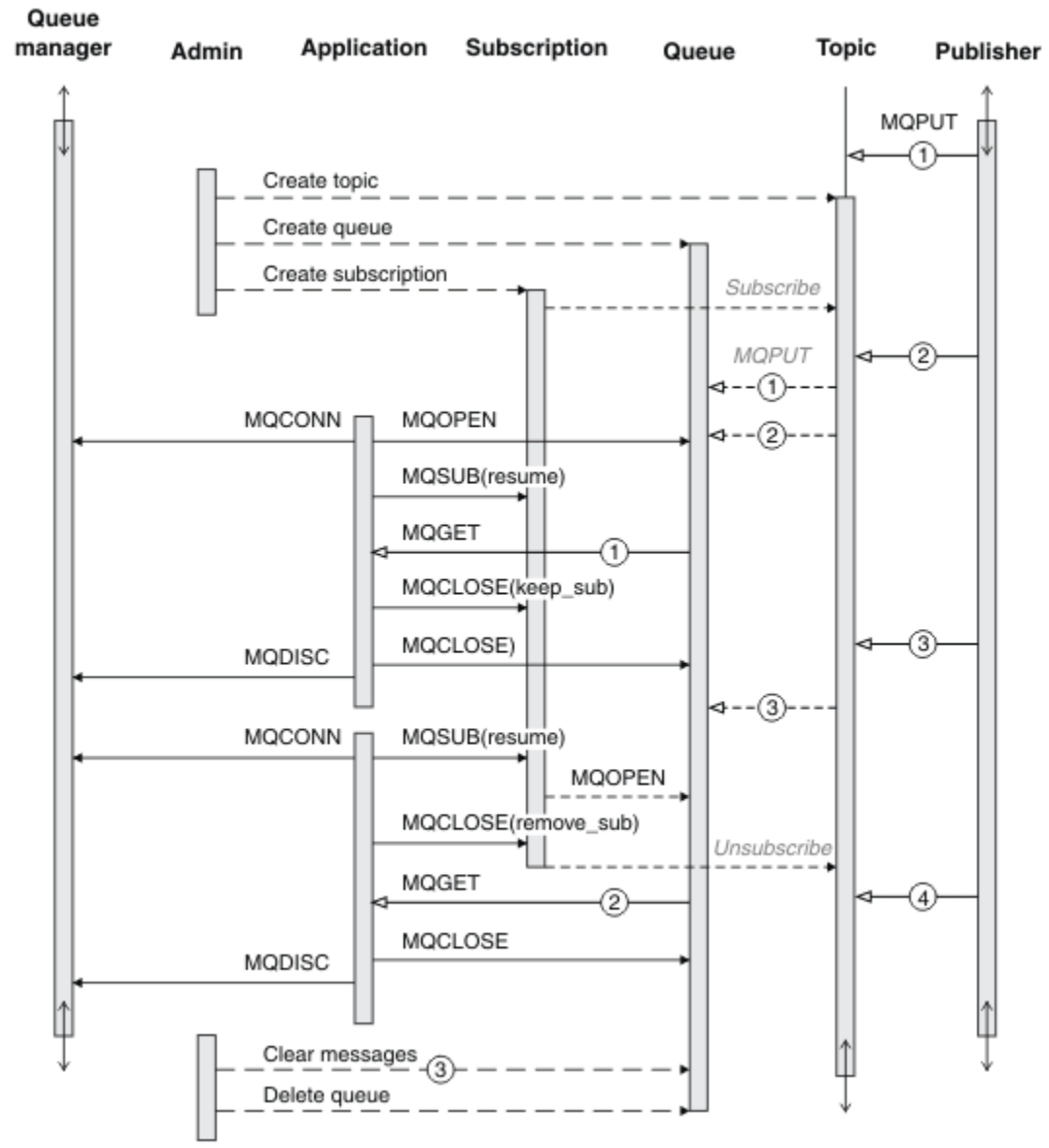


Figura 85. Linhas de vida do assinante não gerenciado durável

Um padrão normal para uma assinatura não gerenciada é a manutenção de filas e assinaturas ser executada pelo administrador. Geralmente, não se tentaria emular o comportamento de um assinante gerenciado e organizar as filas e assinaturas programaticamente no código do aplicativo. Se achar necessário escrever lógica de gerenciamento, questione se é possível obter os mesmos resultados usando um padrão gerenciado. Não é fácil escrever código de gerenciamento fortemente sincronizado e completamente confiável. É mais fácil organizar posteriormente, seja manualmente ou usando

um programa de gerenciamento automatizado, quando for possível ter certeza de que mensagens, assinaturas e filas podem ser simplesmente excluídas, independentemente de seu estado.

Propriedades de mensagem de publicação/assinatura

Várias propriedades de mensagens relacionadas ao sistema de mensagens de publicação/assinatura do IBM MQ.

PubAccountingToken

Esse é o valor que estará no campo AccountingToken do Message Descriptor (MQMD) de todas as mensagens de publicação que correspondam a essa assinatura. AccountingToken faz parte do contexto de identidade da mensagem. Para obter mais informações sobre o contexto da mensagem, consulte "Contexto da mensagem" na página 48. Para obter mais informações sobre o campo AccountingToken no MQMD, consulte AccountingToken.

PubApplIdentityData

Esse é o valor que estará no campo ApplIdentityData do Message Descriptor (MQMD) de todas as mensagens de publicação que correspondam a essa assinatura. ApplIdentityData faz parte do contexto de identidade da mensagem. Para obter mais informações sobre o contexto da mensagem, consulte "Contexto da mensagem" na página 48. Para obter mais informações sobre o campo ApplIdentityData no MQMD, consulte ApplIdentityData.

Se a opção MQSO_SET_IDENTITY_CONTEXT não for especificada, ApplIdentityData que será configurado em cada mensagem publicada para essa assinatura ficará em branco, como informações de contexto padrão.

Se a opção MQSO_SET_IDENTITY_CONTEXT for especificada, PubApplIdentityData está sendo gerado pelo usuário e esse campo é um campo de entrada que contém ApplIdentityData a ser configurado em cada publicação para essa assinatura.

PubPriority

Esse é o valor que estará no campo Prioridade do Message Descriptor (MQMD) de todas as mensagens de publicação que correspondam a essa assinatura. Para obter mais informações sobre o campo Prioridade no MQMD, consulte Prioridade.

O valor deve ser maior ou igual a zero; zero é a prioridade mais baixa. Os valores especiais a seguir também podem ser usados:

- MQPRI_PRIORITY_AS_Q_DEF – Quando uma fila de assinatura for fornecida no campo Hobj na chamada MQSUB e não for uma manipulação gerenciada, a prioridade da mensagem será obtida a partir do atributo DefProirity dessa fila. Se a fila assim identificada for uma fila de clusters ou houver mais de uma definição no caminho de resolução do nome da fila, a prioridade será determinada quando a mensagem de publicação for colocada na fila, conforme descrito para Prioridade no MQMD. Se a chamada MQSUB usar uma manipulação gerenciada, a prioridade para essa mensagem será obtida do atributo DefPriority da fila modelo associada ao tópico assinado.
- MQPRI_PRIORITY_AS_PUBLISHED – A prioridade para a mensagem é a prioridade da publicação original. Esse é o valor inicial desse campo.

SubCorrelId

Atenção: Um identificador de correlação só pode ser transmitido entre os gerenciadores de filas em um cluster de publicação/assinatura, não uma hierarquia.

Todas as publicações enviadas para corresponderem a essa assinatura conterão esse identificador de correlação no descritor de mensagens. Se múltiplas assinaturas usarem a mesma fila para obter suas publicações, usar MQGET por ID de correlação permitirá que somente publicações para uma assinatura específica sejam obtidas. Esse identificador de correlação pode ser gerado pelo gerenciador de filas ou pelo usuário.

Se a opção MQSO_SET_CORREL_ID não for especificada, o identificador de correlação será gerado pelo gerenciador de filas e esse campo será um campo de saída que contém o identificador de correlação que será configurado em cada mensagem publicada para essa assinatura.

Se a opção MQSO_SET_CORREL_ID for especificada, o identificador de correlação está sendo gerado pelo usuário e esse campo é um campo de entrada que contém o identificador de correlação a ser configurado em cada publicação para essa assinatura. Nesse caso, se o campo contiver MQCI_NONE, o identificador de correlação que será configurado em cada mensagem publicada para essa assinatura será o identificador de correlação criado pelo put original da mensagem.

Se a opção MQSO_GROUP_SUB for especificada e o identificador de correlação especificado for o mesmo que uma assinatura agrupada existente usando a mesma fila e uma sequência de tópicos sobreposta, somente a assinatura mais significativa do grupo será fornecida com uma cópia da publicação.

SubUserData

Esses são os dados do usuário de assinatura. Os dados fornecidos na assinatura nesse campo serão incluídos como a propriedade de mensagem MQSubUserData de cada publicação enviada para essa assinatura.

Propriedades de publicação

Tabela 124 na página 851 lista as propriedades de publicação que são fornecidas com uma mensagem de publicação.

É possível acessar essas propriedades diretamente da pasta **MQRFH2** ou recuperá-las usando MQINQMP. MQINQMP aceita o nome da propriedade ou o nome **MQRFH2** como o nome da propriedade a consultar.

Tabela 124. Propriedades de publicação

Nome da Propriedade	Nome do MQRFH2	tipo	Descrição
MQTopicString	mqps.Top	MQTYPE_STRING	Cadeia do tópico
MQSubUserData	mqps.Sud	MQTYPE_STRING	Dados do usuário do assinante
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	Publicação retida
MQPubOptions	mqps.Pub	MQTYPE_INT32	Opções de publicação
MQPubLevel	mqps.Pbl	MQTYPE_INT32	Nível de publicação
MQPubTime	mqpse.Pts	MQTYPE_STRING	Hora da publicação
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	Número de sequência da publicação
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	Dados de sequência/ número inteiro incluídos pelo publicador
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	Formato da mensagem: MQRFH1 MQRFH2 PCF

Ordenação de mensagens

Para um tópico específico, as mensagens são publicadas pelo gerenciador de filas na mesma ordem que são recebidas dos aplicativos de publicação (sujeitas à reorganização com base na prioridade da mensagem).

Ordenação de mensagens normalmente significa que cada assinante recebe mensagens de um gerenciador de filas específico, em um tópico específico, de um publicador específico na ordem que são publicadas por esse publicador.

No entanto, como com todas as mensagens do IBM MQ, é possível que, ocasionalmente, as mensagens sejam entregues fora de ordem. Isso pode ocorrer nas situações a seguir:

- Se um link na rede ficar inativo e as mensagens subsequentes forem roteadas novamente por outro link
- Se uma fila ficar temporariamente cheia ou inibida para put, de forma que uma mensagem seja colocada em uma fila de mensagens não entregues e, portanto, atrasada, enquanto as mensagens subsequentes passam direto.
- Se o administrador excluir um gerenciador de filas quando publicadores e assinantes ainda estiverem em operação, fazendo com que mensagens enfileiradas sejam colocadas na fila de mensagens não entregues e assinaturas sejam interrompidas.

Se estas circunstâncias não puderem ocorrer, as publicações sempre serão entregues em ordem.

Nota: Não é possível usar mensagens agrupadas ou segmentadas com Publicar/Assinar.

Interceptando publicações

É possível interceptar uma publicação, modificá-la e, em seguida, publicá-la novamente antes de atingir qualquer outro assinante.

Pode ser que deseje interceptar uma publicação antes de atingir um assinante para executar uma das ações a seguir:

- Anexar informações adicionais à mensagem
- Bloquear a mensagem
- Transformar a mensagem

É possível executar a mesma operação em cada mensagem ou variar a operação, dependendo da assinatura, da mensagem ou do cabeçalho da mensagem.

Referências relacionadas

MQ_PUBLISH_EXIT - saída Publish

Níveis de assinatura

Configure o nível de uma assinatura para interceptar uma publicação antes de atingir seus assinantes finais. Um assinante de interceptação assina em um nível de assinatura superior e publica novamente em um nível de publicação inferior. Construa uma cadeia de assinantes de interceptação para executar o processamento de mensagens em uma publicação antes que seja entregue aos assinantes finais.

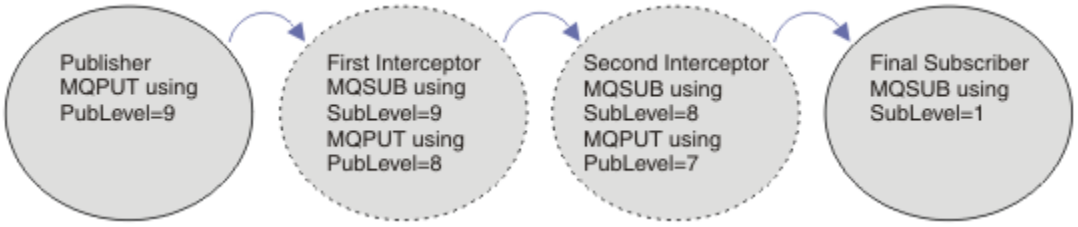


Figura 86. Sequência de assinantes de interceptação

Para interceptar uma publicação, use o atributo **MQSD** `SubLevel`. Após uma mensagem ter sido interceptada, ela pode ser transformada e republicada em uma publicação de nível inferior, mudando o atributo **MQPMO** `PubLevel`. A mensagem então vai para os assinantes finais ou é interceptada novamente por um assinante intermediário no nível de assinatura inferior.

O assinante de interceptação geralmente transforma uma mensagem antes de publicá-la novamente. Uma sequência de assinantes de interceptação forma um fluxo de mensagens. Como alternativa, você não pode publicar novamente a publicação interceptada: assinantes em níveis inferiores de assinatura não receberiam a mensagem.

Assegure que o interceptador receba as publicações antes de quaisquer outros assinantes. Configure o nível de assinatura do interceptador para nível superior a dos outros assinantes. Por padrão, os assinantes têm um Subnível de 1. O valor mais alto é 9. Uma publicação deve começar com um PubLevel pelo menos tão alto quanto o mais alto SubLevel. Publique inicialmente com o PubLevel padrão de 9.

- Se você tiver um assinante de interceptação em um tópico, configure o SubLevel para 9.
- Para vários aplicativos de interceptação em um tópico, configure um SubLevel inferior para cada assinante de interceptação sucessivo.
- É possível implementar no máximo 8 aplicativos de interceptação, com níveis de assinatura de 9 a 2, inclusive. O destinatário final da mensagem tem um SubLevel iguala a 1.

O interceptador com o nível de assinatura mais alto que é igual ou menor que o PubLevel da publicação recebe a primeira publicação. Configure somente um assinante de interceptação para um tópico em um determinado nível de assinatura. Ter vários assinantes em um nível de assinatura específico resulta em várias cópias da publicação serem enviadas ao conjunto final de aplicativos de assinatura.

Um assinante com um SubLevel igual a 0 é usado como um depósito. Ele recebe a publicação se nenhum assinante final obtiver a mensagem. Um assinante com SubLevel igual a 0 pode ser usado para monitorar as publicações que nenhum outro assinante recebeu.

Programando um assinante de interceptação

Use as opções de assinatura descritas em Tabela 125 na página 853.

Tabela 125. Opções de assinatura para assinantes de interceptação

Opção de assinatura	**Notas**
MQSO_SET_CORREL_ID e SubCorrelId configurados para MQCI_NONE	Mantenha o CorrelId da publicação interceptada igual ao da publicação original. **Nota:** Não é possível passar o identificador de correlação de uma publicação em uma hierarquia. O campo é usado pelo gerenciador de filas.
PubPriority configurado para MQPRI_PRIORITY_AS_PUBLISHED	Mantenha a prioridade da publicação interceptada igual à da publicação original.

As opções em Tabela 125 na página 853 devem ser usadas por todos os assinantes de interceptação. O resultado é que o identificador de correlação e a prioridade da mensagem não são modificados a partir da configuração do publicador original.

Quando o assinante de interceptação tiver processado a publicação, ele publica novamente a mensagem para o mesmo tópico em um PubLevel um nível inferior ao SubLevel de sua própria assinatura. Se o assinante de interceptação tiver configurado um SubLevel igual a 9, ele publica novamente a mensagem com um PubLevel igual a 8.

Para publicar novamente a mensagem corretamente, várias partes de informações da publicação original são necessárias. Reutilize o mesmo **MQMD** que da mensagem original e configure MQPMO_PASS_ALL_CONTEXT para assegurar que todas as informações no **MQMD** sejam passadas ao próximo assinante. Copie os valores das propriedades da mensagem mostrados em Tabela 126 na página 854 nos campos correspondentes da mensagem publicada novamente. O assinante de interceptação pode mudar esses valores. Use o operador OR para incluir valores adicionais no **MQPMO**. O campo Options para combinar as opções de mensagem put.

Deve-se abrir a fila de publicação explicitamente em vez de usar uma fila de publicação gerenciada. Não é possível configurar MQSO_SET_CORREL_ID para uma fila gerenciada. Também não é possível configurar MQOO_SAVE_ALL_CONTEXT em uma fila gerenciada. Consulte os fragmentos de código listados em “Exemplos” na página 854.

Tabela 126. Valores de MQPUT para mensagens publicadas novamente	
Publicar mensagem novamente usando MQPUT	**Informações na mensagem de publicação**
MQOD. ObjectString	Propriedade da mensagem MQTopicString
MQPMO. Options	Propriedade da mensagem MQPubOptions

O assinante final tem a opção de configurar suas opções de assinatura de forma diferente.
Por exemplo, pode configurar a prioridade da publicação explicitamente em vez de para MQPRI_PRIORITY_AS_PUBLISHED. As configurações de um assinante final afetam somente publicação do assinante de interceptação final na cadeia.

Publicações Retidas

Uma publicação retida deve ser preservada após ter sido interceptada, copiando as opções put-message originais na mensagem publicada novamente.

A opção MQPMO_RETAIN é configurada pelo publicador. Cada assinante de interceptação deve transferir o MQPubOptions para as opções put-message da mensagem publicada novamente conforme mostrado em Tabela 126 na página 854. Copiar as opções put-message preserva as opções configuradas pelo publicador original, incluindo se deseja reter a publicação.

Quando uma publicação conclui sua passagem pela cadeia de assinantes de interceptação e é entregue a assinantes finais, é finalmente retida. Novos assinantes, no SubLevel 1, que solicitam a publicação retida, recebem a mesma sem qualquer interceptação adicional. Os assinantes em um SubLevel maior que 1 não são enviadas à publicação retida. Como resultado, a publicação retida não é modificada pela cadeia de assinantes de interceptação uma segunda vez.

Exemplos

Os exemplos são fragmentos de código que podem ser combinados para construir um assinante de interceptação. O código é escrito para ser breve, em vez da qualidade da produção.

As diretivas do pré-processador em Figura 87 na página 854 definem as duas propriedades a serem extraídas a partir das mensagens da publicação necessárias pela chamada MQI MQINQMP.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS        (MQPTR)(char*) "MQPubOptions",\
                                 0,\
                                 12,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
#define      MQTOPICSTRING       (MQPTR)(char*) "MQTopicString",\
                                 0,\
                                 13,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
```

Figura 87. Diretivas do pré-processador

Figura 88 na página 855 lista as declarações usadas nos fragmentos de código. Exceto para os termos destacados, as declarações são padrão para um aplicativo IBM MQ.

As opções Put e Get destacadas são inicializadas para passar todo o contexto. MQTOPICSTRING e MQPUBOPTIONS destacados são inicializadores de MQCHARV para nomes de propriedades definidos nas diretivas do pré-processador. Os nomes são passados para MQINQMP.

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

Figura 88. Declarações

Inicializações que não são facilmente executadas nas declarações são mostradas em Figura 89 na página 856. Os valores destacados requerem explicação.

SYSTEM.NDURABLE.MODEL.QUEUE
Neste exemplo, em vez de usar MQSUB para abrir uma assinatura não durável gerenciada, a fila modelo, SYSTEM.NDURABLE.MODEL.QUEUE, é usada para criar uma fila dinâmica temporária. Seu identificador é passado para MQSUB. Ao abrir a fila diretamente, você é capaz de salvar todo o contexto da mensagem e configurar a opção de assinatura, MQSO_SET_CORREL_ID.

MQGMO_CURRENT_VERSION
É importante usar a versão atual da maioria das estruturas do IBM MQ. Os campos, como gmo.MsgHandle, estão disponíveis somente na versão mais recente das estruturas de controle.

MQGMO_PROPERTIES_IN_HANDLE
As opções da sequência de tópicos e put message configuradas na publicação original devem ser recuperadas pelo assinante de interceptação usando as propriedades de mensagem. Uma alternativa seria ler a estrutura **MQRFH2** na mensagem diretamente.

MQSO_SET_CORREL_ID
Use MQSO_SET_CORREL_ID em combinação com

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

O efeito dessas opções é passar adiante o identificador de correlação. O identificador de correlação configurado pelo publicador original é colocado no campo do identificador de correlação da publicação recebida pelo assinante de interceptação. Cada assinante de interceptação passa adiante o mesmo identificador de correlação. O assinante final tem, então, a opção de receber o mesmo identificador de correlação.

Nota: Se a publicação for passada por meio de uma hierarquia de publicar/assinar, o identificador de correlação nunca será retido.

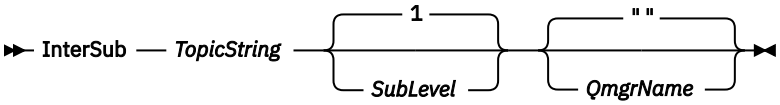
MQPRI_PRIORITY_AS_PUBLISHED
A publicação é colocada na fila de publicação com a mesma prioridade de mensagem que foi publicada.

```
strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version                = MQGMO_VERSION_4;
gmo.Options                = MQGMO_WAIT
                           | MQGMO_PROPERTIES_IN_HANDLE
                           | MQGMO_CONVERT;
gmo.WaitInterval           = 30000;
sd.Options                 = MQSO_CREATE
                           | MQSO_FAIL_IF_QUIESCING
                           | MQSO_SET_CORREL_ID;
sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version                 = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType             = MQOT_TOPIC;
putOD.ObjectString.VSPtr     = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
putOD.Version                = MQOD_VERSION_4;
pmo.Version                = MQPMO_VERSION_3;
```

Figura 89. Inicializações

Figura 90 na página 857 mostra o fragmento de código para ler os parâmetros da linha de comandos, concluir a inicialização e criar a assinatura de interceptação.

Execute o programa com o comando:



Para tornar a manipulação de erros o mais não obstrutiva possível, o código de razão de cada chamada MQI é armazenado em um elemento de matriz diferente. Após cada chamada, o código de conclusão é testado e, se o valor for MQCC_FAIL, o controle sai do bloco de código do `{ } while(0)`.

As duas linhas de código que valem a pena ser observadas são:

pmo.PubLevel = sd.SubLevel - 1;
Configura o nível de publicação da mensagem publicada novamente para um inferior ao nível de assinatura do assinante de interceptação.

gmo.MsgHandle = Hmsg;
Fornece um identificador de mensagem para MQGET para retornar as propriedades da mensagem.

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

Figura 90. Preparando para interceptar publicações

O fragmento de código principal, Figura 91 na página 858, recebe mensagens da fila de publicação. Ele consulta as propriedades de mensagem e publica as mensagens novamente usando a sequência de tópicos e o **MQPMO** original. Propriedades de `option` da publicação.

Neste exemplo, nenhuma transformação é executada na publicação. A sequência de tópicos da publicação publicada novamente sempre corresponde à sequência de tópicos que o assinante de interceptação assinou. Se o assinante de interceptação for responsável por interceptar várias assinaturas enviadas para a mesma fila de publicação, pode ser necessário consultar a sequência de tópicos para distinguir as publicações que correspondem a assinaturas diferentes.

As chamadas para `MQINQMP` estão destacadas. As propriedades das opções de sequência de tópicos e put message de publicação são gravadas diretamente nas estruturas de controle de saída. A única razão para alterar o campo de comprimento MQCHARV de `putOD.ObjectString` de um comprimento explícito para uma sequência finalizada em nulo é para usar `printf` para saída da sequência.

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

Figura 91. Interceptar publicação e publicar novamente

O fragmento de código final é mostrado em Figura 92 na página 858.

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

Figura 92. Conclusão

Interceptando publicações e publicar/assinar distribuído
Siga um padrão simples ao implementar a intercepção de assinantes ou saídas Publish para uma topologia distribuída de publicar/assinar. Implemente a interceptação de assinantes nos mesmos gerenciadores de filas que publicadores e saídas Publish nos mesmos gerenciadores de filas que assinantes finais.

Figura 93 na página 859 mostra dois gerenciadores de filas conectados em um cluster de assinatura de publicação. Um publicador cria uma publicação para um tópico de cluster no nível de publicação 9. As setas numeradas mostram a sequência de etapas tomadas pela publicação à medida que ela flui para os assinantes do tópico do cluster. A publicação é interceptada pelo assinante com `Sublevel` 9 e republicada com `Publevel` 8. Ela é interceptada novamente por um assinante no `Sublevel` 8. O assinante republica no `Publevel` 7. O assinante do proxy fornecido pelo gerenciador de filas encaminha a publicação para o gerenciador de filas B, em que uma saída Publish foi implementada além de um assinante final. A publicação é processada pela saída Publish antes de ser finalmente recebida pelo assinante final no `Sublevel` 1. Os assinantes de interceptação e a saída Publish são mostrados com as linhas de saída quebradas.

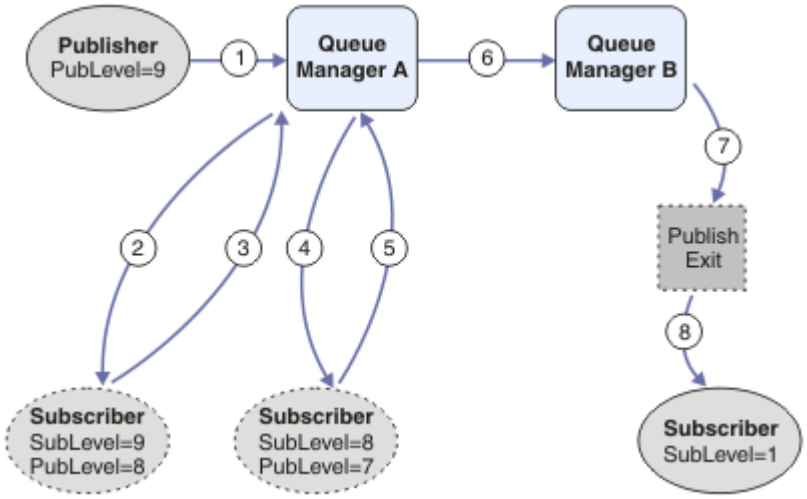


Figura 93. Interceptação e saída Publish em um cluster

O objetivo do padrão simples é para cada assinante receber uma publicação para receber a publicação idêntica. A publicação percorre a mesma sequência de transformações independentemente de onde o assinante está conectado. Provavelmente, você deseja evitar ter a sequência de transformações variando, dependendo de onde os publicadores ou assinantes finais estão conectados. Uma exceção razoável seria customizar a publicação definitivamente entregue a cada assinante individual. Use a saída Publish para customizar o aplicativo com base na fila na qual a publicação é finalmente entregue.

Deve-se considerar cuidadosamente onde implementar assinantes de interceptação e saídas Publish em uma topologia distribuída de publicar/assinar. O padrão direto implementa assinantes de interceptação no mesmo gerenciador de filas que os publicadores e saídas Publish nos mesmos gerenciadores de filas que os assinantes finais.

Antipadrão

Figura 94 na página 860 mostra como as coisas podem ficar distorcidas, se você não seguir um padrão simples. Para complicar a implementação, um assinante final é incluído no gerenciador de filas A e dois assinantes de interceptação adicionais são incluídos no gerenciador de filas B.

A publicação é encaminhada para o gerenciador de filas B no `PubLevel` 7, em que ela é interceptada por um assinante no `subnível` 5 antes de ser consumida pelo assinante final no `subnível` 1. A saída Publish intercepta a publicação antes de ela ser transmitida tanto para o consumidor interceptador quanto para o consumidor final no gerenciador de filas B. A publicação chega ao assinante final no gerenciador de filas A sem ser processada pela saída Publish.

Em uma topologia de publicar/assinar, os assinantes proxy assinam no `SubLevel` 1 e passam no `PubLevel` configurado pelo último assinante de interceptação. Em Figura 94 na página 860, o resultado é que a publicação não é interceptada pelo assinante usando `SubLevel` 9 no gerenciador de filas B.

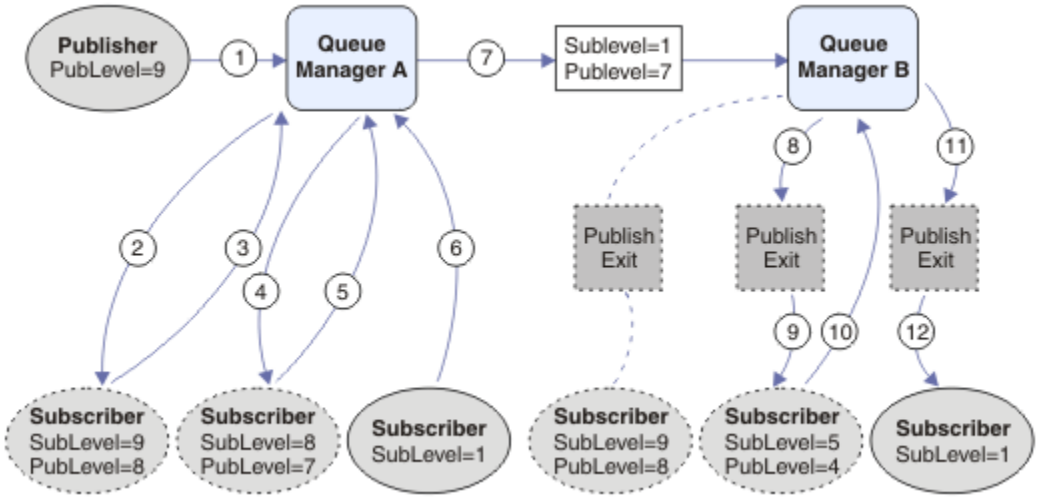


Figura 94. Implementação complexa de assinantes de interceptação

Opções de publicação

Estão disponíveis várias opções que controlam a maneira como as mensagens são publicadas.

Retendo informações de respostas de assinantes

Se não desejar que assinantes possam responder a publicações que recebem, é possível reter as informações nos campos ReplyToQ e ReplyToQmgr de MQMD usando a opção de put-message MQPMO_SUPPRESS_REPLYTO. Se essa opção for usada, o gerenciador de filas remove essas informações do MQMD quando ele recebe a publicação antes de encaminhá-la a todos os assinantes.

Essa opção não pode ser usada em combinação com uma opção de relatório que precisa de um ReplyToQ. Se isso for tentado, a chamada falhará com MQRC_MISSING_REPLY_TO_Q.

Nível de publicação

Usar níveis de publicação é uma maneira de controlar quais assinantes recebem a publicação. O nível de publicação indica o nível de assinatura almejado pela publicação. Somente assinaturas com o nível de assinatura mais alto menor ou igual ao nível da publicação receberão a publicação. Esse valor deve estar no intervalo de zero a nove; zero é o nível de publicação mais baixo. O valor inicial deste campo é de 9. Um dos usos dos níveis de publicação e de assinatura é para interceptar publicações.

Verificando se uma publicação não foi entregue a algum assinante

Para verificar se uma publicação não tiver sido entregue a todos os assinantes, use a opção de put-message MQPMO_WARN_IF_NO_SUBS_MATCHED com a chamada MQPUT. Se um código de conclusão MQCC_WARNING e um código de razão MQRC_NO_SUBS_MATCHED forem retornados pela operação put, a publicação não foi entregue para nenhuma assinaturas. Se a opção MQPMO_RETAIN for especificada na operação put, a mensagem será retida e entregue a qualquer assinatura correspondente definida subsequentemente. Em um sistema distribuído de publicar/assinar, o código de razão MQRC_NO_SUBS_MATCHED será retornado somente se não houver nenhuma assinatura de proxy registrado para o tópico no gerenciador de filas.

Opções de Assinatura

Estão disponíveis várias opções que controlam a maneira como as assinaturas de mensagens são manipuladas.

Persistência de mensagem

Gerenciadores de filas mantêm a persistência das publicações que encaminham aos assinantes conforme configurado pelo publicador. O publicador configura a persistência para uma das opções a seguir:

0
Não persistente

1
Persistente

2
Persistência como definição de fila/tópico

Para publicar/assinar, o publicador resolve o objeto do tópico e **topicString** para um objeto do tópico resolvido. Se o publicador especificar Persistência como definição de fila/tópico, então, a persistência padrão do objeto do tópico resolvido será configurada para a publicação.

Publicações Retidas

Para controlar quando as publicações retidas são recebidas, os assinantes podem usar duas opções de assinatura:

Publicar somente sob solicitação, MQSO_PUBLICATIONS_ON_REQUEST

Se desejar que um assinante tenha controle de quando recebe publicações, será possível usar a opção de assinatura MQSO_PUBLICATIONS_ON_REQUEST. Um assinante pode, então, controlar quando recebe publicações usando a chamada MQSUBRQ (especificando o identificador Hsub que foi retornado da chamada MQSUB original) para solicitar que seja enviada a ele uma publicação retida do tópico. Os assinantes que usam a opção de assinatura MQSO_PUBLICATIONS_ON_REQUEST não recebem nenhuma não retida.

Se você especificar MQSO_PUBLICATIONS_ON_REQUEST, deve-se usar MQSUBRQ para recuperar qualquer publicação. Se não usar MQSO_PUBLICATIONS_ON_REQUEST, obterá as mensagens conforme elas forem publicadas.

Se um assinante usar a chamada MQSUBRQ e usar curingas no tópico da assinatura, a assinatura poderá corresponder a vários tópicos ou nós em uma árvore de tópicos, todos com mensagens retidas (se houver alguma) serão enviados ao assinante.

Essa opção pode ser útil principalmente quando usada com assinaturas duráveis porque um gerenciador de filas continuará a enviar publicações a um assinante se tiver assinado de forma durável mesmo que esse aplicativo de assinante não esteja em execução. Isso pode levar a um acúmulo de mensagens na fila de assinantes. Esse acúmulo pode ser evitado se o assinante se registrar usando a opção MQSO_PUBLICATIONS_ON_REQUEST. Como alternativa, será possível usar assinaturas não duráveis, se apropriado para seu aplicativo, para evitar um acúmulo de mensagens indesejadas.

Se uma assinatura for durável e um publicador usar publicações retidas, o aplicativo de assinante poderá usar a chamada MQSUBRQ para atualizar suas informações de estado após uma reinicialização. O assinante deve, então, atualizar seu estado periodicamente usando a chamada MQSUBRQ.

Nenhuma publicação será enviada como resultado da chamada MQSUB usando essa opção. Uma assinatura durável continuada após desconexão usará a opção MQSO_PUBLICATIONS_ON_REQUEST se a assinatura original tiver sido configurada para usar esta opção.

Somente novas publicações, MQSO_NEW_PUBLICATIONS_ONLY

Se existir uma publicação retida em um tópico, todos os assinantes que fizerem uma assinatura após a publicação receberão uma cópia dessa publicação. Se um assinante não desejar receber as publicações que foram feitas antes da assinatura que está sendo feita, ele poderá usar a opção de assinatura MQSO_NEW_PUBLICATIONS_ONLY.

Agrupando assinaturas

Considere o agrupamento de assinaturas se você tiver configurado uma fila para receber publicações e tiver várias assinaturas sobrepostas alimentando publicações para a mesma fila. Essa situação é semelhante ao exemplo em Assinaturas sobrepostas.

É possível evitar receber publicações duplicadas configurando a opção MQSO_GROUP_SUB ao assinar um tópico. O resultado é que quando mais de uma assinatura no grupo corresponde ao tópico de uma publicação, somente uma assinatura é responsável por colocar a publicação na fila. As outras assinaturas correspondentes ao tópico de publicação são ignoradas.

A assinatura responsável por colocar a publicação na fila é escolhida com base no fato de ter a sequência de tópicos correspondente mais longa, antes de encontrar quaisquer curingas. Ela pode ser considerada como a assinatura correspondente mais próxima. Suas propriedades são propagadas para a publicação, inclusive se tiver a propriedade MQSO_NOT_OWN_PUBS. Se ocorrer, nenhuma publicação será entregue à fila, mesmo que outras assinaturas correspondentes possam não ter a propriedade MQSO_NOT_OWN_PUBS.

Não é possível colocar todas as suas assinaturas em um único grupo para eliminar publicações duplicadas. As assinaturas agrupadas devem atender as condições a seguir:

1. Nenhuma das assinaturas é gerenciada.
2. Um grupo de assinaturas entrega publicações para a mesma fila.
3. Cada assinatura deve estar no mesmo nível de assinatura.
4. A mensagem de publicação para cada assinatura no grupo tem o mesmo identificador de correlação.

   Para assegurar que cada assinatura resulte em uma mensagem de publicação com o mesmo identificador de correlação, configure MQSO_SET_CORREL_ID para criar seu próprio identificador de correlação na publicação e configure o mesmo valor no campo **SubCorrelId** em cada assinatura. Não configure **SubCorrelId** para o valor MQCI_NONE.

Consulte o ../refdev/q100080_.dita#q100080_/mqso_group_sub para obter informações adicionais.

Consultando e configurando atributos de objeto

Atributos são as propriedades que definem as características de um objeto do IBM MQ.

Elas afetam a maneira que um gerenciador de filas processa um objeto. Os atributos de cada tipo de objeto IBM MQ são descritos em detalhes em Atributos de objetos.

Alguns atributos são configurados quando o objeto é definido e podem ser mudados apenas usando os comandos do IBM MQ; um exemplo de tal atributo é a prioridade padrão para mensagens colocadas em uma fila. Outros atributos são afetados pela operação do gerenciador de filas e podem mudar com o tempo; um exemplo é a profundidade atual de uma fila.

É possível questionar sobre os valores atuais da maioria dos atributos usando a chamada MQINQ. O MQI também fornece uma chamada MQSET com a qual é possível mudar alguns atributos da fila. Não é possível usar as chamadas MQI para mudar os atributos de qualquer outro tipo de objeto. Em vez disso, deve-se usar um dos recursos a seguir:

- ALW O recurso do MQSC, que é descrito em Comandos do MQSC.
- IBM i Os comandos CL CHGMQMx, que são descritos em referência de comandos para IBM i, ou o recurso MQSC.
- z/OS Os comandos do operador ALTER ou os comandos DEFINE com a opção REPLACE, que são descritos em Comandos MQSC.

Nota: Os nomes dos atributos de objetos são mostrados nesta documentação no formulário que você usa com as chamadas MQINQ e MQSET. Ao usar comandos do IBM MQ para definir, alterar ou exibir os atributos, deve-se identificar os atributos usando as palavras-chave mostradas nas descrições dos comandos nos links dos tópicos.

Ambas chamadas MQINQ e MQSET usam matrizes de seletores para identificar os atributos que você deseja pesquisar sobre ou configurar. Há um seletor para cada atributo com que é possível trabalhar. O nome do seletor possui um prefixo determinado pela natureza do atributo:

Tabela 127. Prefixos para nomes de seletor	
Prefixo	**Descrição**
MQCA_	Esses seletores referem-se a atributos que contêm dados de caractere (por exemplo, o nome de uma fila).
MQIA_	Esses seletores se referem a atributos que contêm tanto valores numéricos (como *CurrentQueueDepth*, o número de mensagens em uma fila) ou um valor constante (como *SyncPoint*, se o gerenciador de filas suportar sincronização).

Antes de usar as chamadas MQINQ ou MQSET, o seu aplicativo deve estar conectado ao gerenciador de filas e deve-se usar a chamada MQOPEN para abrir o objeto para configurar ou consultar sobre atributos. Essas operações estão descritas em "Conectando-se e desconectando-se de um gerenciador de filas" na página 747 e "Abrindo e fechando objetos" na página 754.

Use os seguintes links para descobrir mais sobre consultar e configurar atributos do objeto:

- "Consultando sobre os atributos de um objeto" na página 863
- "Alguns casos em que a chamada MQINQ falha" na página 864
- "Configurando os atributos da fila" na página 865

Conceitos relacionados

"Visão geral da Message Queue Interface" na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

"Conectando-se e desconectando-se de um gerenciador de filas" na página 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

"Abrindo e fechando objetos" na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

"Colocando mensagens em uma fila" na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" na página 780
Use estas informações para aprender como obter mensagens de uma fila.

"Confirmando e fazendo backup de unidades de trabalho" na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

"Iniciando aplicativos IBM MQ usando acionadores" na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"Using and writing applications on IBM MQ for z/OS" na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

"IMS and IMS bridge applications on IBM MQ for z/OS" na página 71
This information helps you to write IMS applications using IBM MQ.

Consultando sobre os atributos de um objeto

Use a chamada MQINQ para consultar sobre os atributos de qualquer tipo de IBM MQ.

Como entrada para essa chamada, deve-se fornecer:

- Uma manipulação de conexões.
- Uma manipulação de objetos.
- O número de seletores.
- Uma matriz de seletores de atributos, cada seletor tendo o formulário MQCA_* ou MQIA_*. Cada seletor representa um atributo com um valor que você deseja consultar e cada seletor deve ser válido para o tipo de objeto que o identificador de objeto representa. É possível especificar os seletores em qualquer ordem.
- O número de atributos de número inteiro sobre os quais você está consultando. Especifique zero se você não estiver consultando sobre atributos de número inteiro.
- O comprimento do buffer de atributos de caracteres em *CharAttrLength*. Deve ter pelo menos a soma dos comprimentos necessários para reter cada sequência de atributos de caracteres. Especifique zero se não estiver consultando sobre atributos de caracteres.

A saída de MQINQ é:

- Um conjunto de valores de atributos de número inteiro copiado para a matriz. O número de valores é determinado por *IntAttrCount*. Se *IntAttrCount* ou *SelectorCount* for zero, esse parâmetro não será usado.
- O buffer no qual os atributos de caracteres são retornados. O comprimento do buffer é fornecido pelo parâmetro **CharAttrLength**. Se *CharAttrLength* ou *SelectorCount* for zero, esse parâmetro não será usado.
- Um código de conclusão. Se o código de conclusão fornecer um aviso, isso significa que a chamada foi concluída apenas parcialmente. Neste caso, examine o código de razão.
- Um código de razão. Há três situações de conclusão parcial:
  - O seletor não se aplica ao tipo de fila
  - Não há espaço suficiente permitido para atributos de número inteiro
  - Não há espaço suficiente permitido para atributos de caracteres

  Se mais de uma dessas situações surgir, o primeiro que se aplicar será retornado.

Se você abrir uma fila para saída ou consulta e ela resolver em uma fila de cluster não local, será possível consultar somente o nome da fila, o tipo de fila e os atributos comuns. Os valores dos atributos comuns são aqueles da fila escolhida se MQOO_BIND_ON_OPEN tiver sido usado. Os valores são aqueles de uma arbitrária entre as filas de cluster possíveis se MQOO_BIND_NOT_FIXED ou MQOO_BIND_ON_GROUP tiver sido usado ou MQOO_BIND_AS_Q_DEF tiver sido usado e o atributo de fila **DefBind** era MQBND_BIND_NOT_FIXED. Consulte "MQOPEN e clusters" na página 897 e MQOPEN para obter mais informações.

Nota: Os valores retornados pela chamada são uma captura instantânea dos atributos selecionados. Os atributos podem mudar antes que seu programa atue nos valores retornados.

Há uma descrição da chamada MQINQ em MQINQ.

Alguns casos em que a chamada MQINQ falha

Se você abrir um alias para perguntar sobre seus atributos, você é retornado os atributos da fila de alias (o objeto IBM MQ usado para acessar outra fila), não aqueles da fila de base.

No entanto, a definição da fila de base para a qual o alias resolve também é aberta pelo gerenciador de filas e se outro programa mudar o uso da fila de base no intervalo entre suas chamadas MQOPEN e chamadas MQINQ, sua chamada MQINQ falha e retorna o código de razão MQRC_OBJECT_CHANGED. A chamada também falhará se os atributos do objeto da fila de alias forem mudados.

Da mesma forma, quando você abre uma fila remota para consultar sobre seus atributos, os atributos da definição local da fila remota apenas são retornados.

Se você especificar um ou mais seletores que não sejam válidos para o tipo de atributos de filas, o qual você está consultando, a chamada MQINQ será concluída com um aviso e configurará a saída conforme a seguir:

- Para atributos de número inteiro, os elementos correspondentes de *IntAttrs* são configurados como MQIAV_NOT_APPLICABLE.
- Para os atributos de caracteres, as partes correspondentes da sequência *CharAttrs* são configuradas como asteriscos.

Se você especificar um ou mais seletores que não sejam válidos para o tipo de atributos de objeto, o qual você está consultando, a chamada MQINQ falhará e retornará o código de razão MQRC_SELECTOR_ERROR.

Não é possível chamar MQINQ para consultar uma fila modelo; use o recurso MQSC ou os comandos disponíveis em sua plataforma.

Configurando os atributos da fila

Use estas informações para aprender como configurar atributos de filas usando a chamada MQSET.

É possível configurar somente os seguintes atributos de filas usando a chamada MQSET:

- *InhibitGet* (mas não para filas remotas)
- Multi *DistList*
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

A chamada MQSET possui os mesmos parâmetros que a chamada MQINQ. No entanto, para MQSET, todos os parâmetros, exceto o código de conclusão e o código de razão, são parâmetros de entrada. Não há situações conclusão parcial.

Nota: Não é possível usar o MQI para configurar os atributos de objetos do IBM MQ além de filas definidas localmente.

Para obter mais detalhes sobre a chamada MQSET, consulte MQSET.

Confirmando e fazendo backup de unidades de trabalho

Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

Os termos a seguir são usados neste tópico:

- Confirmar
- Restaurar
- Coordenação do ponto de sincronização
- Ponto de Sincronização
- Unidade de trabalho
- Single-phase commit
- Two-phase commit

Se você estiver familiarizado com esses termos de processamento de transações, será possível pular para “Considerações sobre o ponto de sincronização em aplicativos IBM MQ” na página 867.

Confirmação e restauração

Quando um programa coloca uma mensagem em uma fila em uma unidade de trabalho, essa mensagem é visível para outros programas apenas quando o programa confirma a unidade de trabalho. Para confirmar uma unidade de trabalho, todas as atualizações devem ser bem-sucedidas para preservar a integridade dos dados. Se o programa detectar um erro e decidir que a operação

put não é permanente, ele poderá restaurar a unidade de trabalho. Quando um programa executa uma restauração, o IBM MQ restaura a fila removendo as mensagens que foram colocadas na fila por essa unidade de trabalho. A maneira na qual o programa executa a confirmação e a restauração das operações depende do ambiente no qual o programa está sendo executado.

Da mesma forma, quando um programa obtém uma mensagem de uma fila dentro de uma unidade de trabalho, essa mensagem permanece na fila até que o programa confirme a unidade de trabalho, mas a mensagem não está disponível para ser recuperada por outros programas. A mensagem é permanentemente excluída da fila quando o programa confirma a unidade de trabalho. Se o programa restaurar a unidade de trabalho, o IBM MQ restaurará a fila tornando as mensagens disponíveis para serem recuperadas por outros programas.

Coordenação do ponto de sincronização, ponto de sincronização, unidade de trabalho
Coordenação do ponto de sincronização é o processo pelo qual as unidades de trabalho são confirmadas ou restauradas com integridade de dados.

A decisão de confirmar ou restaurar as mudanças é tomada, no caso mais simples, no final de uma transação. No entanto, pode ser mais útil para um aplicativo sincronizar mudanças de dados em outros pontos lógicos dentro de uma transação. Esses pontos lógicos são chamados *pontos de sincronização* (ou *pontos de sincronização*) e o período de processamento de um conjunto de atualizações entre dois pontos de sincronização é chamado *unidade de trabalho*. Várias chamadas MQGET e MQPUT podem fazer parte de uma única unidade de trabalho.

O número máximo de mensagens dentro de uma unidade de trabalho pode ser controlado pelo atributo MAXUMSGS do comando ALTER QMGR.

Single-phase commit
Um processo *single-phase commit* é aquele no qual um programa pode confirmar atualizações em uma fila sem coordenar suas mudanças com outros gerenciadores de recursos.

Two-phase commit
Um processo *two-phase commit* é aquele em que atualizações feitas por um programa nas filas do IBM MQ podem ser coordenadas com atualizações para outros recursos (por exemplo, banco de dados sob o controle do Db2). Sob esse processo, as atualizações em todos os recursos são confirmadas ou restauradas juntas.

Para ajudar a manipular unidades de trabalho, o IBM MQ fornece o atributo **BackoutCount**. Isso é incrementado cada vez que uma mensagem em uma unidade de trabalho é restaurada. Se a mensagem fizer repetidamente com que a unidade de trabalho seja encerrada de forma anormal, o valor de *BackoutCount* finalmente excederá aquele do *BackoutThreshold*. Este valor é configurado quando a fila é definida. Nesta situação, o aplicativo pode remover a mensagem da unidade de trabalho e colocá-la em outra fila, conforme definido em *BackoutRequeueQName*. Quando a mensagem é movida, a unidade de trabalho pode confirmar.

Use os seguintes links para saber mais sobre a confirmação e a restauração de unidades de trabalho:

- “Considerações sobre o ponto de sincronização em aplicativos IBM MQ” na página 867
- z/OS “Syncpoints in IBM MQ for z/OS applications” na página 868
- IBM i “Pontos de sincronização em CICS para os aplicativos IBM i” na página 870
- “Pontos de sincronização em IBM MQ for Multiplatforms” na página 871
- IBM i “Interfaces para o gerenciador de ponto de sincronização externa do IBM i” na página 875

Conceitos relacionados
“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 747

Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Abrindo e fechando objetos” na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Iniciando aplicativos IBM MQ usando acionadores” na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Considerações sobre o ponto de sincronização em aplicativos IBM MQ

Use estas informações para aprender sobre como usar pontos de sincronização em aplicativos IBM MQ.

Two-phase commit é suportado pelos ambientes a seguir:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Servidor de Transação para z/OS .
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS Lote de z/OS com RRS
- Outros coordenadores externos usando a interface X/Open XA

Single-phase commit é suportado pelos ambientes a seguir:

- Multi IBM MQ for Multiplatforms
- z/OS Lote do z/OS

Para obter informações adicionais sobre interfaces externas, consulte “Interfaces para gerenciadores de pontos de sincronização externos em multiplataformas” na página 874 e a documentação do XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publicada pelo The Open Group. Gerenciadores de transações (como CICS, IMS, Encina e Tuxedo) podem participar do two-phase commit coordenado com outros recursos recuperáveis. Isso significa que as funções de enfileiramento fornecidas pelo IBM MQ podem ser colocadas no escopo de uma unidade de trabalho gerenciada pelo gerenciador de transações.

Amostras fornecidas com o IBM MQ mostram o IBM MQ coordenando bancos de dados compatíveis com XA. Para obter informações adicionais sobre essas amostras, consulte “Usando os programas processuais de amostra do IBM MQ” na página 1070.

No seu aplicativo IBM MQ, é possível especificar em cada chamada put e get se você deseja que a chamada seja sob controle do ponto de sincronização. Para fazer uma operação put operar sob o controle do ponto de sincronização, use o valor MQPMO_SYNCPOINT no campo *Options* da estrutura

MQPMO ao chamar MQPUT. Para uma operação get, use o valor MQGMO_SYNCPOINT no campo *Options* da estrutura MQGMO. Se você não escolher explicitamente uma opção, a ação padrão dependerá da plataforma:

- Multi O padrão de controle de ponto de sincronização é NO.
- z/OS O padrão de controle de ponto de sincronização é YES.

Quando uma chamada MQPUT1 for emitida com MQPMO_SYNCPOINT, o comportamento padrão muda, de forma que a operação put seja concluída de forma assíncrona. Isso pode causar uma mudança no comportamento de alguns aplicativos que dependem de determinados campos nas estruturas MQOD e MQMD que estão sendo retornadas, mas que agora contêm valores não definidos. Um aplicativo pode especificar MQPMO_SYNC_RESPONSE para assegurar que a operação put seja executada de forma síncrona e que todos os valores de campos apropriados sejam preenchidos.

Quando seu aplicativo recebe um código de razão MQRC_BACKED_OUT em resposta a um MQPUT ou MQGET sob o ponto de sincronização, o aplicativo deve normalmente recuperar a transação atual usando MQBACK e, em seguida, se apropriado, tentar a transação inteira novamente. Se o aplicativo receber MQRC_BACKED_OUT em resposta a uma chamada MQCMIT ou MQDISC, não precisará chamar MQBACK.

Toda vez que uma chamada MQGET for restaurada, o campo *BackoutCount* da estrutura MQMD da mensagem afetada será incrementado. Um *BackoutCount* alto indica uma mensagem que foi repetidamente restaurada. Isso pode indicar um problema com essa mensagem, que é necessário investigar. Veja BackoutCount para obter detalhes de *BackoutCount*.

Se um programa emitir a chamada MQDISC enquanto houver solicitações não confirmadas, um ponto de sincronização implícito ocorrerá (exceto no z/OS lote com RRS)... Se o programa for encerrado de forma anormal, ocorre uma restauração implícita.

z/OS No z/OS, um ponto de sincronização implícito também ocorrerá se o programa terminar normalmente sem antes chamar MQDISC. O programa é considerado como tendo sido encerrado de forma anormal se o TCB conectado ao MQ for finalizado normalmente. Ao executar no z/OS UNIX System Services com o ambiente de linguagem (LE), a manipulação de condição padrão é chamada para finalizações anormais de tarefa ou sinais. Os manipuladores de condição LE processam a condição de erro e o TCB é encerrado normalmente. Sob essas condições, MQ confirma a unidade de trabalho. Para mais informações, consulte Introdução à Manipulação de Condição de Ambiente de Linguagem.

z/OS Para programas IBM MQ for z/OS, é possível usar a opção MQGMO_MARK_SKIP_BACKOUT para especificar que uma mensagem não deve ser restaurada se a restauração ocorrer (para evitar um loop *MQGET-error-backout*). Para obter informações sobre como usar essa opção, consulte “Ignorando restauração” na página 811.

Mudanças nos atributos da fila (pela chamada MQSET ou por comandos) não são afetadas pela confirmação ou restauração das unidades de trabalho.

z/OS *Syncpoints in IBM MQ for z/OS applications*

This topic explains how to use syncpoints in transaction manager (CICS and IMS) and batch applications.

z/OS *Syncpoints in CICS Transaction Server for z/OS applications*

In a CICS application you establish a syncpoint by using the EXEC CICS SYNCPOINT command.

To back out all changes to the previous syncpoint, you can use the EXEC CICS SYNCPOINT ROLLBACK command. For more information, see the *CICS Application Programming Reference*.

If other recoverable resources are involved in the unit of work, the queue manager (in conjunction with the CICS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

If a CICS application issues the MQDISC call, no implicit syncpoint is taken. If the application closes down normally, any open queues are closed and an implicit commit occurs. If the application closes down abnormally, any open queues are closed and an implicit backout occurs.

z/OS *Syncpoints in IMS applications*

In an IMS application, establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint).

To back out all changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see the IMS documentation.

The queue manager (in conjunction with the IMS syncpoint manager) participates in a two-phase commit protocol if other recoverable resources are also involved in the unit of work.

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify IBM MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application closes down normally, any open queues are closed and an implicit commit occurs. If the application closes down abnormally, any open queues are closed and an implicit backout occurs.

z/OS *Syncpoints in z/OS batch applications*

For batch applications, you can use the IBM MQ syncpoint management calls: MQCMIT and MQBACK. For compatibility with earlier versions, CSQBCMT and CSQBBAK are available as synonyms.

Note: If you need to commit or back out updates to resources managed by different resource managers, such as IBM MQ and Db2, within a single unit of work you can use RRS. For further information see “Transaction management and recoverable resource manager services” on page 870.

Committing changes using the MQCMIT call

As input, you must supply the connection handle (*Hconn*) that is returned by the MQCONN or MQCONNX call.

The output from MQCMIT is a completion code and a reason code. The call completes with a warning if the syncpoint was completed but the queue manager backed out the put and get operations since the previous syncpoint.

Successful completion of the MQCMIT call indicates to the queue manager that the application has reached a syncpoint and that all put and get operations made since the previous syncpoint have been made permanent.

Not all failure responses mean that the MQCMIT did not complete. For example, the application can receive MQRC_CONNECTION_BROKEN.

There is a description of the MQCMIT call in MQCMIT.

Backing out changes using the MQBACK call

As input, you must supply a connection handle (*Hconn*). Use the handle that is returned by the MQCONN or MQCONNX call.

The output from MQBACK is a completion code and a reason code.

The output indicates to the queue manager that the application has reached a syncpoint and that all gets and puts that have been made since the last syncpoint have been backed out.

There is a description of the MQBACK call in MQBACK.

Transaction management and recoverable resource manager services

Transaction management and recoverable resource manager services (RRS) is a z/OS facility to provide two-phase syncpoint support across participating resource managers.

An application can update recoverable resources managed by various z/OS resource managers such as IBM MQ and Db2, and then commit or back out these updates as a single unit of work. RRS provides the necessary unit-of-work status logging during normal execution, coordinates the syncpoint processing, and provides appropriate unit-of-work status information during subsystem restart.

IBM MQ for z/OS RRS participant support enables IBM MQ applications in the batch, TSO, and Db2 stored procedure environments to update both IBM MQ and non-IBM MQ resources (for example, Db2) within a single logical unit of work. For information about RRS participant support, see *z/OS MVS Programming: Resource Recovery*.

Your IBM MQ application can use either MQCMIT and MQBACK or the equivalent RRS calls, SRRCMIT and SRRBACK. See "The RRS batch adapter" on page 903 for more information.

RRS availability

If RRS is not active on your z/OS system, any IBM MQ call issued from a program linked with either RRS stub (CSQBRSTB or CSQBRRSI) returns MQRC_ENVIRONMENT_ERROR.

Db2 stored procedures

If you use Db2 stored procedures with RRS, be aware of the following:

- Db2 stored procedures that use RRS must be managed by workload manager (WLM-managed).
- If a Db2-managed stored procedure contains IBM MQ calls, and it is linked with either RRS stub (CSQBRSTB or CSQBRRSI), the MQCONN or MQCONNX call returns MQRC_ENVIRONMENT_ERROR.
- If a WLM-managed stored procedure contains IBM MQ calls, and is linked with a non-RRS stub, the MQCONN or MQCONNX call returns MQRC_ENVIRONMENT_ERROR, unless it is the first IBM MQ call executed since the stored procedure address space started.
- If your Db2 stored procedure contains IBM MQ calls and is linked with a non-RRS stub, IBM MQ resources updated in that stored procedure are not committed until the stored procedure address space ends, or until a subsequent stored procedure does an MQCMIT (using an IBM MQ Batch/TSO stub).
- Multiple copies of the same stored procedure can execute concurrently in the same address space. Ensure that your program is coded in a reentrant manner if you want Db2 to use a single copy of your stored procedure. Otherwise you might receive MQRC_HCONN_ERROR on any IBM MQ call in your program.
- Do not code MQCMIT or MQBACK in a WLM-managed Db2 stored procedure.
- Design all programs to run in Language Environment (LE).

IBM i *Pontos de sincronização em CICS para os aplicativos IBM i*

O IBM MQ for IBM i participa no CICS para unidades de trabalho do IBM i. É possível usar o MQI dentro de um CICS para que o aplicativo IBM i coloque e obtenha mensagens dentro da unidade de trabalho atual.

É possível usar o comando EXEC CICS SYNCPOINT para estabelecer um ponto de sincronização que inclui as operações do IBM MQ for IBM i. Para recuperar todas as mudanças até o ponto de sincronização anterior, é possível usar o comando EXEC CICS SYNCPOINT ROLLBACK.

Se você usar o MQPUT, MQPUT1 ou MQGET com a opção MQPMO_SYNCPOINT ou MQGMO_SYNCPOINT configurada em um aplicativo CICS para IBM i, você não poderá efetuar logoff do CICS para IBM i até que o IBM MQ for IBM i tenha removido seu registro como um recurso de confirmação de API. Confirme ou recupere sem quaisquer operações put ou get pendentes antes de desconectar do gerenciador de filas. Isso permite que você efetue logoff do CICS para IBM i.

Multi *Pontos de sincronização em IBM MQ for Multiplatforms*

O suporte do ponto de sincronização opera em dois tipos de unidades de trabalho: local e global.

Uma unidade de trabalho *local* é aquela na qual os únicos recursos atualizados são aqueles do gerenciador de filas do IBM MQ. Aqui, a coordenação do ponto de sincronização é fornecida pelo próprio gerenciador de filas usando um procedimento single-phase commit.

Uma unidade de trabalho *global* é aquela na qual os recursos que pertencem a outros gerenciadores de recursos, como bancos de dados, também são atualizados. O IBM MQ pode coordenar essas unidades de trabalho em si. Elas também podem ser coordenadas por um controlador de confirmação externo. Por exemplo:

- Outro gerenciador de transações
- IBM i O controlador de confirmação do IBM i

Para integridade completa, use um procedimento two-phase commit. Two-phase commit pode ser fornecido por bancos de dados e gerenciadores de transações compatíveis com XA. Por exemplo:

- TXSeries
- UDB
- IBM i O controlador de confirmação do IBM i

ALW Os produtos IBM MQ podem coordenar unidades globais de trabalho usando um processo two-phase commit.

IBM i O IBM MQ for IBM i pode agir como um gerenciador de recursos para unidades de trabalho globais dentro de um ambiente do WebSphere Application Server, mas não pode agir como um gerenciador de transações.

Ponto de sincronização implícito

Ao colocar mensagens persistentes, o IBM MQ é otimizado para colocar mensagens persistentes no ponto de sincronização. Vários aplicativos colocando mensagens persistentes na mesma fila executarão melhor se esses aplicativos usarem ponto de sincronização. Isso ocorre porque haverá menos contenção para a fila, se o ponto de sincronização for usado para colocar mensagens persistentes.

ImplSyncOpenOutput inclui um ponto de sincronização implícito quando os aplicativos colocam mensagens persistentes fora do ponto de sincronização. Isso fornece uma melhoria de desempenho, sem aplicativos estando cientes do ponto de sincronização implícito.

O ponto de sincronização implícito fornece apenas um impulso de desempenho quando há vários aplicativos colocando na fila, porque reduz a contenção para a fila. Assim, **ImplSyncOpenOutput** especifica o número mínimo de aplicativos que possuem uma fila aberta para saída antes de um ponto de sincronização implícito ser incluído. O valor padrão é 2. Isto significa, que se você não especificar **ImplSyncOpenOutput**, o ponto de sincronização implícito será apenas incluído se vários aplicativos estiverem alimentando a fila.

Veja Ajustando parâmetros para obter mais informações.

Unidades locais de trabalho no Multiplatforms

Unidades de trabalho que envolvem somente o gerenciador de filas são chamadas de unidades de trabalho *locais*. A coordenação do ponto de sincronização é fornecida pelo próprio gerenciador de filas (coordenação interna) utilizando um processo single-phase commit.

Para iniciar uma unidade de trabalho local, o aplicativo emite solicitações MQGET, MQPUT ou MQPUT1 especificando a opção de ponto de sincronização apropriada. A unidade de trabalho é confirmada usando MQCMIT ou retrocedida usando MQBACK. No entanto, a unidade de trabalho também é encerrada quando a conexão entre o aplicativo e o gerenciador de filas é interrompida, intencionalmente ou não.

Se um aplicativo se desconectar (MQDISC) de um gerenciador de filas enquanto uma unidade de trabalho global coordenada pelo IBM MQ ainda estiver ativa, será feita uma tentativa para confirmar a unidade de trabalho. Se, no entanto, o aplicativo for finalizado sem desconectar, a unidade de trabalho será retrocedida, pois é considerado que o aplicativo foi finalizado de forma anormal.

Unidades globais de trabalho no AIX, Linux, and Windows
Use unidades de trabalho globais quando também precisar incluir atualizações nos recursos pertencentes a outros gerenciadores de recursos. A coordenação pode ser interna ou externa ao gerenciador de filas.

Coordenação de ponto de sincronização interno

A coordenação do gerenciador de filas de unidades globais de trabalho não é suportada no IBM MQ MQI client ambiente.

Aqui, o IBM MQ faz a coordenação. Para iniciar uma unidade de trabalho global, o aplicativo emite a chamada MQBEGIN.

Como entrada para a chamada MQBEGIN, deve-se fornecer a manipulação de conexões (*Hconn*) que é retornada pela chamada MQCONN ou MQCONNX. Esse identificador representa a conexão com o gerenciador de filas do IBM MQ.

O aplicativo emite solicitações MQGET, MQPUT ou MQPUT1 especificando a opção do ponto de sincronização apropriada. Isso significa que é possível usar MQBEGIN para iniciar uma unidade de trabalho global que atualiza recursos locais, recursos que pertencem a outros gerenciadores de recursos, ou ambos. As atualizações feitas nos recursos que pertencem a outros gerenciadores de recursos são feitas usando a API do gerenciador de recursos. No entanto, não é possível usar a MQI para atualizar filas que pertencem a outros gerenciadores de filas. Emita MQCMIT ou MQBACK antes de iniciar mais unidades de trabalho (local ou global).

A unidade de trabalho global é confirmada usando MQCMIT; isso inicia um two-phase commit de todos os gerenciadores de recursos envolvidos na unidade de trabalho. Um processo two-phase commit é usado, pelo qual os gerenciadores de recursos (por exemplo, gerenciadores de bancos de dados compatíveis com XA, como Db2, Oracle e Sybase) são todos solicitados que se preparem para confirmar. Somente se todos estiverem preparados serão solicitados para confirmar. Se qualquer gerenciador de recursos indicar que ele não pode confirmar, cada um é solicitado para restaurar. Como alternativa, é possível usar MQBACK para retroceder as atualizações de todos os gerenciadores de recursos.

Se um aplicativo se desconectar (MQDISC) enquanto uma unidade de trabalho global ainda estiver ativa, a unidade de trabalho será confirmada. Se, no entanto, o aplicativo for finalizado sem desconectar, a unidade de trabalho será retrocedida, pois é considerado que o aplicativo foi finalizado de forma anormal.

A saída de MQBEGIN é um código de conclusão e um código de razão.

Ao usar MQBEGIN para iniciar uma unidade de trabalho global, todos os gerenciadores de recursos externos que foram configurados com o gerenciador de filas são incluídos. No entanto, a chamada iniciar uma unidade de trabalho, mas é concluída com um aviso se:

- Não há gerenciadores de recursos participantes (ou seja, nenhum gerenciador de recurso foi configurado com o gerenciador de filas)

ou

- Um ou mais gerenciadores de recursos não estão disponíveis.

Nesses casos, a unidade de trabalho deve incluir atualizações somente nos gerenciadores de recursos que estavam disponíveis quando a unidade de trabalho foi iniciada.

Se um dos gerenciadores de recursos não puder confirmar suas atualizações, todos os gerenciadores de recursos serão instruídos a retroceder suas atualizações e MQCMIT será concluído com um aviso. Em circunstâncias incomuns (geralmente, a intervenção do operador), uma chamada MQCMIT pode falhar se alguns gerenciadores de recursos confirmarem suas atualizações, mas outros as recuperarem; o trabalho será considerado como tendo sido concluído com um resultado *misto*. Tais ocorrências são diagnosticadas no log de erros do gerenciador de filas para que ação corretiva possa ser executada.

Um MQCMIT de uma unidade de trabalho global será bem-sucedido se todos os gerenciadores de recursos envolvidos confirmarem suas atualizações.

Para obter uma descrição da chamada MQBEGIN, consulte MQBEGIN.

Coordenação de ponto de sincronização externo

Isso ocorre quando um coordenador de ponto sincronização diferente do IBM MQ foi selecionado; por exemplo, CICS, Encina ou Tuxedo.

Nessa situação, os sistemas IBM MQ for AIX, Linux, and Windows registram seu interesse no resultado da unidade de trabalho com o coordenador do ponto de sincronização de modo que eles possam confirmar ou recuperar quaisquer operações de obter ou colocar não confirmadas conforme necessário. O coordenador do ponto de sincronização externo determina se os protocolos de one-phase commit e two-phase commit são fornecidos.

Ao usar um coordenador externo, MQCMIT, MQBACK e MQBEGIN não podem ser emitidos. Chamadas para essas funções falham com o código de razão MQRC_ENVIRONMENT_ERROR.

A maneira pela qual uma unidade de trabalho coordenada externamente é iniciada depende da interface de programação fornecida pelo coordenador do ponto de sincronização. Uma chamada explícita pode ser necessária. Se uma chamada explícita for necessária e você emitir uma chamada MQPUT especificando a opção MQPMO_SYNCPOINT quando uma unidade de trabalho não estiver iniciada, o código de conclusão MQRC_SYNCPOINT_NOT_AVAILABLE será retornado.

O escopo da unidade de trabalho é determinado pelo coordenador do ponto de sincronização. O estado da conexão entre o aplicativo e o gerenciador de filas afeta o sucesso ou falha de chamadas MQI que um aplicativo emite, não o estado da unidade de trabalho. Um aplicativo pode, por exemplo, desconectar e reconectar a um gerenciador de filas durante uma unidade de trabalho ativa e executar outras operações MQGET e MQPUT na mesma unidade de trabalho. Isso é conhecido como uma desconexão pendente.

É possível usar chamadas API do IBM MQ em programas CICS, se optar por usar as capacidades de XA do CICS. Se não usar XA, então, puts e gets de mensagens nas filas não serão gerenciados nas unidades de trabalho atômicas do CICS. Uma razão para escolher esse método é que a consistência geral da unidade de trabalho não é importante para você.

Se a integridade de suas unidades de trabalho for importante para você, então, deverá usar XA. Ao usar XA, o CICS usa um protocolo two-phase commit para assegurar que todos os recursos dentro da unidade de trabalho sejam atualizados em conjunto.

Para obter mais informações sobre a configuração do suporte transacional, consulte Cenários de suporte transacional e também a documentação TXSeries CICS, por exemplo, *TXSeries para Guia de Administração de Multiplataformas CICS para Sistemas Abertos*.

Multi *Ponto de sincronização implícito no Multiplatforms*

O suporte ao ponto de sincronização implícito permite colocações de mensagens persistentes fora do ponto de sincronização.

Ao colocar mensagens persistentes, o IBM MQ é otimizado para colocar mensagens persistentes no ponto de sincronização. Vários aplicativos simultaneamente colocando mensagens persistentes na mesma fila geralmente executarão melhor se esses aplicativos usarem ponto de sincronização. Isso ocorre porque a estratégia de bloqueio do IBM MQ será mais eficiente se o ponto de sincronização for usado ao colocar mensagens persistentes.

O parâmetro **ImplSyncOpenOutput** no arquivo `qm.ini` controla se um ponto de sincronização implícito pode ser incluído quando os aplicativos colocam mensagens persistentes fora do ponto de sincronização. Isso pode fornecer uma melhoria de desempenho, sem aplicativos estando cientes do ponto de sincronização implícito.

O ponto de sincronização implícito só fornece um impulso de desempenho quando há vários aplicativos simultaneamente colocando na fila, porque reduz a contenção de bloqueio. **ImplSyncOpenOutput** especifica o número mínimo de aplicativos que possuem uma fila aberta para saída antes de um ponto de sincronização implícito poder ser incluído. O valor padrão é *2*. Isso significa que, se você não especificar

explicitamente **ImplSyncOpenOutput**, o ponto de sincronização implícito só será incluído se vários aplicativos estiverem colocando na fila.

Se você incluir um ponto de sincronização implícito, as estatísticas refletirão isso e você poderá ver uma saída de transação do **runmqsc display conn**.

Configure **ImplSyncOpenOutput**=*OFF* se você nunca desejar que um ponto de sincronização implícito seja incluído.

Veja Ajustando parâmetros para obter mais informações.

Interfaces para gerenciadores de pontos de sincronização externos em multiplataformas
O IBM MQ for Multiplatforms suporta a coordenação de transações por gerenciadores de pontos de sincronização externos que usam a interface X/Open XA.

Alguns gerenciadores de transações XA (TXSeries) requerem que cada gerenciador de recursos XA forneça seu nome. Essa é a sequência chamada `name` na estrutura do comutador XA.

- ALW O gerenciador de recursos para o IBM MQ no AIX, Linux, and Windows é denominado MQSeries_XA_RMI.
- IBM i Para o IBM i, o nome do gerenciador de recursos é MQSeries XA RMI.

Para obter detalhes adicionais sobre interfaces XA, consulte a documentação do XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publicada pelo The Open Group.

Em uma configuração de XA, o IBM MQ for Multiplatforms cumpre a função de um gerenciador de recursos XA. Um coordenador de ponto de sincronização XA pode gerenciar um conjunto de gerenciadores de recursos XA e sincronizar a confirmação ou a restauração de transações em ambos os gerenciadores de recursos. É assim que funciona para um gerenciador de recursos registrado estaticamente:

1. Um aplicativo notifica o coordenador do ponto de sincronização que deseja iniciar uma transação.
2. O coordenador do ponto de sincronização emite uma chamada para todos os gerenciadores de recursos de seu conhecimento para notificá-los da transação atual.
3. O aplicativo emite chamadas para atualizar os recursos gerenciados pelos gerenciadores de recursos associados à transação atual.
4. O aplicativo solicita que o coordenador do ponto de sincronização confirme ou retroceda a transação.
5. O coordenador do ponto de sincronização emite chamadas para cada gerenciador de recursos usando protocolos two-phase commit para concluir a transação conforme solicitado.

A especificação XA requer que cada gerenciador de recursos forneça uma estrutura chamada Comutador XA. Essa estrutura declara as capacidades do gerenciador de recursos e as funções que devem ser chamadas pelo coordenador do ponto de sincronização.

Há duas versões dessa estrutura:

Tabela 128. Versões do comutador XA

Versão	**Descrição**
MQRMIXASwitch	Gerenciamento de recursos XA estáticos
MQRMIXASwitchDynamic	Gerenciamento de recursos XA dinâmicos

Para uma lista das bibliotecas contendo esta estrutura, consulte A estrutura de alteração do XA IBM MQ.

O método que deve ser usado para vinculá-las a um coordenador de ponto de sincronização XA é definido pelo coordenador; consulte a documentação fornecida por esse coordenador para determinar como ativar o IBM MQ para cooperar com o coordenador do ponto de sincronização XA.

A estrutura *xa_info* que é passada em qualquer chamada *xa_open* pelo coordenador do ponto de sincronização pode ser o nome do gerenciador de filas que deve ser administrado. Tem o mesmo formato que o nome do gerenciador de filas passado para MQCONN ou MQCONNX e pode ficar em branco se o gerenciador de filas padrão for ser usado. No entanto, é possível usar os dois parâmetros extras TPM e AXLIB

TPM permite especificar ao IBM MQ o nome do gerenciador de transações, por exemplo, CICS. AXLIB permite especificar o nome da biblioteca real no gerenciador de transações em que os pontos de entrada XA AX estão localizados.

Se você usar um desses parâmetros ou um gerenciador de filas não padrão, deve-se especificar o nome do gerenciador de filas usando o parâmetro QMNAME. Para obter informações adicionais, consulte Os parâmetros CHANNEL, TRPTYPE, CONNAME e QMNAME da sequência xa_open.

Restrições

1. Unidades de trabalho globais não são permitidas com um Hconn compartilhado (conforme descrito em “Conexões compartilhadas (independentes de encadeamento) com MQCONNX” na página 751).
2. IBM i O IBM MQ for IBM i não suporta registro dinâmico de gerenciadores de recursos XA.

   O único gerenciador de transações suportado é WebSphere Application Server.
3. Windows Nos sistemas Windows, todas as funções declaradas no comutador XA são declarados como funções _cdecl.
4. Um coordenador de ponto de sincronização externo pode administrar somente um gerenciador de filas por vez. Isso porque o coordenador tem uma conexão efetiva para cada gerenciador de filas e, portanto, está sujeito à regra de que somente uma conexão seja permitida por vez.

   Nota: Nota: um aplicativo cliente JMS (aplicativo CLIENT JEE) em execução em um servidor JEE não tem essa restrição, portanto, uma única transação gerenciada pelo servidor JEE pode coordenar vários gerenciadores de filas na mesma transação. No entanto, um aplicativo do servidor JMS, em execução no modo de ligações, ainda está sujeito à regra de que somente uma conexão é permitida por vez.
5. Todos os aplicativos que são executados usando o coordenador do ponto de sincronização podem se conectar somente ao gerenciador de filas administrado pelo coordenador porque já estão efetivamente conectados a esse gerenciador de filas. Eles devem emitir MQCONN ou MQCONNX para obter uma manipulação de conexões e devem emitir MQDISC antes de sair. Como alternativa, podem usar a saída UE014015 para o TXSeries CICS.

IBM i *Interfaces para o gerenciador de ponto de sincronização externa do IBM i*

O IBM MQ for IBM i pode usar o controle de compromisso nativo IBM i como um coordenador do ponto de sincronização externo.

Conexões independentes de encadeamento (compartilhadas) não são permitidas com o controle de compromisso. Consulte *IBM i Programação: Guia de backup e recuperação, SC21-8079* para obter mais informações sobre os recursos de controle de consolidações do IBM i.

Para iniciar os recursos de controle de compromisso do IBM i, use o comando do sistema STRCMTCTL. Para finalizar o controle de confirmações, use o comando do sistema ENDCMTCTL.

Nota: O valor padrão de *Commitment definition scope* é *ACTGRP. Isso deve ser definido como *JOB para IBM MQ para IBM i. Por exemplo:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

O IBM MQ for IBM i também pode executar unidades de trabalho locais contendo apenas atualizações para os recursos do IBM MQ. A escolha entre unidades de trabalho locais e a participação nas unidades de trabalho globais coordenadas por IBM i é feita em cada aplicativo quando o aplicativo chama MQPUT, MQPUT1 ou MQGET, especificando MQGMO_SYNCPOINT ou MQPMO_SYNCPOINT ou MQBEGIN. Se o controle de confirmações não estiver ativo quando essa primeira chamada for emitida, o IBM MQ inicia

uma unidade de trabalho local e todas as unidades de trabalho adicionais para essa conexão com o IBM MQ também usam unidades de trabalho locais, independentemente se o controle de confirmações é iniciado. Para confirmar uma unidade de trabalho local, use MQCMIT. Para restaurar uma unidade de trabalho local, use MQBACK. As chamadas de confirmação e retrocesso do IBM i como o comando de CL COMMIT não têm efeito nas unidades de trabalho locais do IBM MQ.

Se desejar usar o IBM MQ for IBM i com o controle de compromisso nativo do IBM i como um coordenador de ponto de sincronização externo, certifique-se de que qualquer tarefa com o controle de compromisso esteja ativa e que você esteja usando o IBM MQ em uma tarefa de encadeamento único. Se você chamar MQPUT, MQPUT1 ou MQGET, especificando MQGMO_SYNCPOINT ou MQPMO_SYNCPOINT, em uma tarefa multiencadeada na qual o controle de confirmações foi iniciado, a chamada falhará com um código de razão de MQRC_SYNCPOINT_NOT_AVAILABLE.

É possível usar unidades de trabalho locais e as chamadas MQCMIT e MQBACK em uma tarefa multiencadeada.

Se você chamar MQPUT, MQPUT1 ou MQGET, especificando MQGMO_SYNCPOINT ou MQPMO_SYNCPOINT, depois de iniciar o controle de compromisso, o IBM MQ for IBM i inclui-se como um recurso de confirmação de API na definição de confirmação. Essa é geralmente a primeira chamada em uma tarefa. Enquanto houver quaisquer recursos de confirmação da API registrados sob uma determinada definição de confirmação, você não pode finalizar o controle de compromisso para essa definição.

O IBM MQ for IBM i removerá seu registro como um recurso de confirmação de API quando você se desconecta do gerenciador de filas, se não houver operações MQI pendentes na unidade de trabalho atual.

Se você se desconectar do gerenciador de filas enquanto houver operações MQPUT, MQPUT1 ou MQGET pendentes na unidade de trabalho atual, o IBM MQ for IBM i permanece registrado como um recurso de confirmação de API para que ele seja notificado sobre a próxima confirmação ou retrocesso. Quando o próximo ponto de sincronização é atingido, o IBM MQ for IBM i confirma ou retrocede as mudanças conforme necessário. Um aplicativo pode se desconectar e se reconectar a um gerenciador de filas durante uma unidade de trabalho ativa e executar operações MQGET e MQPUT adicionais dentro da mesma unidade de trabalho (essa é uma desconexão pendente).

Se você tentar emitir um comando do sistema ENDCMTCTL para essa definição de confirmação, a mensagem CPF8355 será emitida, indicando que as mudanças pendentes estavam ativas. Essa mensagem também aparece no log da tarefa quando a tarefa é finalizada. Para evitar isso, confirme ou retroceda todas as operações pendentes do IBM MQ for IBM i e desconecte do gerenciador de filas. Assim, usando os comandos COMMIT ou ROLLBACK antes de ENDCMTCTL permite que o controle de compromisso de término seja concluído com sucesso.

Ao usar o IBM i, o controle de compromisso como um coordenador de ponto de sincronização externo, você não pode emitir as chamadas MQCMIT, MQBACK e MQBEGIN. Chamadas para essas funções falham com o código de razão MQRC_ENVIRONMENT_ERROR.

Para confirmar ou retroceder (ou seja, para restaurar) sua unidade de trabalho, use uma das linguagens de programação que suporta o controle de confirmações. Por exemplo:

- Comandos CL: COMMIT e ROLLBACK
- Funções de programação ILE C: _Rcommit e _Rrollback
- ILE RPG: COMMIT e ROLBK
- COBOL/400: COMMIT e ROLLBACK

Ao usar o controle de compromisso do IBM i como um coordenador de ponto de sincronização externo com o IBM MQ for IBM i, o IBM i executa um protocolo de confirmação de duas fases no qual o IBM MQ participa. Como cada unidade de trabalho é confirmada em duas fases, o gerenciador de filas pode se tornar indisponível para a segunda fase após ter votado para confirmar na primeira fase. Isso pode acontecer, por exemplo, se as tarefas internas do gerenciador de filas forem finalizadas. Nesta situação, o log da tarefa que executa a confirmação contém a mensagem CPF835F indicando que uma operação de confirmação ou retrocesso falhou. As mensagens anteriores indicam a causa do problema, se ocorreu

durante uma operação de confirmação ou retrocesso e também o ID da unidade de trabalho lógica (LUWID) para a unidade de trabalho com falha.

Se o problema foi causado pela falha do recurso de confirmação de API do IBM MQ durante a confirmação ou a retrocesso de uma unidade de trabalho preparada, é possível usar o comando WRKMQMTRN para concluir a operação e restaurar a integridade da transação. O comando requer que você saiba o LUWID da unidade de trabalho para confirmar e retroceder.

Iniciando aplicativos IBM MQ usando acionadores

Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

Alguns aplicativos IBM MQ que atendem filas são executados continuamente, portanto, eles estão sempre disponíveis para recuperar mensagens que chegam nas filas. No entanto, talvez você não queira isso quando o número de mensagens que chegam nas filas for imprevisível. Neste caso, os aplicativos poderiam estar consumindo os recursos do sistema mesmo quando não houver mensagens a recuperar.

O IBM MQ fornece um recurso que permite que um aplicativo seja iniciado automaticamente quando houver mensagens disponíveis a recuperar. Este recurso é conhecido como *acionamento*.

Para obter informações sobre o acionamento de canais, consulte Acionando canais.

O que É o Acionamento?

O gerenciador de filas define certas condições que constituem os *eventos acionadores*.

Se o acionamento for ativado para uma fila e ocorrer um evento acionador, o gerenciador de filas enviará uma *mensagem do acionador* para uma fila denominada *fila de inicialização*. A presença da mensagem do acionador na fila de inicialização indica que ocorreu um evento acionador.

As mensagens do acionador geradas pelo gerenciador de filas não são persistentes. Isso reduz a criação de log (resultando em um desempenho melhor) e minimiza as duplicatas durante a reinicialização, melhorando, assim, o tempo de reinicialização.

O programa que processa a fila de inicialização é chamado de aplicativo *acionador-monitor* e sua função é ler a mensagem do acionador e executar a ação apropriada, com base nas informações contidas na mensagem do acionador. Geralmente, esta ação é iniciar algum outro aplicativo para processar a fila que gerou a mensagem do acionador. Do ponto de vista do gerenciador de filas, não há nada de especial sobre o aplicativo acionador-monitor; ele é simplesmente outro aplicativo que lê mensagens de uma fila (a fila de inicialização).

Se o acionamento for ativado para uma fila, será possível criar um *objeto de definição de processo* associado a ele. Este objeto contém informações sobre o aplicativo que processa a mensagem que causou o evento do acionador. Se o objeto de definição de processo for criado, o gerenciador de filas extrairá essas informações e as colocará na mensagem do acionador para serem usadas pelo aplicativo acionador-monitor. O nome da definição de processo associada a uma fila é determinado pelo atributo de fila local *ProcessName*. Cada fila pode especificar uma definição de processo diferente ou várias filas podem compartilhar a mesma definição de processo.

Se desejar acionar o início de um canal, não será necessário definir um objeto de definição de processo. A definição da fila de transmissão é usada em substituição.

O acionamento é suportado pelos clientes IBM MQ em execução no AIX, Linux, and Windows. Um aplicativo em execução em um ambiente do cliente é o mesmo que um em execução em um ambiente completo do IBM MQ, exceto que você o vincula às bibliotecas do cliente. No entanto, o monitor do acionador e o aplicativo a ser iniciado devem estar no mesmo ambiente.

O acionamento envolve:

Fila de aplicativos
: Uma *fila de aplicativos* é uma fila local que, quando tem o acionamento configurado e quando as condições são atendidas, requer que as mensagens do acionador sejam gravadas.

Definição de processo
Uma fila de aplicativos pode ter um *objeto de definição de processo* associado a ela que mantém detalhes do aplicativo que obterá mensagens da fila de aplicativos. (Consulte Atributos para definições de processo para obter uma lista de atributos.)

Lembre-se de que se desejar que um acionador inicie um canal, não será necessário definir um objeto de definição de processo.

Fila de transmissão
Você precisará de uma fila de transmissão, se desejar que um acionador inicie um canal.

Para uma fila de transmissão em qualquer plataforma diferente do Linux, o atributo *TriggerData* da fila de transmissão pode especificar o nome do canal a ser iniciado. Isso pode substituir a definição do processo para acionar os canais, mas é usado apenas quando uma definição de processo não é criada.

Evento acionador
Um *evento do acionador* é um evento que faz com que uma mensagem do acionador seja gerada pelo gerenciador de filas. Isto é, geralmente, uma mensagem que chega em uma fila de aplicativos, mas pode também ocorrer em outros momentos. Por exemplo, consulte “Condições para um evento acionador” na página 883.

O IBM MQ tem um intervalo de opções para permitir que você controle as condições que causam um evento do acionador (consulte “Controlando eventos acionadores” na página 887).

Mensagem do acionador
O gerenciador de filas cria uma *mensagem do acionador* quando reconhece um evento do acionador. Ele copia para as informações de mensagem do acionador sobre o aplicativo a ser iniciado. Estas informações vêm da fila de aplicativos e do objeto de definição de processo associado à fila de aplicativos.

As mensagens do acionador têm um formato fixo (consulte “Formato de mensagens do acionador” na página 895).

Fila de Inicialização
Uma *fila de inicialização* é uma fila local na qual o gerenciador de filas coloca mensagens do acionador. Observe que uma fila de inicialização não pode ser uma fila de alias ou uma fila modelo.

Um gerenciador de filas pode possuir mais de uma fila de inicialização e cada uma é associada a uma ou mais filas de aplicativos.

z/OS Uma fila compartilhada, uma fila local acessível por gerenciadores de filas em um grupo de filas compartilhadas, pode ser uma fila de iniciação no IBM MQ for z/OS.

Monitor acionador
Um *monitor acionador* é um programa continuamente em execução que atende uma ou mais filas de inicialização. Quando uma mensagem do acionador chega em uma fila de iniciação, o monitor de disparo recupera a mensagem. O monitor acionador usa as informações na mensagem do acionador. Ele emite um comando para iniciar o aplicativo que deve recuperar as mensagens que chegam na fila de aplicativos, transmitindo a ele as informações contidas no cabeçalho da mensagem do acionador, que inclui o nome da fila de aplicativos.

Em todas as plataformas, um monitor acionador especial conhecido como inicializador de canais é responsável por iniciar canais.

z/OS No z/OS, o inicializador de canais é tipicamente iniciado manualmente ou isso pode ser feito automaticamente quando um gerenciador de filas começa alterando o CSQINP2 na inicialização do gerenciador de filas.

Multi Em Multiplataformas, o inicializador de canais é iniciado automaticamente quando o gerenciador de filas é iniciado ou pode ser iniciado manualmente com o comando **runmqchi**.

Para obter informações adicionais, consulte “Processamento da fila de inicialização por monitores acionadores” na página 891.

Para entender como o acionamento funciona, considere Figura 95 na página 879, que é um exemplo do tipo de acionador FIRST (MQTT_FIRST).

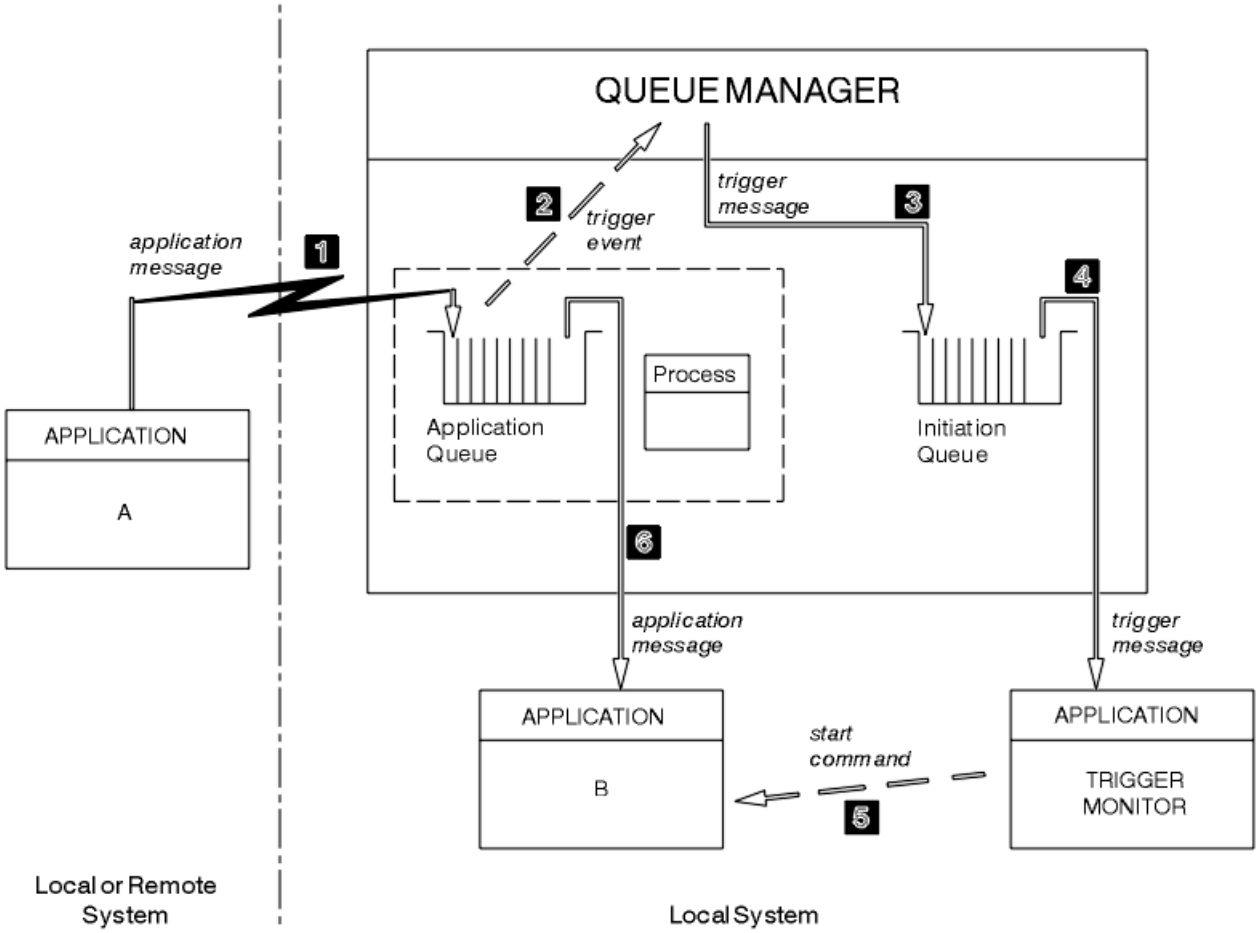


Figura 95. Fluxo de mensagens do aplicativo e do acionador

No Figura 95 na página 879, a sequência de eventos é:

1. O Aplicativo A, que pode ser local ou remoto para o gerenciador de filas, coloca uma mensagem na fila de aplicativos. Nenhum aplicativo tem essa fila aberta para entrada. No entanto, este fato é relevante apenas para acionar o tipo FIRST e DEPTH.
2. O gerenciador de filas verifica se as condições são atendidas sob as quais ele tem que gerar um evento acionador. Elas são, e um evento acionador é gerado. As informações retidas no objeto de definição de processo associado são usadas ao criar a mensagem do acionador.
3. O gerenciador de filas cria uma mensagem do acionador e coloca-a na fila de inicialização associada a esta fila de aplicativos, mas apenas se um aplicativo (monitor acionador) tiver a fila de inicialização aberta para entrada.
4. O monitor acionador recupera a mensagem do acionador da fila de inicialização.
5. O monitor acionador emite um comando para iniciar o aplicativo B (o aplicativo do servidor).
6. O Aplicativo B abre a fila de aplicativos e recupera a mensagem.

Nota:

1. Se a fila de aplicativos for aberta para entrada, por qualquer programa, e tiver o acionamento configurado como FIRST ou DEPTH, nenhum evento acionador ocorrerá, porque a fila já está sendo atendida.
2. Se a fila de inicialização não for aberta para entrada, o gerenciador de filas não gerará nenhuma mensagem do acionador; ele aguardará até que um aplicativo abra a fila de inicialização para entrada.
3. Ao usar o acionamento para os canais, use o tipo de acionador FIRST ou DEPTH.

4. Aplicativos acionados executados sob o ID do usuário e o grupo do usuário que iniciou o monitor acionador, o usuário do CICS ou o usuário que iniciou o gerenciador de filas.

Até o momento, o relacionamento entre as filas no acionamento tem sido apenas na base de um para um. Considere Figura 96 na página 880.

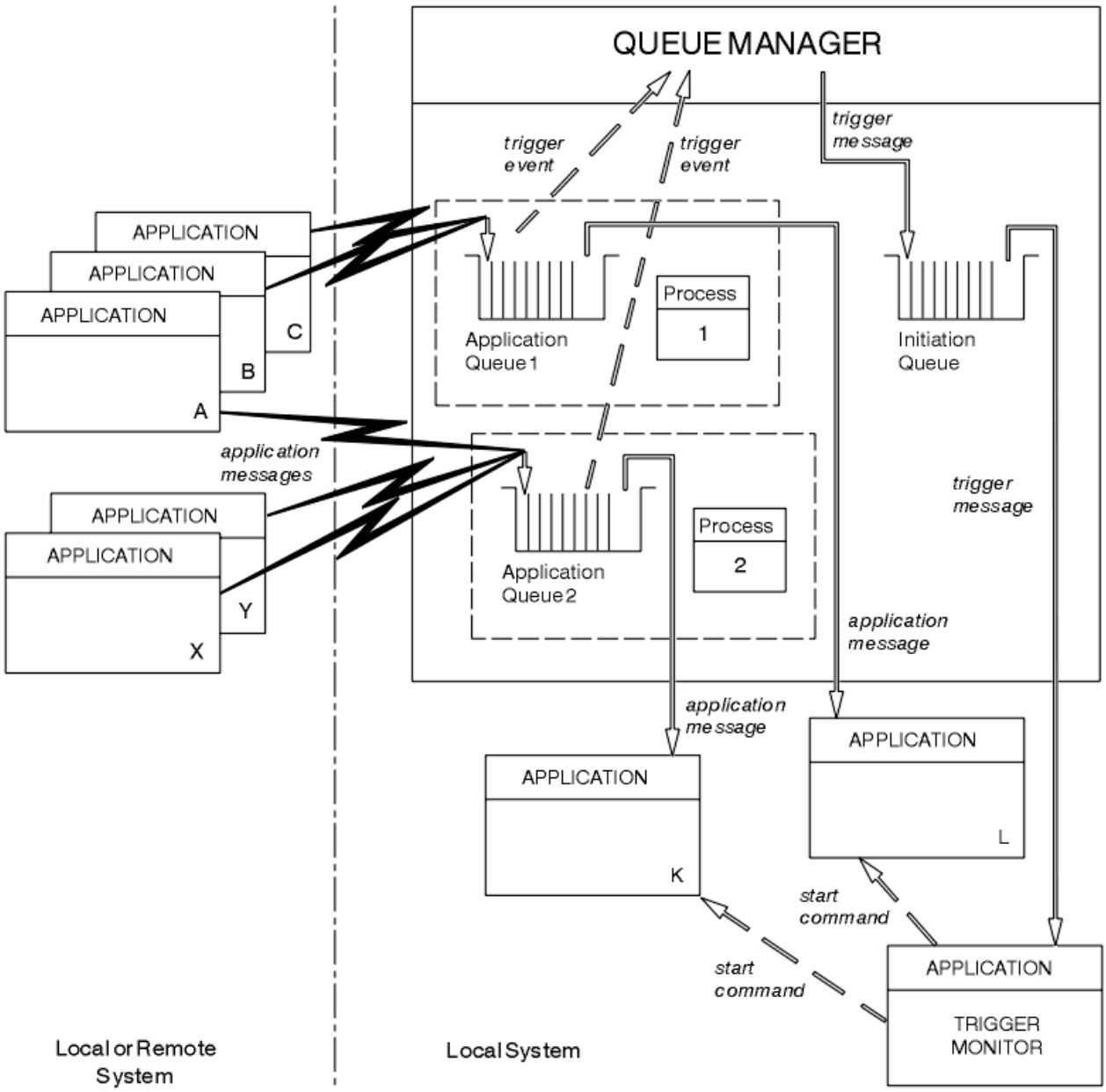


Figura 96. Relacionamento de filas no acionamento

Uma fila de aplicativos possui um objeto de definição de processo associado a ela que retém detalhes do aplicativo que processará a mensagem. O gerenciador de filas coloca as informações na mensagem do acionador, portanto, apenas uma fila de inicialização é necessária. O monitor acionador extrai essas informações da mensagem do acionador e inicia o aplicativo relevante para tratar a mensagem em cada fila de aplicativos.

Lembre-se de que, se desejar acionar o início de um canal, você não precisa definir um objeto de definição de processo. A definição de fila de transmissão pode determinar o canal a ser acionado.

Use os links a seguir para saber mais sobre como iniciar IBM MQ aplicativos usando acionadores:

- "Pré-requisitos para o acionamento" na página 881

- “Condições para um evento acionador” na página 883
- “Controlando eventos acionadores” na página 887
- “Projetando um aplicativo que usa filas acionadas” na página 890
- “Processamento da fila de inicialização por monitores acionadores” na página 891
- “Propriedades de mensagens do acionador” na página 894
- “Quando o acionamento não funciona” na página 896

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Abrindo e fechando objetos” na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Trabalhando com MQI e clusters” na página 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Using and writing applications on IBM MQ for z/OS” na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

“IMS and IMS bridge applications on IBM MQ for z/OS” na página 71
This information helps you to write IMS applications using IBM MQ.

Pré-requisitos para o acionamento

Use essas informações para aprender sobre as etapas a serem executadas antes de usar o acionamento.

Antes de seu aplicativo poder aproveitar o acionamento, conclua as seguintes etapas:

1. Então:
   a. Crie uma fila de inicialização para sua fila de aplicativo. Por exemplo:

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   ou

   b. Determine o nome de uma fila local que existe e pode ser usada pelo aplicativo (geralmente, esse nome é SYSTEM.DEFAULT.INITIATION.QUEUE ou, se você estiver iniciando canais com acionadores, SYSTEM.CHANNEL.INITQ) e especifique seu nome no campo *InitiationQName* da fila de aplicativos.

2. Associe a fila de inicialização à fila de aplicativos. Um gerenciador de filas pode possuir mais de uma fila de inicialização. Talvez você queira que algumas de suas filas de aplicativos sejam atendidas por programas diferentes; neste caso, é possível usar uma fila de inicialização para cada programa de atendimento, embora isso não seja obrigatório. A seguir está um exemplo de como criar uma fila de aplicativos:

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i A seguir está uma extração de um programa CL para o IBM MQ for IBM i que cria uma fila de inicialização:

```
/*   Queue used by AMQSINQA                                        */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                        QTYPE(*LCL)  REPLACE(*YES)                  +
                        MQMNAME                                     +
                        TEXT('queue for AMQSINQA')                  +
                        SHARE(*YES)                 /* Shareable   */+
                        DFTMSGPST(*YES)/* Persistent messages OK   */+
                                                                    +
                        TRGENBL(*YES)  /* Trigger control on       */+
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. Se estiver acionando um aplicativo, crie um objeto de definição de processo para conter informações relacionadas ao aplicativo que deve atender sua fila de aplicativos. Por exemplo, para acionar-iniciar uma transação da folha de pagamento do CICS chamada PAYR:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i A seguir está uma extração de um programa CL para o IBM MQ for IBM i que cria um objeto de definição de processo:

```
/*   Process definition                                            */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        REPLACE(*YES)                               +
                        MQMNAME                                     +
                        TEXT('trigger process for AMQSINQA')        +
                        ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                        APPID('AMQSINQA')    /* Program name      */
```

Quando o gerenciador de filas cria uma mensagem do acionador, ele copia informações dos atributos do objeto de definição de processo para a mensagem do acionador.

Plataforma	Para criar um objeto de definição de processo
Sistemas AIX, Linux, and Windows	Use DEFINE PROCESS ou use SYSTEM.DEFAULT.PROCESS e modifique usando ALTER PROCESS
z/OS z/OS	Use DEFINE PROCESS (consulte o código de amostra na etapa “3” na página 882) ou use as operações e os painéis de controle.

Plataforma	Para criar um objeto de definição de processo
IBM i IBM i	Use um programa CL que contenha código como na etapa “3” na página 882.

4. Opcional: Crie uma definição de fila de transmissão e use espaços em branco para o atributo **ProcessName**.

   O atributo **TrigData** pode conter o nome do canal a ser acionado ou pode ser deixado em branco; exceto em IBM MQ for z/OS, se for deixado em branco, o inicializador de canais procurará os arquivos de definição de canal até localizar um canal que esteja associado à fila de transmissão nomeada. Quando o gerenciador de filas cria uma mensagem do acionador, ele copia as informações do atributo **TrigData** da definição de fila de transmissão para a mensagem do acionador.

5. Se você tiver criado um objeto de definição de processo para especificar propriedades do aplicativo que deve atender sua fila de aplicativos, associe o objeto do processo à sua fila de aplicativos nomeando-o no atributo **ProcessName** da fila.

Plataforma	Use os comandos
Sistemas AIX, Linux, and Windows	ALTER QLOCAL
z/OS z/OS	ALTER QLOCAL
IBM i IBM i	CHGMQMQ

6. Inicie as instâncias dos monitores do acionador IBM i (ou servidores do acionador no IBM MQ for IBM i) que devem atender as filas de inicialização que você definiu. Consulte “Processamento da fila de inicialização por monitores acionadores” na página 891 para obter mais informações.

Se você deseja estar ciente de quaisquer mensagens do acionador não entregues, certifique-se de que seu gerenciador de filas possua uma fila de devoluções (mensagens não entregues) definida. Especifique o nome da fila no campo do gerenciador de filas *DeadLetterQName*.

É possível, então, configurar as condições do acionador que você precisa, usando os atributos do objeto de fila que define sua fila de aplicativos. Para obter mais informações, consulte “Controlando eventos acionadores” na página 887.

Condições para um evento acionador

O gerenciador de filas cria uma mensagem do acionador quando determinadas condições são atendidas

As condições a seguir fazem com que o gerenciador de filas crie uma mensagem do acionador:

1. Uma mensagem tem *put* efetuado em uma fila.
2. A mensagem tem uma prioridade maior ou igual à prioridade do acionador de limite da fila. Essa prioridade é configurada no atributo de fila local **TriggerMsgPriority**; se for configurada para zero, nenhuma mensagem se qualifica.
3. O número de mensagens na fila com prioridade maior ou igual a *TriggerMsgPriority* era anteriormente, dependendo de *TriggerType*:
   - Zero (para tipo de acionador MQTT_FIRST)
   - Qualquer número (para tipo de acionador MQTT_EVERY)
   - *TriggerDepth* menos 1 (para tipo de acionador MQTT_DEPTH)

   Nota:

- Para filas locais não compartilhadas, o gerenciador de filas conta mensagens confirmadas e não confirmadas quando avalia se as condições para um evento do acionador existem. Consequentemente, um aplicativo pode ser iniciado quando não houver mensagens para ele recuperar porque as mensagens na fila não foram confirmadas. Nessa situação, considere usar a opção de espera com um *WaitInterval* adequado para que o aplicativo espere a chegada de suas mensagens.
- z/OS Para filas locais compartilhadas, o gerenciador de filas conta somente as mensagens confirmadas.

4. Para acionamento do tipo FIRST ou DEPTH, nenhum programa tem a fila do aplicativo aberta para a remoção de mensagens (ou seja, o atributo de fila local **OpenInputCount** é zero).

   Nota: z/OS

   - Para filas compartilhadas, condições especiais se aplicam quando vários gerenciadores de filas têm monitores acionadores em execução com relação a uma fila. Nesta situação, se um ou mais gerenciadores de filas têm a fila aberta para entrada compartilhada, os critérios do acionador nos outros gerenciadores de filas são tratados como *TriggerType* MQTT_FIRST e *TriggerMsgPriority* zero. Quando todos os gerenciadores de filas fecham a fila para entrada, as condições acionadoras revertem para as condições especificadas na definição de fila.

     Um cenário de exemplo afetado por essa condição é vários gerenciadores de filas QM1, QM2 e QM3 com um monitor acionador em execução para uma fila de aplicativos A. A mensagem chega em A, atendendo às condições para o acionamento e uma mensagem acionadora é gerada na fila de inicialização. O monitor acionador em QM1 obtém a mensagem do acionador e aciona um aplicativo. O aplicativo acionado abre a fila do aplicativo para entrada compartilhada. A partir desse ponto, as condições acionadoras para uma fila do aplicativo são avaliadas como *TriggerType* MQTT_FIRST e *TriggerMsgPriority* zero nos gerenciadores de filas QM2 e QM3, até QM1 fechar a fila do aplicativo.
   - Para filas compartilhadas, essa condição é aplicada a cada gerenciador de filas. Ou seja, um gerenciador de filas *OpenInputCount* para uma fila deve ser zero para que uma mensagem do acionador seja gerada para a fila por esse gerenciador de filas. No entanto, se algum gerenciador de filas no grupo de filas compartilhadas tiver a fila aberta usando a opção MQOO_INPUT_EXCLUSIVE, nenhuma mensagem do acionador será gerada para essa fila por qualquer um dos gerenciadores de filas no grupo de filas compartilhadas.

     A mudança no modo como as condições acionadoras são avaliadas ocorre quando o aplicativo acionado abre a fila para entrada. Em cenários em que há somente um monitor acionador em execução, outros aplicativos podem ter o mesmo efeito, pois abrem a fila do aplicativo para entrada de forma semelhante. Não importa se a fila do aplicativo foi aberta por um aplicativo iniciado por um monitor acionador ou por algum outro aplicativo; é o fato de que a fila está aberta para entrada em outro gerenciador de filas que causa a mudança de critérios do acionador.

5. z/OS No IBM MQ for z/OS, se a fila do aplicativo for uma com um atributo **Usage** de MQUS_NORMAL, solicitações get para ela não são inibidas (ou seja, o atributo de fila **InhibitGet** é MQQA_GET_ALLOWED). Além disso, se a fila do aplicativo acionada for uma com um atributo **Usage** de MQUS_XMITQ, as solicitações get para ela não são inibidas.
6. Então:
   - O atributo **ProcessName** da fila local para a fila não está em branco e o objeto de definição de processo identificado por esse atributo foi criado ou
   - O atributo **ProcessName** da fila local para a fila está todo em branco, mas a fila é uma fila de transmissão. Como a definição de processo é opcional, o atributo **TriggerData** também pode conter o nome do canal a ser iniciado. Nesse caso, a mensagem do acionador contém atributos com os valores a seguir:
     - **QName**: nome da fila
     - **ProcessName**: em branco
     - **TriggerData**: dados do acionador

- **ApplType**: MQAT_UNKNOWN
- **ApplId**: em branco
- **EnvData**: em branco
- **UserData**: em branco

7. Uma fila de inicialização foi criada e foi especificada no atributo **InitiationQName** da fila local. Além disso:
   - Solicitações de Get não são inibidas para a fila de inicialização (isto é, o valor do atributo de fila **InhibitGet** é MQQA_GET_ALLOWED).
   - Solicitações de Put não devem ser inibidas para a fila de inicialização (isto é, o valor do atributo de fila **InhibitPut** deve ser MQQA_PUT_ALLOWED).
   - O valor do atributo **Usage** da fila de inicialização deve ser MQUS_NORMAL.
   - Em ambientes nos quais filas dinâmicas são suportadas, a fila de inicialização não deve ser uma fila dinâmica que foi marcada como excluída logicamente.
8. Um monitor acionador atualmente tem a fila de inicialização aberta para remover mensagens (ou seja, o atributo **OpenInputCount** da fila local é maior que zero).
9. O controle do acionador (atributo **TriggerControl** da fila local) para a fila do aplicativo está configurado para MQTC_ON. Para fazer isso, configure o atributo **trigger** ao definir a fila ou use o comando ALTER QLOCAL.
10. O tipo de acionador (atributo **TriggerType** da fila local) não é MQTT_NONE.

    Se todas as condições necessárias forem atendidas e a mensagem que causou a condição do acionador for colocada como parte de uma unidade de trabalho, a mensagem do acionador não será disponibilizada para recuperação pelo aplicativo monitor acionador até a unidade de trabalho ser concluída, se a unidade de trabalho for confirmada ou, para o tipo de acionador MQTT_FIRST ou MQTT_DEPTH, restaurada.
11. Uma mensagem adequada é colocado na fila, para um **TriggerType** de MQTT_FIRST ou MQTT_DEPTH e a fila:
    - Não estava vazia anteriormente (MQTT_FIRST) ou
    - Tinha **TriggerDepth** ou mais mensagens (MQTT_DEPTH)

    e condições “2” na página 883 a “10” na página 885 (excluindo “3” na página 883) forem satisfeitas, se no caso de MQTT_FIRST um intervalo suficiente (atributo do gerenciador de filas **TriggerInterval**) tiver decorrido desde que a última mensagem do acionador foi gravada para essa fila.

    Isso é para permitir um servidor da fila que é finalizado antes de processar todas as mensagens na fila. O propósito do intervalo do acionador é reduzir o número de mensagens do acionador duplicadas que são geradas.

    Nota: Se você parar e reiniciar o gerenciador de filas, o cronômetro **TriggerInterval** será reconfigurado. Há uma pequena janela durante a qual é possível produzir duas mensagens do acionador. A janela existe quando o atributo trigger da fila estiver configurado para ativado ao mesmo tempo que uma mensagem chega e a fila não estava anteriormente vazia (MQTT_FIRST) ou tinha **TriggerDepth** ou mais mensagens (MQTT_DEPTH).
12. O único aplicativo que atende uma fila emite uma chamada MQCLOSE para um **TriggerType** igual a MQTT_FIRST ou MQTT_DEPTH e há pelo menos:
    - Um (MQTT_FIRST) ou
    - **TriggerDepth** (MQTT_DEPTH)

    mensagens na fila de prioridade suficiente (condição “2” na página 883) e as condições “6” na página 884 a “10” na página 885 também são satisfeitas.

Isso é para permitir um servidor da fila que emite uma chamada MQGET, encontra a fila vazia e assim finalize; no entanto, no intervalo entre as chamadas MQGET e MQCLOSE, uma ou mais mensagens chegam.

Nota:

a. Se o programa que está atendendo a fila do aplicativo não recuperar todas as mensagens, isso pode causar um loop fechado. Toda vez que o programa fechar a fila, o gerenciador de filas criará outra mensagem do acionador que fará o monitor acionador iniciar o programa do servidor novamente.

b. Se o programa que está atendendo a fila do aplicativo recuperar sua solicitação get (ou se o programa for encerrado de forma anormal) antes de fechar a fila, o mesmo acontece. No entanto, se o programa fechar a fila antes de recuperar a solicitação get e a fila estiver vazia, nenhuma mensagem do acionador será criado.

c. Para evitar que esse loop ocorra, use o campo *BackoutCount* de MQMD para detectar mensagens que são restauradas repetidamente. Para obter mais informações, consulte "Mensagens que são restauradas" na página 47.

13. As condições a seguir são satisfeitas usando MQSET ou um comando:

a. • **TriggerControl** é mudado para MQTC_ON ou

• **TriggerControl** já é MQTC_ON e o valor de um **TriggerType**, **TriggerMsgPriority** ou **TriggerDepth** (se relevante) é mudado,

e há pelo menos:

• Uma (MQTT_FIRST ou MQTT_EVERY) ou

• **TriggerDepth** (MQTT_DEPTH)

mensagens na fila de prioridade suficiente (condição "2" na página 883) e as condições "4" na página 884 a "10" na página 885 (excluindo "8" na página 885) também são satisfeitas.

Isso é para permitir que um aplicativo ou operador mude os critérios de acionamento, quando as condições para que um acionador ocorra já estiverem satisfeitas.

b. O valor do atributo de fila **InhibitPut** de uma fila de inicialização muda de MQQA_PUT_INHIBITED para MQQA_PUT_ALLOWED e há pelo menos:

• Uma (MQTT_FIRST ou MQTT_EVERY) ou

• **TriggerDepth** (MQTT_DEPTH)

mensagens de prioridade suficiente (condição "2" na página 883) em qualquer uma das filas para as quais essa é a fila de inicialização e as condições "4" na página 884 a "10" na página 885 também são satisfeitas. (Uma mensagem do acionador é gerada para cada uma dessas filas que esteja satisfazendo as condições.)

Isso é para permitir que as mensagens do acionador não sejam geradas por causa da condição MQQA_PUT_INHIBITED na fila de inicialização, mas essa condição agora foi mudada.

c. O valor do atributo de fila **InhibitGet** de uma fila do aplicativo muda de MQQA_GET_INHIBITED para MQQA_GET_ALLOWED, e há pelo menos:

• Uma (MQTT_FIRST ou MQTT_EVERY) ou

• **TriggerDepth** (MQTT_DEPTH)

mensagens de prioridade suficiente (condição "2" na página 883) na fila, e as condições "4" na página 884 até "10" na página 885, excluindo "5" na página 884, também são satisfeitas

Isso permite que os aplicativos sejam acionados somente quando puderem recuperar mensagens da fila do aplicativo.

d. Um aplicativo monitor acionador emite uma chamada MQOPEN para entrada a partir de uma fila de inicialização e há pelo menos:

• Uma (MQTT_FIRST ou MQTT_EVERY) ou

- **TriggerDepth** (MQTT_DEPTH)

  mensagens de prioridade suficiente (condição “2” na página 883) em qualquer uma das filas do aplicativo para a qual esta é a fila de inicialização e as condições “4” na página 884 a “10” na página 885 (excluindo a “8” na página 885) também são satisfeitas e nenhum outro aplicativo tem a fila de inicialização aberta para entrada (uma mensagem do acionador será gerada para cada fila que satisfaça as condições).

  Isso é para permitir que mensagens cheguem nas filas enquanto o monitor acionador não está em execução e o gerenciador de filas seja reiniciado e as mensagens do acionador (que são persistentes) sejam perdidas.

14. MSGDLVSQ está configurado corretamente. Se você configurar MSGDLVSQ=FIFO, as mensagens serão entregues à fila em um modo Primeiro a entrar, primeiro a sair. A prioridade da mensagem é ignorada e a prioridade padrão da fila é designada à mensagem. Se **TriggerMsgPriority** for configurado para um valor maior que a prioridade padrão da fila, nenhuma mensagem será acionada. Se **TriggerMsgPriority** for configurado igual ou menor que a prioridade padrão da fila, o acionamento ocorrerá para tipo FIRST, EVERY e DEPTH. Para obter informações sobre esses tipos, consulte a descrição do campo **TriggerType** campo em “Controlando eventos acionadores” na página 887.

    Se você configurar MSGDLVSQ=PRIORITY e a prioridade da mensagem for igual ou maior que o campo *TriggerMsgPriority*, as mensagens serão consideradas apenas para um evento acionador. Nesse caso, o acionamento ocorre para o tipo FIRST, EVERY e DEPTH. Como um exemplo, se você colocar 100 mensagens de prioridade mais baixa que **TriggerMsgPriority**, a profundidade da fila efetiva para propósitos de acionamento ainda será zero. Se, então, colocar outra mensagem na fila, mas, desta vez, a prioridade for maior ou igual a **TriggerMsgPriority**, a profundidade da fila efetiva aumenta de zero para um e a condição para **TriggerType** FIRST é satisfeita.

Notas:

1. A partir da etapa “12” na página 885 (em que as mensagens do acionador são geradas como resultado de algum outro evento diferente de uma mensagem que chega na fila do aplicativo), a mensagem do acionador não será colocada como parte de uma unidade de trabalho. Além disso, se **TriggerType** for MQTT_EVERY e se houver uma ou mais mensagens na fila do aplicativo, somente uma mensagem do acionador será gerada.
2. Se o IBM MQ segmentar uma mensagem durante a chamada MQPUT, um evento acionador não será processado até que todos os segmentos tenham sido colocados na fila com sucesso. No entanto, quando os segmentos de mensagem estiverem na fila, o IBM MQ os trata como mensagens individuais para propósitos de acionamento. Por exemplo, uma única mensagem lógica dividida em três partes faz com que somente um evento do acionador seja processado quando inicialmente passa por MQPUT e é segmentada. No entanto, cada um dos três segmentos faz com que seus próprios eventos acionadores sejam processados à medida que são movidos por meio da rede do IBM MQ.
3. z/OS Para IBM MQ for z/OS, se uma fila compartilhada for configurada para acionamento e conexão com o Recurso de Acoplamento que hospeda a fila compartilhada for perdida, um evento acionador poderá ser gerado e uma mensagem colocada na fila de inicialização. Isso pode acontecer até mesmo quando nenhuma mensagem foi colocada na configuração da fila compartilhada original para acionamento. Isso é causado pela indicação excessiva de bits pela macro IXLVECTR, conforme documentado em O vetor de notificação de lista.

Controlando eventos acionadores

Controle os eventos acionadores usando alguns dos atributos que definem sua fila do aplicativo. Essas informações também fornecem exemplos do uso dos tipos de acionadores: EVERY, FIRST e DEPTH.

É possível ativar e desativar o acionamento e é possível selecionar o número ou a prioridade das mensagens que contam para um evento acionador. Há uma descrição integral desses atributos em Atributos de objetos.

Os atributos relevantes são:

TriggerControl
Use esse atributo para ativar e desativar o acionamento de uma fila do aplicativo.

TriggerMsgPriority
A prioridade mínima que uma mensagem deve ter para que conte com relação a um evento acionador. Se uma mensagem de prioridade menor que *TriggerMsgPriority* chegar à fila do aplicativo, o gerenciador de filas ignora a mensagem quando determina se deve criar uma mensagem do acionador. Se *TriggerMsgPriority* for configurado para zero, todas as mensagens contam com relação a um evento acionador.

TriggerType
Além do tipo de acionador NONE (que desativa o acionamento exatamente como configurar *TriggerControl* para OFF), é possível usar os tipos de acionadores a seguir para configurar a sensibilidade de uma fila para eventos acionadores:

EVERY
Um evento acionador ocorre toda vez que uma mensagem chega à fila do aplicativo. Use esse tipo de acionador se desejar diversas instâncias de um aplicativo iniciadas.

FIRST
Ocorre um evento acionador somente quando o número de mensagens na fila do aplicativo é mudado de zero para um. Use esse tipo de acionador se desejar que um programa de serviço seja iniciado quando a primeira mensagem chegar em uma fila, continue até que não haja mais mensagens a serem processadas, em seguida, termine. Deve-se sempre processar a fila até que esteja vazia. Consulte também “Caso especial de tipo de acionador FIRST” na página 889.

DEPTH
Ocorre um evento acionador apenas quando o número de mensagens na fila do aplicativo atingir o valor do atributo **TriggerDepth**. Um uso típico desse tipo de acionamento é iniciar um programa quando todas as respostas a um conjunto de solicitações forem recebidas.

Acionamento por profundidade: Ao acionar por profundidade, o gerenciador de filas desativa o acionamento (usando o atributo *TriggerControl*) após criar uma mensagem do acionador. Seu aplicativo deve reativar o próprio acionamento (usando a chamada MQSET) após isso ocorrer.

A ação de desativar o acionamento não está sob o controle do ponto de sincronização, portanto, o acionamento não pode ser reativado restaurando uma unidade de trabalho. Se um programa restaurar uma solicitação put que causou um evento acionador ou se o programa for encerrado de forma anormal, deve-se reativar o acionamento usando a chamada MQSET ou o comando ALTER QLOCAL.

TriggerDepth
O número de mensagens em uma fila que causa um evento acionador ao usar o acionamento por profundidade.

As condições que devem ser satisfeitas para um gerenciador de filas criar uma mensagem do acionador estão descritas em “Condições para um evento acionador” na página 883.

Exemplo do uso do tipo de acionador EVERY

Considere um aplicativo que gere solicitações para seguro de carro. O aplicativo pode enviar mensagens de solicitação a diversas seguradoras, especificando a mesma fila de resposta toda vez. Pode configurar um acionador do tipo EVERY nessa fila de resposta de forma que toda vez que uma resposta chegar, a resposta possa acionar uma instância do servidor para processar a resposta.

Exemplo do uso do tipo de acionador FIRST

Considere uma organização com inúmeras filiais, sendo que cada uma transmite detalhes dos negócios dos dias para a matriz. Todas elas fazem isso ao mesmo tempo, no fim do dia útil, e na matriz existe um aplicativo que processa os detalhes de todas as filiais. A primeira mensagem a chegar na matriz poderia causar um evento acionador que iniciaria esse aplicativo. Esse aplicativo continuaria o processamento até que não houvesse mais mensagens em sua fila.

Exemplo do uso do tipo de acionador DEPTH

Considere um aplicativo de agência de viagem que crie uma única solicitação para confirmar uma reserva de voo, confirmar uma reserva de um quarto de hotel, alugar um carro ou solicitar alguns traveler's checks. O aplicativo poderia separar esses itens em quatro mensagens de solicitação, enviando cada uma a um destino separado. Poderia configurar um acionador do tipo DEPTH em sua fila de resposta (com a profundidade configurada para o valor 4), de forma que seja reiniciado somente quando todas as quatro respostas tiverem chegado.

Se outra mensagem (possivelmente de uma solicitação diferente) chegar à fila de resposta antes da última das quatro respostas, o aplicativo de solicitação será acionado antecipadamente. Para evitar isso, ao usar o acionamento DEPTH para coletar diversas respostas a uma solicitação, sempre use uma nova fila de resposta para cada solicitação.

Caso especial de tipo de acionador FIRST

Com o tipo de acionador FIRST, se já houver uma mensagem na fila do aplicativo quando outra mensagem chegar, o gerenciador de filas normalmente não cria outra mensagem do acionador.

No entanto, o aplicativo que atende a fila pode não abrir a fila efetivamente (por exemplo, o aplicativo pode ser finalizado, possivelmente devido a um problema do sistema). Se um nome de aplicativo incorreto tiver sido colocado no objeto de definição de processo, o aplicativo que atende a fila não captará nenhuma das mensagens. Nessas situações, se outra mensagem chegar à fila do aplicativo, não haverá servidor em execução para processar essa mensagem (e qualquer outra mensagem na fila).

Para lidar com isso, o gerenciador de filas cria mensagens adicionais do acionador sob as circunstâncias a seguir:

- Se outra mensagem chegar na fila do aplicativo, mas somente se um intervalo de tempo predefinido tiver decorrido desde quando o gerenciador de filas criou a última mensagem do acionador para essa fila. Esse intervalo de tempo é definido no atributo do gerenciador de filas *TriggerInterval*. Seu valor padrão é 999 999 999 milissegundos.
- z/OS No IBM MQ for z/OS, filas do aplicativo que denominam uma fila de inicialização aberta são verificadas periodicamente. Se *TRIGINT* milissegundos tiverem passado desde o envio da última mensagem do acionador e a fila satisfizer as condições de um evento do acionador e CURDEPTH for maior que zero, uma mensagem do acionador será gerada. Esse processo é chamado de acionamento backstop.

Considere os pontos a seguir ao decidir sobre um valor para o intervalo do acionador a ser usado em seu aplicativo:

- Se você configurar *TriggerInterval* para um valor baixo e não houver nenhum aplicativo que atenda à fila do aplicativo, o tipo de acionador FIRST pode se comportar como o tipo de acionador EVERY. Isso depende da taxa em que as mensagens estão sendo colocadas na fila do aplicativo, que, por sua vez, pode depender de outra atividade do sistema. Isso ocorre porque, se o intervalo do acionador for muito pequeno, outra mensagem do acionador será gerada toda vez que uma mensagem for colocada na fila do aplicativo, embora o tipo de acionador seja FIRST, não EVERY. (O tipo de acionador FIRST com um intervalo do acionador igual a zero é equivalente ao tipo de acionador EVERY.)
- z/OS No IBM MQ for z/OS, se você configurar *TRIGINT* para um valor baixo e não houver nenhum aplicativo que atenda à fila do aplicativo com tipo de acionador FIRST, o acionamento backstop irá gerar uma mensagem do acionador toda vez que a verificação periódica de filas de aplicativos que denominam filas de inicialização abertas ocorrer.
- Se uma unidade de trabalho for restaurada (consulte Mensagens do acionador e unidades de trabalho) e o intervalo do acionador tiver sido configurado para um valor alto (ou o valor padrão), uma mensagem do acionador será gerada quando a unidade de trabalho for restaurada. No entanto, se você tiver configurado o intervalo do acionador para um valor baixo ou para zero (fazendo com que o tipo de acionador FIRST se comporte como o tipo de acionador EVERY), muitas mensagens do acionador poderão ser geradas. Se a unidade de trabalho for restaurada, todas as mensagens do acionador ainda serão disponibilizadas. O número de mensagens do acionador que são geradas depende do intervalo do

acionador. Se o intervalo do acionador for configurado para zero, o número máximo de mensagens será gerado.

Projetando um aplicativo que usa filas acionadas

Você viu como configurar, controlar e acionar seus aplicativos. Aqui estão algumas dicas a se considerar ao projetar seu aplicativo.

Mensagens do acionador e unidades de trabalho

As mensagens do acionador criadas devido a eventos acionadores que não são parte de uma unidade de trabalho são colocadas na fila de inicialização, fora de qualquer unidade de trabalho, sem dependência de quaisquer outras mensagens e estão disponíveis para recuperação pelo monitor acionador imediatamente.

As mensagens do acionador criadas devido a eventos do acionador que fazem parte de uma unidade de trabalho são disponibilizadas na fila de inicialização quando a UOW é resolvida se a unidade de trabalho for confirmada ou restaurada

Se o gerenciador de filas falhar ao colocar uma mensagem do acionador em uma fila de inicialização, ele será colocado na fila de devoluções (mensagem não entregue).

Nota:

1. O gerenciador de filas conta mensagens confirmadas e não confirmadas quando ele avalia se as condições para um evento acionador existem.

   Com o acionamento do tipo FIRST ou DEPTH, as mensagens do acionador são disponibilizadas, mesmo se a unidade de trabalho for restaurada de modo que uma mensagem do acionador esteja sempre disponível quando as condições necessárias forem atendidas. Por exemplo, considere uma solicitação put em uma unidade de trabalho para uma fila que é acionada com tipo de acionador FIRST. Isso faz com que o gerenciador de filas crie uma mensagem do acionador. Se uma outra solicitação put ocorrer, a partir de outra unidade de trabalho, isso não causa outro evento acionador, porque o número de mensagens na fila do aplicativo foi mudado de um para dois, o que não atende às condições de um evento acionador. Agora, se a primeira unidade de trabalho for restaurada, mas a segunda for confirmada, uma mensagem do acionador ainda será criada.

   No entanto, isso significa que as mensagens do acionador às vezes são criadas quando as condições para um evento acionador não são satisfeitas. Aplicativos que usam o acionamento devem sempre estar preparados para lidar com esta situação. É recomendado que você use a opção aguardar com a chamada MQGET, configurando o *WaitInterval* para um valor adequado.

   Mensagens do acionador criadas são sempre disponibilizadas se a unidade de trabalho estiver restaurada ou confirmada.
2. Para filas compartilhadas locais (ou seja, filas compartilhadas em um grupo de filas compartilhadas), o gerenciador de filas conta apenas mensagens confirmadas.

Obtendo mensagens de uma fila acionada

Ao projetar aplicativos que usam o acionamento, esteja ciente de que pode haver um atraso entre um monitor acionador iniciando um programa e outras mensagens se tornando disponíveis na fila do aplicativo. Isso pode acontecer quando a mensagem que causa o evento acionador é confirmada antes das outras.

Para dar tempo de as mensagens chegarem, sempre use a opção de espera ao usar a chamada MQGET para remover mensagens de uma fila para a qual as condições acionadoras são configuradas. O *WaitInterval* deve ser suficiente para permitir o tempo mais longo razoável entre uma mensagem sendo colocada e essa chamada put sendo confirmada. Se a mensagem estiver chegando a partir de um gerenciador de filas remotas, esse tempo será afetado por:

- O número de mensagens que são colocadas antes de serem confirmadas
- A velocidade e a disponibilidade do link de comunicação

- Os tamanhos das mensagens

Para obter um exemplo de uma situação em que é necessário usar a chamada MQGET com a opção de espera, considere o mesmo exemplo que usamos ao descrever unidades de trabalho. Essa era uma solicitação put em uma unidade de trabalho para uma fila que é acionada com tipo de acionador FIRST. Esse evento faz com que o gerenciador de filas crie uma mensagem do acionador. Se uma outra solicitação put ocorrer, a partir de outra unidade de trabalho, isso não causa outro evento acionador porque o número de mensagens na fila do aplicativo não foi mudado de zero para um. Agora, se a primeira unidade de trabalho for restaurada, mas a segunda for confirmada, uma mensagem do acionador ainda será criada. Portanto, a mensagem do acionador é criada no momento em que a primeira unidade de trabalho é restaurada. Se houver um atraso significativo antes de a segunda mensagem ser confirmada, o aplicativo acionado poderá precisar esperar por ela.

Com o acionamento do tipo DEPTH, um atraso pode ocorrer mesmo se todas as mensagens relevantes forem finalmente confirmadas. Digamos que o atributo de fila **TriggerDepth** tenha o valor 2. Quando duas mensagens chegam na fila, a segunda faz com que uma mensagem acionadora seja criada. No entanto, se a segunda mensagem for a primeira a ser confirmada, será nessa altura que a mensagem do acionador se tornará disponível. O monitor acionador inicia o programa do servidor, mas o programa pode recuperar apenas a segunda mensagem até que a primeira seja confirmada. Portanto, o programa pode precisar esperar que a primeira mensagem seja disponibilizada.

Projete seu aplicativo para que ele seja finalizado, se nenhuma mensagem estiver disponível para recuperação quando seu intervalo de espera expirar. Se uma ou mais mensagens chegarem posteriormente, conte com o aplicativo que estava sendo acionado novamente para processá-las. Esse método evita que os aplicativos fiquem inativos e o uso de recursos desnecessariamente.

Processamento da fila de inicialização por monitores acionadores

Para um gerenciador de filas, um monitor acionador é semelhante a qualquer outro aplicativo que atende uma fila. No entanto, um monitor acionador serve filas de inicialização.

Um monitor acionador é geralmente um programa de execução contínua. Quando uma mensagem do acionador chega em uma fila de inicialização, o monitor acionador recupera essa mensagem. Ele utiliza as informações na mensagem para emitir um comando para iniciar o aplicativo que deve processar as mensagens na fila do aplicativo.

O monitor acionador deve transmitir informações suficientes para o programa que está iniciando para que o programa possa executar as ações corretas na fila do aplicativo correto.

Um iniciador de canal é um exemplo de um tipo especial de monitor acionador para agentes do canal de mensagem. Nessa situação, no entanto, deve-se usar o tipo de acionador FIRST ou DEPTH.

ALW *Acione monitores em sistemas AIX, Linux, and Windows*

Este tópico contém informações sobre monitores acionadores fornecidos em sistemas AIX, Linux, and Windows.

Os monitores acionadores a seguir são fornecidos para o ambiente do servidor:

amqstrg0
: Este é um monitor acionador de amostra que fornece um subconjunto da função fornecida pelo **runmqtrm**. Consulte “Usando os programas de amostra em multiplataformas” na página 1071 para obter mais informações sobre amqstrg0.

runmqtrm
: A sintaxe desse comando é **runmqtrm** [*-m QMgrName*] [*-q InitQ*], em que QMgrName é o gerenciador de filas e InitQ é a fila de inicialização. A fila padrão é SYSTEM.DEFAULT.INITIATION.QUEUE no gerenciador de filas padrão. Ela chama programas para as mensagens do acionador apropriadas. Esse monitor acionador suporta o tipo de aplicativo padrão.

  A sequência de comandos transmitida pelo monitor acionador para o sistema operacional é construída da forma a seguir:

1. O *ApplId* na definição de PROCESS relevante (se criado)
2. A estrutura MQTMC2, colocada em aspas duplas
3. O *EnvData* na definição de PROCESS relevante (se criado)

em que *ApplId* é o nome do programa a ser executado como seria inserido na linha de comandos.

O parâmetro transmitido é a estrutura de caracteres MQTMC2. Uma sequência de comandos que possui esta sequência é chamada, exatamente conforme fornecida, entre aspas duplas, na ordem em que o comando do sistema a aceitará como um parâmetro.

O monitor acionador não verifica se há outra mensagem na fila de inicialização até a conclusão do aplicativo que acabou de iniciar. Se o aplicativo tiver muito processamento a fazer, o monitor acionador não poderá acompanhar o número de mensagens do acionador que chegarem. Você tem duas opções:

- Ter mais monitores acionadores em execução
- Executar os aplicativos iniciados em segundo plano

Se você tiver mais monitores acionadores em execução, será possível controlar o número máximo de aplicativos que podem ser executados a qualquer momento. Se você executar aplicativos em segundo plano, não haverá restrição imposta pelo IBM MQ no número de aplicativos que podem ser executados.

Linux AIX Para executar o aplicativo iniciado no plano de fundo no AIX and Linux, coloque um & no final do *EnvData* da definição PROCESS.

Para executar o aplicativo iniciado em segundo plano em sistemas Windows, no campo *ApplId*, prefixe o nome de seu aplicativo com um comando START. Por exemplo:

```
START ?B AMQSECHA
```

Nota: Windows Quando um caminho do Windows tiver espaços como parte do nome do caminho, eles deverão ser colocados entre aspas (") para assegurar que ele seja tratado como um único argumento. Por exemplo, "`C:\Program Files\Application Directory\Application.exe`".

A seguir, está um exemplo de uma sequência APPLICID em que o nome do arquivo inclui espaços como parte do caminho:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

A sintaxe do comando Windows START no exemplo inclui uma sequência vazia entre aspas duplas. START especifica que o primeiro argumento entre aspas será tratado como o título do novo comando. Para assegurar que o Windows não erre o caminho do aplicativo para um argumento 'title', inclua uma sequência de título entre aspas duplas no comando antes do nome do aplicativo.

Os monitores acionadores a seguir são fornecidos para o cliente IBM MQ:

runmqtmc
É o mesmo que runmqtrm, exceto que ele se vincula com as bibliotecas do IBM MQ MQI client.

ALW *Monitor acionador para o CICS*

O monitor acionador amqltmc0 é fornecido para o CICS. Ele funciona da mesma maneira que o monitor acionador padrão, runmqtrm, mas é executado de maneira diferente e aciona transações do CICS.

Este tópico aplica-se apenas a sistemas Windows, AIX and Linux x86-64 .

O monitor acionador é fornecido como um programa CICS; defina-o com um nome de transação de 4 caracteres. Insira o nome de 4 caracteres para iniciar o monitor acionador. Ele usa o gerenciador de filas

padrão (conforme nomeado no arquivo qm.ini ou, no IBM MQ for Windows, no registro) e o SISTEMA SYSTEM.CICS.INITIATION.QUEUE.

Se desejar usar um gerenciador de filas ou uma fila diferente, construa a estrutura MQTMC2 do monitor acionador; isso requer que você escreva um programa usando a chamada EXEC CICS START, pois a estrutura é muito longa para incluir como um parâmetro. Em seguida, passe a estrutura MQTMC2 como dados para a solicitação START para o monitor acionador.

Ao usar a estrutura MQTMC2, é necessário fornecer somente os parâmetros *StrucId*, *Version*, *QName*e **QMgrName** para o monitor acionador conforme, pois ele não faz referência a nenhum outro campo.

As mensagens são lidas na fila de iniciação e usadas para iniciar as transações do CICS com EXEC CICS START, assumindo que o APPL_TYPE na mensagem do acionador é MQAT_CICS. A leitura de mensagens da fila de inicialização é executada sob o controle do ponto de sincronização do CICS.

As mensagens são geradas quando o monitor é iniciado e parado e quando ocorre um erro. Essas mensagens são enviadas à fila de dados temporários CSMT.

Tabela 129. Versões disponíveis do monitor acionador.

Uma tabela com duas colunas. A primeira coluna lista as versões disponíveis do monitor acionador e a segunda coluna mostra em quais plataformas cada versão é usada.

Versão	Uso
amqltmc0	TXSeries para: • AIX AIX • Linux Sistemas Linux x86-64
amqltmc4	Windows TXSeries para Windows 5.1
amqltmcc	Versão ligada ao cliente do monitor acionador do CICS
V 9.4.0 V 9.4.0 amqltmc064	TXSeries de 64 bits para sistemas Linux x86-64
V 9.4.0 V 9.4.0 amqltmcc64	Versão cliente de amqltmc064

Se precisar de um monitor acionador para outros ambientes, escreva um programa que possa processar as mensagens do acionador que o gerenciador de filas coloca nas filas de inicialização. Esse programa deve executar as ações a seguir:

1. Use a chamada MQGET para esperar a chegada de uma mensagem na fila de inicialização.
2. Examine os campos da estrutura MQTM da mensagem do acionador para localizar o nome do aplicativo a ser iniciado e o ambiente no qual ele é executado.
3. Emita um comando inicial específico do ambiente.

   z/OS Por exemplo, no lote do z/OS, envie uma tarefa para o leitor interno.
4. Converta a estrutura MQTM para a estrutura MQTMC2, se necessário.
5. Passe a estrutura MQTMC2 ou MQTM para o aplicativo iniciado. Ela pode conter dados do usuário.
6. Associe à sua fila do aplicativo o aplicativo que deve atender àquela fila. É possível fazer isso denominando o objeto de definição de processo (se criado) no atributo **ProcessName** da fila. Para nomear o objeto de definição de processo, é possível utilizar o comando **DEFINE QLOCAL** ou **ALTER QLOCAL**

   IBM i No IBM i, também é possível usar CRTMQMQ ou CHGMQMQ.

Para obter mais informações sobre a interface do monitor acionador, consulte MQTMC2.

IBM i *Monitores acionadores no IBM i*

No IBM i, em vez do comando de controle **runmqtrm**, use o comando CL do IBM MQ for IBM i **STRMQMTRM**.

Use o comando STRMQMTRM da seguinte forma:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

Os detalhes são como para runmqtrm.

Os programas de amostra a seguir também são fornecidos, os quais é possível usar como modelos para gravar seus próprios monitores acionadores:

AMQSTRG4
Este é um monitor acionador que envia uma tarefa do IBM i para o processo que deve ser iniciado, mas isso significa que existe processamento adicional associado a cada mensagem do acionador.

AMQSERV4
Esse é um servidor acionador. Para cada mensagem do acionador, esse servidor executa o comando para o processo em sua própria tarefa e pode chamar transações do CICS.

Tanto o monitor acionador quanto o servidor acionador transmitem uma estrutura MQTMC2 para os programas que eles iniciam. Para obter uma descrição dessa estrutura, consulte MQTMC2. Duas destas amostras são fornecidas nas formas de origem e executável.

Como esses monitores acionadores podem chamar somente programas nativos do IBM i, eles não podem acionar programas Java diretamente, porque as classes Java estão localizadas no IFS. No entanto, programas Java podem ser acionados indiretamente acionando um programa CL que, em seguida, chama o programa Java e passa pela estrutura TMC2. O tamanho mínimo da estrutura TMC2 é 732 bytes.

Aqui está a origem de um CLP de amostra:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

O programa do monitor acionador a seguir é fornecido para o IBM MQ MQI client: RUNMQTMC

Chame o RUNMQTMC conforme a seguir:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

Propriedades de mensagens do acionador

Os tópicos a seguir descrevem algumas outras propriedades de mensagens do acionador.

- “Persistência e prioridade de mensagens do acionador” na página 894
- “Reinicialização do gerenciador de filas e mensagens do acionador” na página 895
- “Mensagens do acionador e mudanças nos atributos do objeto” na página 895
- “Formato de mensagens do acionador” na página 895

Persistência e prioridade de mensagens do acionador

Mensagens do acionador não são persistentes porque não há requisito para que sejam assim.

No entanto, as condições para gerar eventos de acionamento persistem, de modo que as mensagens do acionador são geradas sempre que essas condições forem atendidas. Se uma mensagem do acionador for perdida, a existência continuada da mensagem do aplicativo na fila do aplicativo garante que o gerenciador de filas gere uma mensagem de acionador assim que todas as condições forem atendidas.

Se uma unidade de trabalho for retrocedida, todas as mensagens do acionador que ela tiver gerado sempre serão entregues.

As mensagens do acionador usam a prioridade padrão da fila de inicialização.

Reinicialização do gerenciador de filas e mensagens do acionador

Após a reinicialização de um gerenciador de filas, quando uma fila de inicialização for aberta da próxima vez para entrada, uma mensagem do acionador poderá ser colocada nessa fila de inicialização se uma fila do aplicativo associada a ele tiver mensagens e estiver definida para acionamento.

Mensagens do acionador e mudanças nos atributos do objeto

Mensagens do acionador são criadas de acordo com os valores dos atributos do acionador em vigor no momento do evento acionador.

Se a mensagem do acionador não for disponibilizada para um monitor acionador até mais tarde (porque a mensagem que causou sua geração foi colocada em uma unidade de trabalho), quaisquer mudanças nos atributos do acionador não terão efeito na mensagem do acionador enquanto isso. Especificamente, desativar o acionamento não evita que uma mensagem do acionador seja disponibilizada quando for criada. Além disso, a fila do aplicativo pode não existir mais no momento que a mensagem do acionador for disponibilizada.

Formato de mensagens do acionador

O formato de uma mensagem do acionador é definido pela estrutura MQTM.

Ela tem os campos a seguir, que o gerenciador de filas preenche ao criar a mensagem do acionador, usando informações nas definições de objeto da fila do aplicativo e do processo associado a essa fila:

StrucId
O identificador de estruturação.

Version
A versão da estrutura.

QName
O nome da fila do aplicativo na qual o evento acionador ocorreu. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **QName** da fila do aplicativo.

ProcessName
O nome do objeto de definição de processo associado à fila do aplicativo. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **ProcessName** da fila do aplicativo.

TriggerData
Um campo de formato livre para ser usado pelo monitor acionador. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **TriggerData** da fila do aplicativo. No IBM MQ for Multiplatforms, esse campo pode ser usado para especificar o nome do canal a ser acionado..

ApplType
O tipo do aplicativo que o monitor acionador deve iniciar. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **ApplType** do objeto de definição de processo identificado em *ProcessName*.

ApplId
Uma sequência de caracteres que identifica o aplicativo que o monitor acionador deve iniciar. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **ApplId** do objeto de definição de processo identificado em *ProcessName*.

Quando você usa o monitor acionador CKTI, fornecido por CICS, o atributo **ApplId** do objeto de definição do processo é um identificador de transação CICS.

z/OS Quando você usa o CSQQTRMN fornecido por IBM MQ for z/OS, o atributo **ApplId** do objeto de definição do processo é um identificador de transação IMS.

EnvData
Um campo de caractere que contém dados relacionados ao ambiente para uso pelo monitor acionador. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **EnvData** do objeto de definição de processo identificado em *ProcessName*. O CICS-fornecido monitor acionador (CKTI) ou o IBM MQ for z/OS-fornecido monitor acionador (CSQQTRMN) não usa esse campo, mas outros monitores acionador podem optar por usá-lo.

UserData
Um campo de caractere que contém dados do usuário para uso pelo monitor acionador. Quando o gerenciador de filas cria uma mensagem do acionador, ele preenche esse campo usando o atributo **UserData** do objeto de definição de processo identificado em *ProcessName*. Esse campo pode ser usado para especificar o nome do canal a ser acionado.

Há uma descrição integral da estrutura da mensagem do acionador em MQTM.

Quando o acionamento não funciona

Um programa não é acionado se o monitor acionador não puder iniciar o programa ou o gerenciador de filas não puder entregar a mensagem do acionador. Por exemplo, o applid no objeto do processo deve especificar que o programa deve ser iniciado em segundo plano; caso contrário, o monitor acionador não pode iniciar o programa.

Se uma mensagem do acionador for criada, mas não puder ser colocada na fila de inicialização (por exemplo, porque a fila está cheia ou o comprimento da mensagem do acionador é maior que o comprimento máximo de mensagem especificado para a fila de inicialização), a mensagem do acionador será colocada na fila de devoluções (da mensagem não entregue).

Se a operação put na fila de mensagens não entregues não puder ser concluída com êxito, a mensagem do acionador será descartada e uma mensagem de aviso será enviada z/OS para o console do z/OS ou para o operador do sistema ou será colocada no log de erros.

Colocar a mensagem do acionador na fila de mensagens não entregues pode gerar uma mensagem do acionador para essa fila. Essa segunda mensagem do acionador será descartada se incluir uma mensagem na fila de mensagens não entregues.

Se o programa for acionado com êxito, mas for finalizado de forma anormal antes de receber a mensagem da fila, use um utilitário de rastreio (por exemplo, CICS AUXTRACE se o programa estiver em execução no CICS) para localizar a causa da falha.

Trabalhando com MQI e clusters

Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

Use os links a seguir para descobrir mais sobre as opções disponíveis nas chamadas e nos códigos de retorno para uso com clusters:

- “MQOPEN e clusters” na página 897
- “MQPUT, MQPUT1 e clusters” na página 898
- “MQINQ e clusters” na página 899
- “MQSET e clusters” na página 899
- “Códigos de retorno” na página 900

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 747

Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

"Abrindo e fechando objetos" na página 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

"Colocando mensagens em uma fila" na página 764
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" na página 780
Use estas informações para aprender como obter mensagens de uma fila.

"Consultando e configurando atributos de objeto" na página 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

"Confirmando e fazendo backup de unidades de trabalho" na página 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

"Iniciando aplicativos IBM MQ usando acionadores" na página 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Using and writing applications on IBM MQ for z/OS" na página 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

"IMS and IMS bridge applications on IBM MQ for z/OS" na página 71
This information helps you to write IMS applications using IBM MQ.

MQOPEN e clusters

A fila na qual uma mensagem é colocada, ou a partir da qual é lida, quando uma fila de clusters é aberta depende da chamada MQOPEN.

Selecionando a fila de destino

Se não fornecer um nome de gerenciador de filas no descritor de objeto, MQOD, o gerenciador de filas seleciona o gerenciador de filas para o qual enviar a mensagem. Se você fornecer um nome de gerenciador de filas no descritor de objeto, então, mensagens sempre serão enviadas ao gerenciador de filas selecionado.

Se o gerenciador de filas estiver selecionando o gerenciador de filas de destino, a seleção dependerá das opções de ligação, MQOO_BIND_* e se uma fila local existe. Se houver uma instância local da fila, sempre será aberta preferencialmente a uma instância remota, a menos que o atributo CLWLUSEQ esteja configurado para ANY. Caso contrário, a seleção depende das opções de ligação. MQOO_BIND_ON_OPEN ou MQOO_BIND_ON_GROUP deve ser especificado ao usar grupos de mensagens com clusters para assegurar que todas as mensagens no grupo sejam processadas no mesmo destino.

Se o gerenciador de filas estiver selecionando o gerenciador de filas de destino, faz isso de modo round-robin, usando o algoritmo de gerenciamento de carga de trabalho; consulte Balanceamento de carga de trabalho em clusters.

Quando o algoritmo de balanceamento de carga de trabalho é usado, depende da maneira como a fila de clusters é aberta:

- MQOO_BIND_ON_OPEN – o algoritmo é usado uma vez quando a fila é aberta pelo aplicativo.
- MQOO_BIND_NOT_FIXED – o algoritmo é usado para cada mensagem colocada na fila.
- MQOO_BIND_ON_GROUP – o algoritmo é usado uma vez no início de cada grupo de mensagens.

MQOO_BIND_ON_OPEN
A opção MQOO_BIND_ON_OPEN na chamada MQOPEN especifica que o gerenciador de filas de destino deve ser fixo. Use a opção MQOO_BIND_ON_OPEN se houver várias instâncias da mesma fila em um cluster. Todas as mensagens colocadas na fila especificando a manipulação de objetos retornada da chamada MQOPEN são direcionadas para o mesmo gerenciador de filas.

- Use a opção MQOO_BIND_ON_OPEN se as mensagens tiverem afinidades. Por exemplo, se um lote de mensagens for ser totalmente processado pelo mesmo gerenciador de filas, especifique MQOO_BIND_ON_OPEN ao abrir a fila. O IBM MQ fixa o gerenciador de filas e a rota a ser tomada por todas as mensagens colocadas nessa fila.
- Se a opção MQOO_BIND_ON_OPEN for especificada, a fila deverá ser reaberta para uma nova instância da fila a ser selecionada.

MQOO_BIND_NOT_FIXED
A opção MQOO_BIND_NOT_FIXED na chamada MQOPEN especifica que o gerenciador de filas de destino não é fixo. As mensagens gravadas na fila especificando a manipulação de objetos retornada da chamada MQOPEN são roteadas para um gerenciador de filas no momento de MQPUT mensagem por mensagem. Use a opção MQOO_BIND_NOT_FIXED se não desejar forçar que todas as suas mensagens sejam gravadas no mesmo destino.

- Não especifique MQOO_BIND_NOT_FIXED e MQMF_SEGMENTATION_ALLOWED ao mesmo tempo. Se fizer isso, os segmentos de sua mensagem poderão ser entregues a diferentes gerenciadores de filas, espalhados por todo o cluster.

MQOO_BIND_ON_GROUP
Permite que um aplicativo solicite que um grupo de mensagens seja alocado para a mesma instância de destino. Essa opção é válida somente para filas e afeta somente filas de clusters. Se especificada para uma fila que não seja uma fila de cluster, a opção será ignorada.

- Grupos são roteados para um único destino somente quando MQPMO_LOGICAL_ORDER é especificado na chamada MQPUT. Quando MQOO_BIND_ON_GROUP é especificado, mas uma mensagem não faz parte de um grupo lógico, o comportamento de BIND_NOT_FIXED será usado em seu lugar.

MQOO_BIND_AS_Q_DEF
Se você não especificar MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED ou MQOO_BIND_ON_GROUP, a opção padrão é MQOO_BIND_AS_Q_DEF. Usar MQOO_BIND_AS_Q_DEF faz com que a ligação usada para o identificador de filas seja obtida do atributo da fila DefBind.

Relevância de opções MQOPEN

As opções de MQOPEN MQOO_BROWSE, MQOO_INPUT_* ou MQOO_SET requerem uma instância local da fila de clusters para que MQOPEN obtenha sucesso.

As opções de MQOPEN MQOO_OUTPUT, MQOO_BIND_* ou MQOO_INQUIRE não requerem uma instância local da fila de clusters para obter êxito.

Nome do Gerenciador de Filas Resolvido

Quando um nome do gerenciador de filas é resolvido no momento de MQOPEN, o nome resolvido é retornado ao aplicativo. Se o aplicativo tentar usar esse nome em uma chamada MQOPEN subsequente, ele poderá achar que não está autorizado a acessar o nome.

MQPUT, MQPUT1 e clusters

Se MQOO_BIND_NOT_FIXED for especificado em um MQOPEN, as rotinas de gerenciamento de carga de trabalho escolherão qual destino MQPUT ou MQPUT1 selecionar.

Se MQOO_BIND_NOT_FIXED for especificado em uma chamada MQOPEN, cada chamada MQPUT subsequente chamará a rotina de gerenciamento de carga de trabalho para determinar para qual gerenciador de filas enviar a mensagem. O destino e a rota a serem obtidos são selecionados em uma base mensagem por mensagem. O destino e a rota poderão mudar após a mensagem ter sido colocada se as condições na rede mudarem. A chamada MQPUT1 sempre opera como se MQOO_BIND_NOT_FIXED estivesse em vigor, ou seja, sempre chama a rotina de gerenciamento de carga de trabalho.

Quando a rotina de gerenciamento de carga de trabalho tiver selecionado um gerenciador de filas, o gerenciador de filas local concluirá a operação de colocação. A mensagem pode ser colocada em filas diferentes:

1. Se o destino for a instância local da fila, a mensagem será colocada na fila local.
2. Se o destino for um gerenciador de filas em um cluster, a mensagem será colocada em uma fila de transmissão do cluster.
3. Se o destino for um gerenciador de filas fora de um cluster, a mensagem será colocada em uma fila de transmissão com o mesmo nome que o gerenciador de filas de destino.

Se MQOO_BIND_ON_OPEN for especificado na chamada MQOPEN, as chamadas MQPUT não chamarão a rotina de gerenciamento de carga de trabalho porque o destino e a rota já foram selecionados.

MQINQ e clusters

Qual fila de clusters é consultada depende das opções que forem combinadas com MQOO_INQUIRE.

Antes de poder inquirir sobre uma fila, abra-a usando a chamada MQOPEN e especifique MQOO_INQUIRE.

Para inquirir sobre uma fila de clusters, use a chamada MQOPEN e combine outras opções com MQOO_INQUIRE. Os atributos que podem ser inquiridos dependem se há uma instância local da fila de clusters e de como a fila é aberta:

- Combinar MQOO_BROWSE, MQOO_INPUT_* ou MQOO_SET com MQOO_INQUIRE requer uma instância local da fila de clusters para que a abertura seja bem-sucedida. Neste caso, é possível consultar sobre todos os atributos que são válidos para filas locais.
- Combinar MQOO_OUTPUT com MQOO_INQUIRE e não especificar nenhuma das opções anteriores, a instância aberta será:
  - A instância no gerenciador de filas locais, se houver uma. Neste caso, é possível consultar sobre todos os atributos que são válidos para filas locais.
  - Uma instância em outro lugar no cluster, se não houver nenhuma instância do gerenciador de filas locais. Neste caso apenas os seguintes atributos podem ser consultados. O atributo QType tem o valor MQQT_CLUSTER nesse caso.
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

Para inquirir sobre o atributo DefBind de uma fila de clusters, use a chamada MQINQ com o seletor MQIA_DEF_BIND. O valor retornado é MQBND_BIND_ON_OPEN ou MQBND_BIND_NOT_FIXED ou MQBND_BIND_ON_GROUP. QualquerMQBND_BIND_ON_OPEN ouMQBND_BIND_ON_GROUP deve ser especificado ao usar grupos com clusters.

Para inquirir sobre os atributos CLUSTER e CLUSNL da instância local de uma fila, use a chamada MQINQ com o seletor MQCA_CLUSTER_NAME ou o seletor MQCA_CLUSTER_NAMELIST.

Nota: Se uma fila de clusters for aberta sem corrigir a fila ligada ao MQOPEN, sucessivas chamadas MQINQ podem consultar diferentes instâncias da fila de clusters.

Conceitos relacionados

"Opção MQOPEN para fila de cluster" na página 760
A ligação usada para a manipulação de fila é obtida a partir do atributo da fila **DefBind**, que pode obter o valor MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED ou MQBND_BIND_ON_GROUP.

MQSET e clusters

A opção MQOO_SET da opção MQOPEN requer que haja uma instância local de uma fila de clusters para que MQSET seja bem-sucedido.

Não é possível usar a chamada MQSET para configurar os atributos de uma fila em qualquer outra parte do cluster.

É possível abrir um alias local ou fila remota definida com o atributo do cluster e usar a chamada MQSET. É possível configurar os atributos do alias local ou a fila remota. Não importa se a fila de destino for uma fila de cluster definida em um gerenciador de filas diferente.

Códigos de retorno

Códigos de retorno específicos para clusters

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

Uma chamada MQOPEN, MQPUT ou MQPUT1 é emitida para abrir uma fila de clusters ou para colocar uma mensagem nela. A saída de carga de trabalho do cluster, definida pelo atributo ClusterWorkloadExit de um gerenciador de filas, falha inesperadamente ou não responde a tempo.

z/OS Uma mensagem é gravada no log do sistema no IBM MQ for z/OS fornecendo mais informações sobre este erro.

Chamadas subsequentes MQOPEN, MQPUT e MQPUT1 para esse manipulador de filas são processadas como se o atributo ClusterWorkloadExit estivesse em branco.

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

z/OS No z/OS, a saída de carga de trabalho do cluster não pode ser carregada.

Uma mensagem é gravada no registro do sistema e o processamento continuará como se o atributo ClusterWorkloadExit estivesse em branco.

Multi Em Multiplataformas, uma chamada MQCONN ou MQCONNX é emitida para conectar-se a um gerenciador de filas. A chamada falha porque a saída de carga de trabalho do cluster, definida pelo atributo ClusterWorkloadExit do gerenciador de filas, não pode ser carregada.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Uma chamada MQOPEN com as opções MQOO_OUTPUT e MQOO_BIND_ON_OPEN é efetivamente emitida para uma fila de clusters. Todas as instâncias da fila no cluster têm atualmente o put inibido ao ter o atributo InhibitPut configurado para MQQA_PUT_INHIBITED. Como não há instâncias de fila disponíveis para receber mensagens, a chamada MQOPEN falhará.

Esse código de razão ocorre somente quando ambas as instruções a seguir são verdadeiras:

- Não há ocorrência local da fila. Se houver uma ocorrência local, a chamada MQOPEN terá êxito mesmo se a ocorrência local estiver com o put inibido.
- Não há saída de carga de trabalho do cluster para a fila ou há uma saída de carga de trabalho do cluster mas ela não escolhe uma instância de fila. (Se a saída da carga de trabalho do cluster escolher uma instância de fila, a chamada MQOPEN terá êxito, mesmo se a ocorrência estiver com put inibido.)

Se a opção MQOO_BIND_NOT_FIXED for especificada na chamada MQOPEN, a chamada poderá ser bem-sucedida mesmo se todas as filas no cluster estiverem com put inibido. No entanto, uma chamada subsequente MQPUT pode falhar se todas as filas ainda estiverem com put inibido no horário dessa chamada.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Uma chamada MQOPEN, MQPUT ou MQPUT1 é emitida para abrir uma fila de clusters ou para colocar uma mensagem nela. A definição da fila não pode ser resolvida corretamente porque é necessária uma resposta do gerenciador de filas de repositório completo, mas nenhum está disponível.
2. Uma chamada MQOPEN, MQPUT, MQPUT1 ou MQSUB é emitida para um objeto de tópico especificando PUBSCOPE (ALL) ou SUBSCOPE (ALL). A definição de tópico de cluster não pode ser resolvida corretamente porque uma resposta é necessária do gerenciador de filas do repositório completo, mas nenhuma está disponível.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Uma chamada MQOPEN, MQPUT ou MQPUT1 é emitida para uma fila de clusters. Um erro ocorre durante a tentativa de usar um recurso requerido para clusterização.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Uma chamada MQPUT ou MQPUT1 é emitida para colocar uma mensagem em uma fila de clusters. Na hora da chamada, não há mais nenhuma instância da fila no cluster. O MQPUT falha e a mensagem não é enviada.

O erro pode ocorrer se MQOO_BIND_NOT_FIXED for especificado na chamada MQOPEN que abre a fila ou MQPUT1 for usado para colocar a mensagem.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Uma chamada MQOPEN, MQPUT ou MQPUT1 é emitida para abrir ou colocar uma mensagem em uma fila de clusters. A saída de carga de trabalho do cluster rejeita a chamada.

z/OS Using and writing applications on IBM MQ for z/OS

IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

This information explains the IBM MQ facilities available to programs running in each of the supported environments. In addition,

- For information about using the IBM MQ-CICS bridge, see Using IBM MQ with CICS .
- For information about using IMS and the IMS bridge, see “IMS and IMS bridge applications on IBM MQ for z/OS” on page 71.

Use the following links to find out more about using and writing applications on IBM MQ for z/OS:

- “Environment-dependent IBM MQ for z/OS functions” on page 902
- “Debugging facilities, syncpoint support, and recovery support” on page 902
- “The IBM MQ for z/OS interface with the application environment” on page 903
- “Writing z/OS UNIX System Services applications” on page 905
- “Application programming with shared queues” on page 908

Related concepts

“Visão geral da Message Queue Interface” on page 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” on page 747
Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

“Abrindo e fechando objetos” on page 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” on page 764
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” on page 780
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” on page 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” on page 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

“Iniciando aplicativos IBM MQ usando acionadores” on page 877

Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" on page 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"IMS and IMS bridge applications on IBM MQ for z/OS" on page 71
This information helps you to write IMS applications using IBM MQ.

z/OS *Environment-dependent IBM MQ for z/OS functions*

Use this information when considering IBM MQ for z/OS functions.

The main differences to be considered between IBM MQ functions in the environments in which IBM MQ for z/OS runs are:

- IBM MQ for z/OS supplies the following trigger monitors:
  - CKTI for use in the CICS environment
  - CSQQTRMN for use in the IMS environment

  You must write your own module to start applications in other environments.
- Syncpointing using two-phase commit is supported in the CICS and IMS environments. It is also supported in the z/OS batch environment using transaction management and recoverable resource manager services (RRS). Single-phase commit is supported in the z/OS environment by IBM MQ itself.
- For the batch and IMS environments, the MQI provides calls to connect programs to, and to disconnect them from, a queue manager. Programs can connect to more than one queue manager.
- A CICS system can connect to only one queue manager. This can be made to happen when CICS is initiated if the subsystem name is defined in the CICS system startup job. The MQI connect and disconnect calls are tolerated, but have no effect, in the CICS environment.
- The API-crossing exit allows a program to intervene in the processing of all MQI calls. This exit is available in the CICS environment only.
- In CICS on multiprocessor systems, some performance advantage is gained because MQI calls can be executed under multiple z/OS TCBs. For more information, see the Planejando em z/OS . *IBM MQ for z/OS Concepts and Planning Guide*.

These features are summarized in Table 130 on page 902.

Table 130. z/OS environmental features

	CICS	**IMS**	**Batch/TSO**
Trigger monitor supplied	Yes	Yes	No
Two-phase commit	Yes	Yes	Yes
Single-phase commit	Yes	No	Yes
Connect/disconnect MQI calls	Tolerated	Yes	Yes
API-crossing exit	Yes	No	No

Note: Two-phase commit is supported in the Batch/TSO environment using RRS.

z/OS *Debugging facilities, syncpoint support, and recovery support*

Use this information to learn about program debugging facilities, syncpoint support, and recovery support.

Program debugging facilities

IBM MQ for z/OS provides a trace facility that you can use to debug your programs in all environments.

Additionally, in the CICS environment you can use:

- The CICS Execution Diagnostic Facility (CEDF)

- The CICS Trace Control Transaction (CETR)
- The IBM MQ for z/OS API-crossing exit

On the z/OS platform, you can use any available interactive debugging tool that is supported by the programming language that you are using.

Syncpoint support

Synchronizing the start and end of units of work is necessary in a transaction processing environment so that transaction processing can be used safely.

This is fully supported by IBM MQ for z/OS in the CICS and IMS environments. Full support means cooperation between resource managers so that units of work can be committed or backed out in unison, under control of CICS or IMS. Examples of resource managers are Db2, CICS File Control, IMS, and IBM MQ for z/OS.

z/OS batch applications can use IBM MQ for z/OS calls to give a single-phase commit facility. This means that an application-defined set of queue operations can be committed, or backed out, without reference to other resource managers.

Two-phase commit is also supported in the z/OS batch environment using transaction management and recoverable resource manager services (RRS). For further information see Syncpoints in z/OS batch applications.

Recovery support

If the connection between a queue manager and a CICS or IMS system is broken during a transaction, some units of work might not be backed out successfully.

However, these units of work are resolved by the queue manager (under the control of the syncpoint manager) when its connection with the CICS or IMS system is reestablished.

z/OS *The IBM MQ for z/OS interface with the application environment*

To allow applications running in different environments to send and receive messages through a message queuing network, IBM MQ for z/OS provides an *adapter* for each of the environments it supports.

These adapters are the interface between application programs and IBM MQ for z/OS subsystems. They allow the programs to use the MQI.

z/OS *The batch adapter*

Use this information to learn about the batch adapter and the commit protocol it supports.

The *batch adapter* provides access to IBM MQ for z/OS resources for programs running in:

- Task (TCB) mode
- Problem or supervisor state
- Primary address space control mode

The programs must not be in cross-memory mode.

Connections between application programs and IBM MQ for z/OS are at the task level. The adapter provides a single connection thread from an application task control block (TCB) to IBM MQ for z/OS.

The adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ for z/OS ; it does not support multiphase-commit protocols.

z/OS *The RRS batch adapter*

Use this information to learn about the RRS batch adapter and the two RRS batch adapters provided by IBM MQ.

The transaction management and recoverable resource manager services (RRS) adapter:

- Uses z/OS RRS for commit control.
- Supports simultaneous connections to multiple IBM MQ subsystems running on a single z/OS instance from a single task.
- Provides z/OS-wide coordinated commitment control (using z/OS RRS) for recoverable resources accessed through z/OS RRS-compliant recoverable managers for:
  - Applications that connect to IBM MQ using the RRS batch adapter.
  - Db2-stored procedures executing in a Db2-stored procedures address space that is managed by a workload manager (WLM) on z/OS.
- Supports the ability to switch an IBM MQ batch thread between TCBs.

IBM MQ for z/OS provides two RRS batch adapters:

CSQBRSTB
This adapter requires you to change any MQCMIT statement to SRRCMIT and any MQBACK statement to SRRBACK in your IBM MQ application. (If you code MQCMIT or MQBACK in an application linked with CSQBRSTB, you receive MQRC_ENVIRONMENT_ERROR.)

CSQBRRSI
This adapter allows your IBM MQ application to use either MQCMIT and MQBACK or SRRCMIT and SRRBACK.

Note: CSQBRSTB and CSQBRRSI are shipped with linkage attributes AMODE(31) RMODE(ANY). If your application loads either stub below the 16 MB line, first relink the stub with RMODE(24).

Migration

You can migrate existing Batch/TSO IBM MQ applications to use RRS coordination with few or no changes.

If you link-edit your IBM MQ application with the CSQBRRSI adapter, MQCMIT and MQBACK syncpoint your unit of work across IBM MQ and all other RRS-enabled resource managers. If you link-edit your IBM MQ application with the CSQBRSTB adapter, change MQCMIT to SRRCMIT and MQBACK to SRRBACK. The latter approach is preferable; it clearly indicates that the syncpoint is not restricted to IBM MQ resources only.

z/OS *The IMS adapter*
If you are using the IMS adapter from an IBM MQ for z/OS system, ensure that IMS can obtain sufficient storage to accommodate messages up to 100 MB long.

Note to users

The *IMS adapter* provides access to IBM MQ for z/OS resources for:

- Online message processing programs (MPPs)
- Interactive fast path programs (IFPs)
- Batch message processing programs (BMPs)

To use these resources, the programs must be running in task (TCB) mode and problem state; they must not be in cross-memory mode or access-register mode.

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ. The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ for z/OS, with IMS acting as the syncpoint coordinator.

The adapter also provides a trigger monitor program that can start programs automatically when certain trigger conditions on a queue are met. For more information, see “Iniciando aplicativos IBM MQ usando acionadores” on page 877.

If you are writing batch DL/I programs, follow the guidance given in this topic for z/OS batch programs.

z/OS *Writing z/OS UNIX System Services applications*

The batch adapter supports queue manager connections from batch and TSO address spaces.

For a batch address space, the adapter supports connections from multiple TCBs within that address space as follows:

- Each TCB can connect to multiple queue managers using the MQCONN or MQCONNX call (but a TCB can only have one instance of a connection to a particular queue manager at any one time).
- Multiple TCBs can connect to the same queue manager (but the queue manager handle returned on any MQCONN or MQCONNX call is bound to the issuing TCB and cannot be used by any other TCB).

z/OS UNIX System Services supports two types of pthread_create call:

1. Heavyweight threads, run one for each TCB, that are ATTACHed and DETACHed at thread start and end by z/OS.
2. Medium-weight threads, run one for each TCB, but the TCB can be one of a pool of long-running TCBs. The application must perform all necessary application cleanup, because, if it is connected to a server, the default thread termination that might be provided by the server at task (TCB) termination, is **not** always driven.

Lightweight threads are not supported. (If an application creates permanent threads that dispatch their own work requests, the ***application*** is responsible for cleaning up any resources before starting the next work request.)

IBM MQ for z/OS supports z/OS UNIX System Services threads using the Batch Adapter as follows:

1. Heavyweight threads are fully supported as batch connections. Each thread runs in its own TCB, which is attached and detached at thread start and end. Should the thread end before issuing an MQDISC call, IBM MQ for z/OS performs its standard task cleanup, which includes committing any outstanding unit of work if the thread terminated normally, or backing it out if the thread terminated abnormally.
2. Medium-weight threads are fully supported, but if the TCB is going to be reused by another thread, the application must ensure that an MQDISC call, preceded by either MQCMIT or MQBACK, is issued before the next thread start. This implies that if the application has established a Program Interrupt Handler, and the application then abends, the Interrupt Handler must issue MQCMIT and MQDISC calls before reusing the TCB for another thread.

Note: Threading models do **not** support access to common IBM MQ resources from multiple threads.

z/OS *The API-crossing exit for z/OS*

This topic contains product-sensitive programming interface information.

An exit is a point in IBM-supplied code where you can run your own code. IBM MQ for z/OS provides an *API-crossing exit* that you can use to intercept calls to the MQI, and to monitor or modify the function of the MQI calls. This section describes how to use the API-crossing exit, and describes the sample exit program that is supplied with IBM MQ for z/OS.

This section is applicable only for users of CICS TS V3.1 and earlier. Users of CICS TS V3.2 and later should refer to the section CICS Integration with IBM MQ in the CICS product documentation.

Note

The API-crossing exit is invoked only by the CICS adapter of IBM MQ for z/OS. The exit program runs in the CICS address space.

z/OS *Writing your own exit program*

You can use the sample API-crossing exit program (CSQCAPX) that is supplied with IBM MQ for z/OS as a framework for your own program.

This is described in "The sample API-crossing exit program, CSQCAPX" on page 906.

When writing an exit program, to find the name of an MQI call issued by an application, examine the *ExitCommand* field of the MQXP structure. To find the number of parameters on the call, examine the *ExitParmCount* field. You can use the 16-byte *ExitUserArea* field to store the address of any dynamic storage that the application obtains. This field is retained across invocations of the exit and has the same lifetime as a CICS task.

If you are using CICS Transaction Server V3.2, you must write your exit program to be threadsafe and declare your exit program as threadsafe. If you are using earlier CICS releases, you are also recommended to write and declare your exit programs as threadsafe to be ready for migrating to CICS Transaction Server V3.2.

Your exit program can suppress execution of an MQI call by returning MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION in the *ExitResponse* field. To allow the call to be executed (and the exit program to be reinvoked after the call has completed), your exit program must return MQXCC_OK.

When invoked after an MQI call, an exit program can inspect and modify the completion and reason codes set by the call.

Usage notes

Here are some general points to consider when writing your exit program:

- For performance reasons, write your program in assembler-language. If you write it in any of the other languages supported by IBM MQ for z/OS, you must provide your own data definition file.
- Link-edit your program as AMODE(31) and RMODE(ANY).
- To define the exit parameter block to your program, use the assembler-language macro, CMQXPA.
- Specify CONCURRENCY(THREADSAFE) when you define your exit program and any programs that your exit program calls.
- If you are using the CICS Transaction Server for z/OS storage protection feature, your program must run in CICS execution key. That is, you must specify EXECKEY(CICS) when defining both your exit program and any programs to which it passes control. For information about CICS exit programs and the CICS storage protection facility, see the *CICS Customization Guide*.
- Your program can use all the APIs (for example, IMS, Db2, and CICS) that a CICS task-related user exit program can use. It can also use any of the MQI calls except MQCONN, MQCONNX, and MQDISC. However, any MQI calls within the exit program do not invoke the exit program a second time.
- Your program can issue EXEC CICS SYNCPOINT or EXEC CICS SYNCPOINT ROLLBACK commands. However, these commands commit or roll back ***all*** the updates done by the task up to the point that the exit was used, and so their use is not recommended.
- Your program must end by issuing an EXEC CICS RETURN command. It must not transfer control with an XCTL command.
- Exits are written as extensions to the IBM MQ for z/OS code. Ensure that your exit does not disrupt any IBM MQ for z/OS programs or transactions that use the MQI. These are typically indicated with a prefix of CSQ or CK.
- If CSQCAPX is defined to CICS, the CICS system attempts to load the exit program when CICS connects to IBM MQ for z/OS. If this attempt is successful, message CSQC301I is sent to the CKQC panel or to the system console. If the load is unsuccessful (for example, if the load module does not exist in any of the libraries in the DFHRPL concatenation), message CSQC315 is sent to the CKQC panel or to the system console.
- Because the parameters in the communication area are addresses, the exit program must be defined as local to the CICS system (that is, not as a remote program).

z/OS *The sample API-crossing exit program, CSQCAPX*

The sample exit program is supplied as an assembler-language program. The source file (CSQCAPX) is supplied in the library **thlqual**.SCSQASMS (where **thlqual** is the high-level qualifier used by your installation). This source file includes pseudocode that describes the program logic.

The sample program contains initialization code and a layout that you can use when writing your own exit programs.

The sample shows how to:

- Set up the exit parameter block
- Address the call and exit parameter blocks
- Determine for which MQI call the exit is being invoked
- Determine whether the exit is being invoked before or after processing of the MQI call
- Put a message on a CICS temporary storage queue
- Use the macro DFHEIENT for dynamic storage acquisition to maintain reentrancy
- Use DFHEIBLK for the CICS exec interface control block
- Trap error conditions
- Return control to the caller

Design of the sample exit program

The sample exit program writes messages to a CICS temporary storage queue (CSQ1EXIT) to show the operation of the exit.

The messages show whether the exit is being invoked before or after the MQI call. If the exit is invoked after the call, the message contains the completion code and reason code returned by the call. The sample uses named constants from the CMQXPA macro to check on the type of entry (that is, before or after the call).

The sample does not perform any monitoring function, but simply places time-stamped messages into a CICS queue indicating the type of call it is processing. This provides an indication of the performance of the MQI, as well as the correct functioning of the exit program.

Note: The sample exit program issues six EXEC CICS calls for each MQI call that is made while the program is running. If you use this exit program, IBM MQ for z/OS performance is degraded.

z/OS *Preparing and using the API-crossing exit*

The sample exit is supplied in source form only.

To use the sample exit, or an exit program that you have written, create a load library, as you would for any other CICS program, as described in "Building CICS applications in z/OS" on page 1039.

- For CICS Transaction Server for z/OS and CICS for MVS™/ESA, when you update the CICS system definition (CSD) data set, the definitions you need are in the member **thlqual**.SCSQPROC(CSQ4B100).

  Note: The definitions use a suffix of MQ. If this suffix is already used in your enterprise, this must be changed before the assembly stage.

If you use the default CICS program definitions supplied, the exit program CSQCAPX is installed in a ***disabled*** state. This is because using the exit program can produce a significant reduction in performance.

To activate the API-crossing exit temporarily:

1. Issue the command **CEMT S PROGRAM(CSQCAPX) ENABLED** from the CICS master terminal.
2. Run the CKQC transaction, and use option 3 in the Connection pull-down to alter the status of the API-crossing exit to **Enabled**.

If you want to run IBM MQ for z/OS with the API-crossing exit permanently enabled, with CICS Transaction Server for z/OS and CICS for MVS/ESA, do one of the following:

- Alter the CSQCAPX definition in member CSQ4B100, changing STATUS(DISABLED) to STATUS(ENABLED). You can update the CICS CSD definition using the CICS-supplied batch program DFHCSDUP.

- Alter the CSQCAPX definition in the CSQCAT1 group by changing the status from DISABLED to ENABLED.

In both cases, you must reinstall the group. You can do this by cold-starting your CICS system or by using the CICS CEDA transaction to reinstall the group while CICS is running.

Note: Using CEDA might cause an error if any of the entries in the group are currently in use.

End of product-sensitive programming interface information.

z/OS *Application programming with shared queues*

This topic provides information on some of the factors that you need to take into account when designing new applications to use shared queues, and when migrating existing applications to the shared-queue environment.

z/OS *Serializing your applications*

Certain types of applications might have to ensure that messages are retrieved from a queue in exactly the same order as they arrived on the queue.

For example, if IBM MQ is being used to shadow database updates on to a remote system, a message describing the update to a record must be processed after a message describing the insert of that record. In a local queuing environment, this is often achieved by the application that is getting the messages opening the queue with the MQOO_INPUT_EXCLUSIVE option, thus preventing any other getting application from processing the queue at the same time.

IBM MQ allows applications to open shared queues exclusively in the same way. However, if the application is working from a partition of a queue (for example, all database updates are on the same queue, but those for table A have a correlation identifier of A, and those for table B a correlation identifier of B), and applications want to get messages for table A updates and table B updates concurrently, the simple mechanism of opening the queue exclusively is not possible.

If this type of application is to take advantage of the high availability of shared queues, you might decide that another instance of the application that accesses the same shared queues, running on a secondary queue manager, should take over if the primary getting application or queue manager fails.

If the primary queue manager fails, two things happen:

- Shared queue peer recovery ensures that any incomplete updates from the primary application are completed or backed out.
- The secondary application takes over processing the queue.

The secondary application might start before all the incomplete units of work have been dealt with, which could lead to the secondary application retrieving the messages out of sequence. To solve this type of problem, the application can choose to be a *serialized application*.

A serialized application uses the `MQCONNX` call to connect to the queue manager, specifying a connection tag when it connects that is unique to that application. Any units of work performed by the application are marked with the connection tag. IBM MQ ensures that units of work within the queue sharing group with the same connection tag are serialized (according to the serialization options on the `MQCONNX` call).

This means that, if the primary application uses the `MQCONNX` call with a connection tag of `Database shadow retriever`, and the secondary takeover application attempts to use the `MQCONNX` call with an identical connection tag, the secondary application cannot connect to the second IBM MQ until any outstanding primary units of work have been completed, in this case by peer recovery.

Consider using the serialized application technique for applications that depend on the exact sequence of messages on a queue. In particular:

- Applications that must not restart after an application or queue manager failure until all commit and backout operations for the previous execution of the application are complete.

  In this case, the serialized application technique is only applicable if the application works in syncpoint.
- Applications that must not start while another instance of the same application is already running.

In this case, the serialized application technique is only required if the application cannot open the queue for exclusive input.

Note: IBM MQ only guarantees to preserve the sequence of messages when certain criteria are met. These are described in the description of MQGET.

z/OS *Applications that are not suitable for use with shared queues*

Some features of IBM MQ are not supported when you are using shared queues, so applications that use these features are not suitable for the shared queue environment.

Consider the following points when designing your shared-queue applications:

- Queue indexing is limited for shared queues. If you want to use the message identifier or correlation identifier to select the message that you want to get from the queue, the queue should be indexed with the correct value. If you are selecting messages by message identifier alone, the queue needs an index type of MQIT_MSG_ID (although you can also use MQIT_NONE). If you are selecting messages by correlation identifier alone, the queue must have an index type of MQIT_CORREL_ID.
- You cannot use temporary dynamic queues as shared queues. However, you can use permanent dynamic queues. The models for shared dynamic queues have a DEFTYPE of SHAREDYN (shared dynamic) although they are created and destroyed in the same way as PERMDYN (permanent dynamic) queues.

z/OS *Deciding whether to share non-application queues*

Use this information when considering sharing non-application queues.

There are queues other than application queues that you might want to consider sharing:

Initiation queues

If you define a shared initiation queue, you do not need to have a trigger monitor running on every queue manager in the queue sharing group, as long as there is at least one trigger monitor running. (You can also use a shared initiation queue even if there is a trigger monitor running on each queue manager in the queue sharing group.)

If you have a shared application queue and use the trigger type of EVERY (or a trigger type of FIRST with a small trigger interval, which behaves like a trigger type of EVERY) your initiation queue must always be a shared queue. For more information about when to use a shared initiation queue, see Table 131 on page 910.

SYSTEM.* queues

You can define the SYSTEM.ADMIN.* queues used to hold event messages as shared queues. This can be useful to check load balancing if an exception occurs. Each event message created by IBM MQ contains a correlation identifier indicating which queue manager produced it.

You must define the SYSTEM.QSG.* queues used for shared channels and intra-group queuing as shared queues.

You can also change the definitions of the SYSTEM.DEFAULT.LOCAL.QUEUE to be shared, or define your own default shared queue definition. See Defining system objects for IBM MQ for z/OS for more information.

You cannot define any other SYSTEM.* queues as shared queues.

z/OS *Migrating your existing applications to use shared queues*

Reason codes, triggering, and the MQINQ API call can work differently in a shared queue environment.

See Migrating non-shared queues to shared queues for information on migrating your existing queues to shared queues.

When you migrate your existing applications, consider the following things, which might work in a different way in the shared queue environment:

Reason codes
When you migrate your existing applications to use shared queues, check for the new reason codes that can be issued.

Triggering
If you are using a shared application queue, triggering works on committed messages only (on a non-shared application queue, triggering works on all messages).

If you use triggering to start applications, you might want to use a shared initiation queue. Table 131 on page 910 describes what you need to consider when deciding which type of initiation queue to use.

Table 131. When to use a shared-initiation queue

	Non-shared application queue	**Shared application queue**
Non-shared initiation queue	As for previous releases.	If you use a trigger type of FIRST or DEPTH, you can use a non-shared initiation queue with a shared application queue. Extra trigger messages might be generated, but this setup is good for triggering long-running applications (like the CICS bridge) and provides high availability. For trigger type FIRST or DEPTH, a trigger message triggers an instance of the application on every queue manager that is running a trigger monitor and that does not already have the application queue open for input. One trigger message is generated for every queue manager; if there is more than one trigger monitor running against the non-shared local initiation queue, on a particular queue manager, they will compete to process the message.

Table 131. When to use a shared-initiation queue (continued)		
	Non-shared application queue	**Shared application queue**
Shared initiation queue	Do not use a shared initiation queue with a non-shared application queue.	For trigger type EVERY, when an application puts a message to a shared application queue, the putting queue manager determines which queue managers have an interest in the trigger-every event and sends a notification to one of those queue managers. On the notified queue manager, the resulting action is to generate a trigger message to the initiation queue., **Note:** If you have a shared application queue that has a trigger type of EVERY, use a shared initiation queue, or you might lose trigger messages in certain circumstances; for example, a queue manager failing. For trigger type FIRST or DEPTH, one trigger message is generated by each queue manager that has the named initiation queue open for input. **Note:** For trigger type FIRST or DEPTH, if one trigger monitor instance is busy, this leaves the potential for less busy trigger monitors to process more than one trigger message from the shared initiation queue. Hence, multiple instances of the server application may be started against a given queue manager. Note that these multiple instances are started as a result of processing multiple trigger messages. Ordinarily, for trigger type FIRST or DEPTH, if an application instance is already serving an application queue, another trigger message will not be generated by the queue manager that the application is connected to.

MQINQ

When you use the MQINQ call to display information about a shared queue, the values of the number of MQOPEN calls that have the queue open for input and output relate only to the queue manager that issued the call. No information is produced about other queue managers in the queue sharing group that have the queue open.

z/OS IMS and IMS bridge applications on IBM MQ for z/OS

This information helps you to write IMS applications using IBM MQ.

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using IBM MQ” on page 71.
- To write applications that use the IBM MQ - IMS bridge, see “Writing IMS bridge applications” on page 75.

Use the following links to find out more about IMS and IMS bridge applications on IBM MQ for z/OS:

- “Writing IMS applications using IBM MQ” on page 71
- “Writing IMS bridge applications” on page 75

Related concepts

“Visão geral da Message Queue Interface” on page 733
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” on page 747

Para usar os serviços de programação do IBM MQ, um programa deve ter uma conexão a um gerenciador de filas. Use essas informações para aprender como conectar-se e desconectar-se de um gerenciador de filas.

"Abrindo e fechando objetos" on page 754
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

"Colocando mensagens em uma fila" on page 764
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" on page 780
Use estas informações para aprender como obter mensagens de uma fila.

"Consultando e configurando atributos de objeto" on page 862
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

"Confirmando e fazendo backup de unidades de trabalho" on page 865
Essas informações descrevem como confirmar e restaurar quaisquer operações de GET e PUT que ocorreram em uma unidade de trabalho.

"Iniciando aplicativos IBM MQ usando acionadores" on page 877
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" on page 896
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"Using and writing applications on IBM MQ for z/OS" on page 901
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

z/OS *Writing IMS applications using IBM MQ*

There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

Use the following links to find out more about writing IMS applications on IBM MQ for z/OS:

- "Syncpoints in IMS applications" on page 72
- "MQI calls in IMS applications" on page 72

Restrictions

There are restrictions on which IBM MQ API calls can used by an application using the IMS adapter.

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

Related concepts

"Writing IMS bridge applications" on page 75
This topic contains information about writing applications to use the IBM MQ - IMS bridge.

z/OS *Syncpoints in IMS applications*

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint).

To back out all changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see ROLB call in the IMS documentation.

The queue manager is a participant in a two-phase commit protocol; the IMS syncpoint manager is the coordinator.

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ

security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify IBM MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason, applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application ends normally, any open queues are closed and an implicit commit occurs. If the application ends abnormally, any open queues are closed and an implicit backout occurs.

z/OS *MQI calls in IMS applications*

Use this information to learn about the use of MQI calls on Server applications and Enquiry applications.

This section covers the use of MQI calls in the following types of IMS applications:

- “Server applications” on page 913
- “Inquiry applications” on page 915

Server applications

Here is an outline of the MQI server application model:

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

Sample program CSQ4ICB3 shows the implementation, in C/370, of a BMP using this model. The program establishes communication with IMS first, and then with IBM MQ:

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

Return
```

The IMS initialization determines whether the program has been called as a message-driven or a batch-oriented BMP and controls IBM MQ queue manager connection and queue handles accordingly:

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

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

The IBM MQ initialization connects to the queue manager and opens the queues. In a message-driven BMP this is called after each IMS syncpoint is taken; in a batch-oriented BMP, this is called only during program startup:

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

Return to calling function
```

The implementation of the server model in an MPP is influenced by the fact that the MPP processes a single unit of work per invocation. This is because, when a syncpoint (GU) is taken, the connection and queue handles are closed and the next IMS message is delivered. This limitation can be partially overcome by one of the following:

- **Processing many messages within a single unit-of-work**

  This involves:

  – Reading a message
  – Processing the required updates

– Putting the reply

in a loop until all messages have been processed or until a set maximum number of messages has been processed, at which time a syncpoint is taken.

Only certain types of application (for example, a simple database update or inquiry) can be approached in this way. Although the MQI reply messages can be put with the authority of the originator of the MQI message being handled, the security implications of any IMS resource updates need to be addressed carefully.

- **Processing one message per invocation of the MPP and ensuring multiple scheduling of the MPP to process all available messages.**

  Use the IBM MQ IMS trigger monitor program (CSQQTRMN) to schedule the MPP transaction when there are messages on the IBM MQ queue and no applications serving it.

  If trigger monitor starts the MPP, the queue manager name and queue name are passed to the program, as shown in the following COBOL code extract:

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

The server model, which is expected to be a long running task, is better supported in a batch processing region, although the BMP cannot be triggered using CSQQTRMN.

Inquiry applications

A typical IBM MQ application initiating an inquiry or update works as follows:

- Gather data from the user
- Put one or more IBM MQ messages
- Get the reply messages (you might have to wait for them)
- Provide a response to the user

Because messages put on to IBM MQ queues do not become available to other IBM MQ applications until they are committed, they must either be put out of syncpoint, or the IMS application must be split into two transactions.

If the inquiry involves putting a single message, you can use the *no syncpoint* option; however, if the inquiry is more complex, or resource updates are involved, you might get consistency problems if failure occurs and you do not use syncpointing.

To overcome this, you can split IMS MPP transactions using MQI calls using a program-to-program message switch; see *IMS Intersystem Communication (ISC)* for information about this. This allows an inquiry program to be implemented in an MPP:

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

z/OS *Writing IMS bridge applications*

This topic contains information about writing applications to use the IBM MQ - IMS bridge.

For information about the IBM MQ - IMS bridge, see The IMS bridge.

Use the following links to find out more about writing IMS bridge applications on IBM MQ for z/OS:

- “How the IMS bridge deals with messages” on page 76
- “Writing IMS transaction programs through IBM MQ” on page 923

Related concepts

“Writing IMS applications using IBM MQ” on page 71
There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

z/OS *How the IMS bridge deals with messages*

When you use the IBM MQ - IMS bridge to send messages to an IMS application, you need to construct your messages in a special format.

You must also put your messages on IBM MQ queues that have been defined with a storage class that specifies the XCF group and member name of the target IMS system. These are known as MQ-IMS bridge queues, or simply **bridge** queues.

The IBM MQ-IMS bridge requires exclusive input access (MQOO_INPUT_EXCLUSIVE) to the bridge queue if it is defined with QSGDISP(QMGR), or if it is defined with QSGDISP(SHARED) together with the NOSHARE option.

A user does not need to sign on to IMS before sending messages to an IMS application. The user ID in the *UserIdentifier* field of the MQMD structure is used for security checking. The level of checking is determined when IBM MQ connects to IMS, and is described in Application access control for the IMS bridge. This enables a pseudo signon to be implemented.

The IBM MQ - IMS bridge accepts the following types of message:

- Messages containing IMS transaction data and an MQIIH structure (described in MQIIH):

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

Note:

1. The square brackets, [ ], represent optional multi-segments.
2. Set the *Format* field of the MQMD structure to MQFMT_IMS to use the MQIIH structure.

- Messages containing IMS transaction data but no MQIIH structure:

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

IBM MQ validates the message data to ensure that the sum of the LL bytes plus the length of the MQIIH (if it is present) is equal to the message length.

When the IBM MQ - IMS bridge gets messages from the bridge queues, it processes them as follows:

- If the message contains an MQIIH structure, the bridge verifies the MQIIH (see MQIIH), builds the OTMA headers, and sends the message to IMS. The transaction code is specified in the input message. If this is an LTERM, IMS replies with a DFS1288E message. If the transaction code represents a command, IMS executes the command; otherwise the message is queued in IMS for the transaction.
- If the message contains IMS transaction data, but no MQIIH structure, the IMS bridge makes the following assumptions:
  - The transaction code is in bytes 5 through 12 of the user data
  - The transaction is in nonconversational mode
  - The transaction is in commit mode 0 (commit-then-send)
  - The *Format* in the MQMD is used as the *MFSMapName* (on input)
  - The security mode is MQISS_CHECK

  The reply message is also built without an MQIIH structure, taking the *Format* for the MQMD from the *MFSMapName* of the IMS output.

The IBM MQ - IMS bridge uses one or two Tpipes for each IBM MQ queue:

- A synchronized Tpipe is used for all messages using Commit mode 0 (COMMIT_THEN_SEND) (these show with SYN in the status field of the IMS /DIS TMEMBER client TPIPE xxxx command)
- A non-synchronized Tpipe is used for all messages using Commit mode 1 (SEND_THEN_COMMIT)

The Tpipes are created by IBM MQ when they are first used. A non-synchronized Tpipe exists until IMS is restarted. Synchronized Tpipes exist until IMS is cold started. You cannot delete these Tpipes yourself.

See the following topics for more information about how the IBM MQ - IMS bridge deals with messages:

- “Mapping IBM MQ messages to IMS transaction types” on page 77
- “If the message cannot be put to the IMS queue” on page 77
- “IMS bridge feedback codes” on page 78
- “The MQMD fields in messages from the IMS bridge” on page 78
- “The MQIIH fields in messages from the IMS bridge” on page 79
- “Reply messages from IMS” on page 80
- “Using alternate response PCBs in IMS transactions” on page 81
- “Sending unsolicited messages from IMS” on page 81
- “Message segmentation” on page 81
- “Data conversion for messages to and from the IMS bridge” on page 81

Related concepts

“Writing IMS transaction programs through IBM MQ” on page 923
The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

z/OS *Mapping IBM MQ messages to IMS transaction types*

A table describing the mapping of IBM MQ messages to IMS transaction types.

Table 132. How IBM MQ messages map to IMS transaction types

IBM MQ message type	**Commit-then-send (mode 0) - uses synchronized IMS Tpipes**	**Send-then-commit (mode 1) - uses non-synchronized IMS Tpipes**
Persistent IBM MQ messages	• Recoverable full function transactions • Unrecoverable transactions are rejected by IMS	• Fastpath transactions • Conversational transactions • Full function transactions
Nonpersistent IBM MQ messages	• Unrecoverable full function transactions • Recoverable transactions are permitted with IMS V8 and APAR PQ61404 and all later versions of IMS	• Fastpath transactions • Conversational transactions • Full function transactions

Note: IMS commands cannot use persistent IBM MQ messages with commit mode 0. See Commit mode (commitMode) for more information.

z/OS *If the message cannot be put to the IMS queue*

Learn about actions to take if the message cannot be put to the IMS queue.

If the message cannot be put to the IMS queue, the following action is taken by IBM MQ:

- If a message cannot be put to IMS because the message is invalid, the message is put to the dead-letter queue, and a message is sent to the system console.
- If the message is valid, but is rejected by IMS, IBM MQ sends an error message to the system console, the message includes the IMS sense code, and the IBM MQ message is put to the dead-letter queue. If the IMS sense code is 001A, IMS sends an IBM MQ message containing the reason for the failure to the reply-to queue.

  Note: In the circumstances listed previously, if IBM MQ cannot put the message to the dead-letter queue for any reason, the message is returned to the originating IBM MQ queue. An error message is sent to the system console, and no further messages are sent from that queue.

  To resend the messages, do ***one*** of the following:

  – Stop and restart the Tpipes in IMS corresponding to the queue
  – Alter the queue to GET(DISABLED), and again to GET(ENABLED)
  – Stop and restart IMS or the OTMA
  – Stop and restart your IBM MQ subsystem
- If the message is rejected by IMS for anything other than a message error, the IBM MQ message is returned to the originating queue, IBM MQ stops processing the queue, and an error message is sent to the system console.

  If an exception report message is required, the bridge puts it to the reply-to queue with the authority of the originator. If the message cannot be put to the queue, the report message is put to the dead-letter queue with the authority of the bridge. If it cannot be put to the DLQ, it is discarded.

z/OS *IMS bridge feedback codes*

IMS sense codes are typically output in hexadecimal format in IBM MQ console messages such as CSQ2001I (for example, sense code 0x001F). IBM MQ feedback codes as seen in the dead-letter header of messages put to the dead-letter queue are decimal numbers.

The IMS bridge feedback codes are in the range 301 through 399, or 600 through 855 for NACK sense code 0x001A. They are mapped from the IMS-OTMA sense codes as follows:

1. The IMS-OTMA sense code is converted from a hexadecimal number to a decimal number.
2. 300 is added to the number resulting from the calculation in 1, giving the IBM MQ *Feedback* code.
3. The IMS-OTMA sense code 0x001A, decimal 26 is a special case. A *Feedback* code in the range 600-855 is generated.
   a. The IMS-OTMA reason code is converted from a hexadecimal number to a decimal number.
   b. 600 is added to the number resulting from the calculation in a, giving the IBM MQ *Feedback* code.

For information about IMS-OTMA sense codes, see OTMA sense codes for NAK messages.

The MQMD fields in messages from the IMS bridge

Learn about the MQMD fields in messages from the IMS bridge.

The MQMD of the originating message is carried by IMS in the User Data section of the OTMA headers. If the message originates in IMS, this is built by the IMS Destination Resolution Exit. The MQMD of a message received from IMS is built as follows:

StrucID
"MD "

Version
MQMD_VERSION_1

Report
MQRO_NONE

MsgType
MQMT_REPLY

Expiry
If MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH, this field contains the remaining expiry time, else it is set to MQEI_UNLIMITED

Feedback
MQFB_NONE

Encoding
MQENC.Native (the encoding of the z/OS system)

CodedCharSetId
MQCCSI_Q_MGR (the CodedCharSetID of the z/OS system)

Format
MQFMT_IMS if the MQMD.Format of the input message is MQFMT_IMS, otherwise IOPCB.MODNAME

Priority
MQMD.Priority of the input message

Persistence
Depends on commit mode: MQMD.Persistence of the input message if CM-1; persistence matches recoverability of the IMS message if CM-0

MsgId
MQMD.MsgId if MQRO_PASS_MSG_ID, otherwise New MsgId (the default)

CorrelId
MQMD.CorrelId from the input message if MQRO_PASS_CORREL_ID, otherwise MQMD.MsgId from the input message (the default)

BackoutCount
0

ReplyToQ
Blanks

ReplyToQMgr
Blanks (set to local qmgr name by the queue manager during the MQPUT)

UserIdentifier
MQMD.UserIdentifier of the input message

AccountingToken
MQMD.AccountingToken of the input message

ApplIdentityData
MQMD.ApplIdentityData of the input message

PutApplType
MQAT_XCF if no error, otherwise MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
Date when message was put

PutTime
Time when message was put

ApplOriginData
Blanks

z/OS *The MQIIH fields in messages from the IMS bridge*

Learn about the MQIIH fields in messages from the IMS bridge.

The MQIIH of a message received from IMS is built as follows:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat of the input message if MQIIH.ReplyToFormat is not blank, otherwise IOPCB.MODNAME

Flags
0

LTermOverride
LTERM name (Tpipe) from OTMA header

MFSMapName
Map name from OTMA header

ReplyToFormat
Blanks

Authenticator
MQIIH.Authenticator of the input message if the reply message is being put to an MQ-IMS bridge queue, otherwise blanks.

TranInstanceId
Conversation ID / Server Token from OTMA header if in conversation. In versions of IMS prior to V14, this field is always nulls if not in conversation. From IMS V14 onwards, this field may be set by IMS even if not in conversation.

TranState
"C" if in conversation, otherwise blank

CommitMode
Commit mode from OTMA header ("0" or "1")

SecurityScope
Blank

Reserved
Blank

z/OS *Reply messages from IMS*

When an IMS transaction ISRTs to its IOPCB, the message is routed back to the originating LTERM or TPIPE.

These are seen in IBM MQ as reply messages. Reply messages from IMS are put onto the reply-to queue specified in the original message. If the message cannot be put onto the reply-to queue, it is put onto the dead-letter queue using the authority of the bridge. If the message cannot be put onto the dead-letter queue, a negative acknowledgment is sent to IMS to say that the message cannot be received. Responsibility for the message is then returned to IMS. If you are using commit mode 0, messages from that Tpipe are not sent to the bridge, and remain on the IMS queue; that is, no further messages are sent until restart. If you are using commit mode 1, other work can continue.

If the reply has an MQIIH structure, its format type is MQFMT_IMS; if not, its format type is specified by the IMS MOD name used when inserting the message.

z/OS *Using alternate response PCBs in IMS transactions*

When an IMS transaction uses alternate response PCBs (ISRTs to the ALTPCB, or issues a CHNG call to a modifiable PCB), the pre-routing exit (DFSYPRX0) is invoked to determine if the message should be rerouted.

If the message is to be rerouted, the destination resolution exit (DFSYDRU0) is invoked to confirm the destination and prepare the header information See Using OTMA exits in IMS and The pre-routing exit DFSYPRX0 for information about these exit programs.

Unless action is taken in the exits, all output from IMS transactions initiated from an IBM MQ queue manager, whether to the IOPCB or to an ALTPCB, will be returned to the same queue manager.

z/OS *Sending unsolicited messages from IMS*

To send messages from IMS to an IBM MQ queue, you need to invoke an IMS transaction that ISRTs to an ALTPCB.

You need to write pre-routing and destination resolution exits to route unsolicited messages from IMS and build the OTMA user data, so that the MQMD of the message can be built correctly. See The pre-routing exit DFSYPRX0 and The destination resolution user exit for information about these exit programs.

Note: The IBM MQ - IMS bridge does not know whether a message that it receives is a reply or an unsolicited message. It handles the message the same way in each case, building the MQMD and MQIIH of the reply based on the OTMA UserData that arrived with the message

Unsolicited messages can create new Tpipes. For example, if an existing IMS transaction switched to a new LTERM (for example PRINT01), but the implementation requires that the output be delivered through OTMA, a new Tpipe (called PRINT01 in this example) is created. By default, this is a non-synchronized Tpipe. If the implementation requires the message to be recoverable, set the destination resolution exit output flag. See the *IMS Customization Guide* for more information.

z/OS *Message segmentation*

You can define IMS transactions as expecting single- or multi-segment input.

The originating IBM MQ application must construct the user input following the MQIIH structure as one or more LLZZ-data segments. All segments of an IMS message must be contained in a single IBM MQ message sent with a single MQPUT.

The maximum length of an LLZZ-data segment is defined by IMS/OTMA (32767 bytes). The total IBM MQ message length is the sum of the LL bytes, plus the length of the MQIIH structure.

All the segments of the reply are contained in a single IBM MQ message.

There is a further restriction on the 32 KB limitation on messages with format MQFMT_IMS_VAR_STRING. When the data in an ASCII-mixed CCSID message is converted to an EBCDIC-mixed CCSID message, a shift-in byte or a shift-out byte is added every time that there is a transition between SBCS and DBCS characters. The 32 KB restriction applies to the maximum size of the message. That is, because the LL field in the message cannot exceed 32 KB, the message must not exceed 32 KB including all shift-in and shift-out characters. The application building the message must allow for this.

z/OS *Data conversion for messages to and from the IMS bridge*

The data conversion is performed by either the distributed queuing facility (which may call any necessary exits) or by the intra group queuing agent (which does not support the use of exits) when it puts a message to a destination queue that has XCF information defined for its storage class. The data conversion does not occur when a message is delivered to a queue by publish/subscribe.

Any exits needed must be available to the distributed queuing facility in the data set referenced by the CSQXLIB DD statement. This means that you can send messages to an IMS application using the IBM MQ - IMS bridge from any IBM MQ platform.

If there are conversion errors, the message is put to the queue unconverted; this results eventually in it being treated as an error by the IBM MQ - IMS bridge, because the bridge cannot recognize the header format. If a conversion error occurs, an error message is sent to the z/OS console.

See “Escrevendo saídas de conversão de dados” on page 995 for detailed information about data conversion in general.

Sending messages to the IBM MQ - IMS bridge

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIIH structure, the *Format* in the MQMD must be set to the built-in format MQFMT_IMS, and the *Format* in the MQIIH must be set to the name of the format that describes your message data. If there is no MQIIH, set the *Format* in the MQMD to your format name.

If your data (other than the LLZZs) is all character data (MQCHAR), use as your format name (in the MQIIH or MQMD, as appropriate) the built-in format MQFMT_IMS_VAR_STRING. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any MQIIH at the start of the message).

If your application uses *MFSMapName*, you can use messages with the MQFMT_IMS instead, and define the map name passed to the IMS transaction in the MFSMapName field of the MQIIH.

Receiving messages from the IBM MQ - IMS bridge

If an MQIIH structure is present on the original message that you are sending to IMS, one is also present on the reply message.

To ensure that your reply is converted correctly:

- If you have an MQIIH structure on your original message, specify the format that you want for your reply message in the MQIIH *ReplytoFormat* field of the original message. This value is placed in the MQIIH *Format* field of the reply message. This is particularly useful if all your output data is of the form LLZZ<character data>.

- If you do not have an MQIIH structure on your original message, specify the format that you want for the reply message as the MFS MOD name in the IMS application's ISRT to the IOPCB.

z/OS *Writing IMS transaction programs through IBM MQ*

The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

When an IMS transaction is started from a 3270 screen, the message passes through IMS Message Format Services. This can remove all terminal dependency from the data stream seen by the transaction. When a transaction is started through OTMA, MFS is not involved. If application logic is implemented in MFS, this must be re-created in the new application.

In some IMS transactions, the end-user application can modify certain 3270 screen behavior, for example, highlighting a field that has had invalid data entered. This type of information is communicated by adding a two-byte attribute field to the IMS message for each screen field that needs to be modified by the program.

Thus, if you are coding an application to mimic a 3270, you need to take account of these fields when building or receiving messages.

You might need to code information in your program to process:

- Which key is pressed (for example, Enter and PF1)
- Where the cursor is when the message is passed to your application
- Whether the attribute fields have been set by the IMS application
  - High, normal, or zero intensity
  - Color
  - Whether IMS is expecting the field back the next time that Enter is pressed
- Whether the IMS application has used null characters (X'3F') in any fields.

If your IMS message contains only character data (apart from the LLZZ-data segment), and you are using an MQIIH structure, set the MQMD format to MQFMT_IMS and the MQIIH format to MQFMT_IMS_VAR_STRING.

If your IMS message contains only character data (apart from the LLZZ-data segment), and you are **not** using an MQIIH structure, set the MQMD format to MQFMT_IMS_VAR_STRING and ensure that your IMS application specifies MODname MQFMT_IMS_VAR_STRING when replying. If a problem occurs (for example, user not authorized to use the transaction) and IMS sends an error message, this has an MODname of the form DFSMOx, where x is a number in the range 1 through 5. This is put in the MQMD.Format.

If your IMS message contains binary, packed, or floating point data (apart from the LLZZ-data segment), code your own data-conversion routines. Refer to *IMS/ESA Application Programming: Transaction Manager* for information about IMS screen formatting.

Consider the following topics when writing code to handle IMS transactions through IBM MQ.

- “Writing IBM MQ applications to invoke IMS conversational transactions” on page 923
- “Writing programs containing IMS commands” on page 924
- “Triggering” on page 924

Writing IBM MQ applications to invoke IMS conversational transactions

Use this information as a guide for considerations when writing IBM MQ application to invoke IMS conversational transactions.

When you write an application that invokes an IMS conversation, consider the following:

- Include an MQIIH structure with your application message.

- Set the *CommitMode* in MQIIH to MQICM_SEND_THEN_COMMIT.
- To invoke a new conversation, set *TranState* in MQIIH to MQITS_NOT_IN_CONVERSATION.
- To invoke second and subsequent steps of a conversation, set *TranState* to MQITS_IN_CONVERSATION, and set *TranInstanceId* to the value of that field returned in the previous step of the conversation.
- There is no easy way in IMS to find the value of a *TranInstanceId*, should you lose the original message sent from IMS.
- The application must check the *TranState* of messages from IMS to check whether the IMS transaction has terminated the conversation.
- You can use /EXIT to end a conversation. You must also quote the *TranInstanceId*, set *TranState* to MQITS_IN_CONVERSATION, and use the IBM MQ queue on which the conversation is being carried out.
- You cannot use /HOLD or /REL to hold or release a conversation.
- Conversations invoked through the IBM MQ - IMS bridge are terminated if IMS is restarted.

Writing programs containing IMS commands

An application program can build an IBM MQ message of the form LLZZ*command*, instead of a transaction, where *command* is of the form /DIS TRAN PART or /DIS POOL ALL.

Most IMS commands can be issued in this way; see *IMS V11 Communications and Connections* for details. The command output is received in the IBM MQ reply message in the text form as would be sent to a 3270 terminal for display.

OTMA has implemented a special form of the IMS display transaction command, which returns an architected form of the output. The exact format is defined in *IMS V11 Communications and Connections*. To invoke this form from an IBM MQ message, build the message data as before, for example /DIS TRAN PART, and set the TranState field in the MQIIH to MQITS_ARCHITECTED. IMS processes the command, and returns the reply in the architected form. An architected response contains all the information that could be found in the text form of the output, and one additional piece of information: whether the transaction is defined as recoverable or non-recoverable.

Triggering

The IBM MQ - IMS bridge does not support trigger messages.

If you define an initiation queue that uses a storage class with XCF parameters, messages put to that queue are rejected when they get to the bridge.

Escrevendo aplicativos clientes processuais

O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

Os aplicativos podem ser construídos e executados no ambiente do cliente IBM MQ. O aplicativo deve ser construído e vinculado ao IBM MQ MQI client usado. A maneira como os aplicativos são construídos e vinculados varia de acordo com a plataforma e a linguagem de programação usadas. Para obter informações sobre como construir aplicativos clientes, consulte “Construindo aplicativos para IBM MQ MQI clients” na página 930.

É possível executar um aplicativo IBM MQ em um ambiente integral do IBM MQ e em um ambiente do IBM MQ MQI client sem mudar seu código, desde que determinadas condições sejam atendidas. Para obter mais informações sobre como executar seus aplicativos no ambiente do cliente IBM MQ, consulte “Executando aplicativos no ambiente do IBM MQ MQI client” na página 932.

Se você usar a interface de fila de mensagens (MQI) para escrever aplicativos para execução em um ambiente do IBM MQ MQI client, há alguns controles adicionais a impor durante uma chamada MQI

para assegurar que o processamento do aplicativo IBM MQ não seja interrompido. Para obter mais informações sobre esses controles, consulte “Usando o MQI em um aplicativo cliente” na página 925.

Consulte os tópicos a seguir para obter informações sobre como preparar e executar outros tipos de aplicativos, como aplicativos clientes:

- “Preparando e executando aplicativos CICS e Tuxedo” na página 945
- “Preparando e executando aplicativos Microsoft Transaction Server” na página 50
- “Preparando e executando aplicativos IBM MQ JMS” na página 948

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

“Desenvolvendo aplicativos para o IBM MQ” na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

“Considerações de design para aplicativos IBM MQ” na página 50
Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

“Escrevendo um aplicativo processual para enfileiramento” na página 732
Use essas informações para aprender como escrever aplicativos de enfileiramento, conectar e desconectar de um gerenciador de filas, publicar/assinar e abrir e fechar objetos.

“Escrevendo aplicativos de publicar/assinar” na página 821
Inicie a escrever aplicativos de publicar/assinar IBM MQ.

“Construindo um aplicativo processual” na página 1012
É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

“Manipulando erros de programa processual” na página 1050
Estas informações explicarão os erros associados às chamadas MQI de aplicativos, ao realizar uma chamada ou quando sua mensagem for entregue para seu destino final.

Tarefas relacionadas

“Usando os programas processuais de amostra do IBM MQ” na página 1070
Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

Usando o MQI em um aplicativo cliente

Esta coleção de tópicos considera as diferenças entre a gravação de seu aplicativo IBM MQ para execução no ambiente do cliente de uma interface de fila de mensagens (MQI) e para execução no ambiente do gerenciador de filas completo do IBM MQ.

Ao projetar um aplicativo, considere quais controles você precisa impor durante uma chamada MQI para assegurar que o processamento do aplicativo IBM MQ não seja interrompido.

Antes de poder executar aplicativos que usam o MQI, deve-se criar certos objetos do IBM MQ. Para obter mais informações, consulte Programas de aplicativo usando o MQI.

Limitando o tamanho de uma mensagem em um aplicativo cliente

Um gerenciador de filas tem um comprimento máximo de mensagem, mas o tamanho máximo de mensagem que é possível transmitir de um aplicativo cliente é limitado pela definição de canal.

O atributo de comprimento máximo da mensagem (MaxMsgLength) de um gerenciador de filas é o comprimento máximo de uma mensagem que pode ser manipulada por esse gerenciador de filas.

Multi Em Multiplataformas, é possível aumentar o atributo de comprimento máximo da mensagem de um gerenciador de filas. Para obter informações adicionais, consulte ALTER QMGR.

É possível descobrir o valor de MaxMsgLength para um gerenciador de filas usando a chamada MQINQ.

Se o atributo MaxMsgLength for mudado, não será feita nenhuma verificação para ver se já existem filas, e até mesmo mensagens, com um comprimento maior que o novo valor. Após mudar esse atributo, reinicie aplicativos e canais para assegurar que a mudança entrou em vigor. Então, não será possível que novas mensagens sejam geradas excedendo o MaxMsgLength do gerenciador de filas ou da fila (a menos que a segmentação do gerenciador de filas seja permitida).

O comprimento máximo da mensagem em uma definição de canal limita o tamanho de uma mensagem que é possível transmitir ao longo de uma conexão do cliente. Se um aplicativo IBM MQ tentar usar a chamada MQPUT ou a chamada MQGET com uma mensagem maior que isso, um código de erro será retornado ao aplicativo. O parâmetro de tamanho máximo da mensagem da definição de canal não afeta o tamanho máximo da mensagem que pode ser consumido usando MQCB por meio de uma conexão do cliente.

Conceitos relacionados

"Usando MQCONNX" na página 930
É possível usar a chamada MQCONNX para especificar uma estrutura de definição de canal (MQCD) na estrutura MQCNO.

Referências relacionadas

Comprimento máximo da mensagem (MAXMSGL)

ALTER CHANNEL

2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

Escolhendo o CCSID do cliente ou servidor

Use o identificador de conjunto de caracteres codificados (CCSID) local para o cliente. O gerenciador de filas executa a conversão necessária. É possível usar a variável de ambiente **MQCCSID** para substituir o CCSID Se seu aplicativo desempenhar múltiplas operações PUT, o CCSID e campos de codificação do MQMD poderão ser sobrescritos após a conclusão da primeira PUT.

Os dados passados através da interface de fila de mensagens (MQI) do aplicativo para o stub do cliente devem estar no CCSID local, codificados para o IBM MQ MQI client. Se o gerenciador de filas conectado requerer que os dados sejam convertidos, então a conversão será feita pelo código de suporte a clientes no gerenciador de filas.

No IBM WebSphere MQ 7.0 e versões mais recentes, o cliente Java poderá fazer a conversão se o gerenciador de filas não puder fazer. Consulte "Conexões do cliente do IBM MQ classes for Java" na página 379.

O código do cliente assume que os dados de caractere que cruzam o MQI no cliente estão no CCSID configurados para aquela estação de trabalho. Se esse CCSID for um CCSID não suportado ou não for o CCSID necessário, ele poderá ser substituído pela variável de ambiente **MQCCSID** usando um destes comandos:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Se este parâmetro estiver configurado no perfil, supõe-se que todos os dados de MQI estarão na página de códigos 850.

Nota: A suposição sobre a página de códigos 850 não se aplica aos dados do aplicativo na mensagem.

Se seu aplicativo estiver executando múltiplas operações PUT que incluem cabeçalhos do IBM MQ após o descritor de mensagens (MQMD), esteja ciente de que o CCSID e campos de codificação do MQMD serão sobrescritos após a conclusão da primeira PUT.

Após a primeira operação PUT, estes campos conterão o valor usado pelo gerenciador de filas conectado para converter os cabeçalhos do IBM MQ. Assegure que seu aplicativo reconfigure os valores para os valores que ele requer.

Usando MQINQ em um aplicativo cliente

Alguns valores consultados usando MQINQ são modificados pelo código do cliente.

CCSID
é configurado para o CCSID do cliente, não para aquele do gerenciador de filas.

MaxMsgLength
será reduzido se for restringido pela definição de canal. Isto será menor que:

- O valor definido na definição de fila ou
- O valor definido na definição de canal

Para obter informações adicionais, consulte o MQINQ.

Usando a coordenação de ponto de sincronização em um aplicativo cliente

Um aplicativo em execução no cliente base poderá emitir MQCMIT e MQBACK, mas o escopo do controle do ponto de sincronização é limitado aos recursos do MQI. É possível usar um gerenciador de transações externo com um cliente transacional estendido.

No IBM MQ, uma das funções do gerenciador de filas é o controle do ponto de sincronização em um aplicativo. Se um aplicativo for executado em um cliente base do IBM MQ, ele poderá emitir MQCMIT e MQBACK, mas o escopo do controle do ponto de sincronização é limitado aos recursos do MQI. O verbo do IBM MQ MQBEGIN não é válido em um ambiente do cliente base.

Aplicativos em execução no ambiente do gerenciador de filas completo no servidor podem coordenar múltiplos recursos (por exemplo, bancos de dados) por meio de um monitor de transação. No servidor, é possível usar o Monitor de Transação fornecido com os produtos IBM MQ ou outro monitor de transação, como CICS. Não é possível usar um monitor de transação com um aplicativo cliente base.

É possível usar um gerenciador de transações externo com um cliente transacional estendido do IBM MQ. Consulte O que é um cliente transacional estendido? para obter os detalhes.

Usando leia mais adiante em um aplicativo cliente

É possível usar leia mais adiante em um cliente para permitir que mensagens não persistentes sejam enviadas a um cliente sem que o aplicativo cliente precise solicitar as mensagens.

Quando um cliente requer uma mensagem de um servidor, ele envia uma solicitação ao servidor. Ele envia uma solicitação separada para cada uma das mensagens que consome. Para melhorar o desempenho de um cliente que consome mensagens não-persistentes evitando a necessidade de enviar estas mensagens de pedido, um cliente pode ser configurado para usar leitura antecipada. Leia mais adiante permite que mensagens sejam enviadas a um cliente sem que um aplicativo precise solicitá-las.

Usar leia mais adiante pode melhorar o desempenho ao consumir mensagens não persistentes a partir de um aplicativo cliente. Essa melhoria de desempenho está disponível para MQI e aplicativos JMS. Aplicativos clientes que usam MQGET ou consumo assíncrono se beneficiam das melhorias de desempenho ao consumir mensagens não persistentes.

Ao chamar MQOPEN com MQOO_READ_AHEAD, o cliente IBM MQ somente ativará o modo leia mais adiante se determinadas condições forem atendidas. Essas condições incluem:

- O aplicativo cliente deve ser compilado e vinculado em relação as bibliotecas encadeadas do cliente IBM MQ MQI.
- O canal do cliente deve estar utilizando o protocolo TCP/IP
- O canal deve ter a configuração de SharingConversations diferente de zero (SHARECNV) em ambas as definições de canal, do cliente e do servidor.

Quando a opção leia mais adiante estiver ativada, as mensagens são enviadas para um buffer de memória no cliente chamado de buffer de leia mais adiante. O cliente tem um buffer de leia mais adiante para cada fila que tiver aberta com a opção leia mais adiante ativada. As mensagens no buffer de leia mais adiante não são persistidas. O cliente atualiza periodicamente o servidor com informações sobre a quantia de dados que consumiu.

Nem todos os designs de aplicativo cliente são adequados para usar leia mais adiante, pois nem todas as opções são suportadas para uso. Algumas opções precisam ser consistentes entre chamadas MQGET quando leia mais adiante está ativada. Se um cliente alterar seus critérios de seleção entre chamadas MQGET, as mensagens que estão sendo armazenadas no buffer de leia mais adiante permanecerão conectadas ao buffer de leia mais adiante do cliente. Para obter mais informações, consulte “Melhorando o desempenho de mensagens não persistentes” na página 799

A configuração de leia mais adiante é controlada por três atributos, MaximumSize, PurgeTime e UpdatePercentage, que são especificados na sub-rotina MessageBuffer do arquivo de configuração do cliente IBM MQ.

Usando put assíncrono em um aplicativo cliente

Usando put assíncrono, um aplicativo pode colocar uma mensagem em uma fila sem esperar uma resposta do gerenciador de filas. É possível usar isso para melhorar o desempenho do sistema de mensagens em algumas situações.

Normalmente, quando um aplicativo coloca uma mensagem ou mensagens em uma fila, usando MQPUT ou MQPUT1, o aplicativo precisa esperar o gerenciador de filas confirmar se ele processou a solicitação MQI. É possível melhorar o desempenho do sistema de mensagens, principalmente para aplicativos que usam ligações do cliente e aplicativos que colocam um grande número de mensagens pequenas em uma fila, optando por colocar as mensagens de forma assíncrona. Quando um aplicativo efetuar put de uma mensagem de forma assíncrona, o gerenciador de filas não retornará o sucesso ou falha de cada chamada, mas será possível verificar erros periodicamente.

Para colocar uma mensagem em uma fila de forma assíncrona, use a opção MQPMO_ASYNC_RESPONSE no campo *Options* da estrutura MQPMO.

Se uma mensagem não for elegível para put assíncrono, ela será colocada em uma fila de forma síncrona.

Ao solicitar resposta de put assíncrono para MQPUT ou MQPUT1, um CompCode e Reason de MQCC_OK e MQRC_NONE não significam necessariamente que a mensagem foi colocada com sucesso em uma fila. Embora o sucesso ou falha de cada chamada MQPUT ou MQPUT1 individual possa não ser retornado imediatamente, o primeiro erro que ocorreu sob uma chamada assíncrona pode ser determinado posteriormente por meio de uma chamada para MQSTAT.

Para obter mais detalhes sobre MQPMO_ASYNC_RESPONSE, consulte Opções de MQPMO.

O programa de amostra Asynchronous Put demonstra alguns dos recursos disponíveis. Para obter detalhes dos recursos e o design do programa, e como executá-lo, consulte “O programa de amostra Asynchronous Put” na página 1091.

Usando conversas de compartilhamento em um aplicativo cliente

Em um ambiente no qual conversas de compartilhamento são permitidas, as conversas podem compartilhar uma instância do canal MQI.

Conversas de compartilhamento são controladas por dois campos, ambos chamados SharingConversations, um dos quais faz parte da estrutura de definição de canal (MQCD) e um deles faz parte da estrutura do parâmetro de saída do canal (MQCXP). O campo SharingConversations no MQCD é um valor de número inteiro, que determina o número máximo de conversas que podem compartilhar uma

instância do canal associada ao canal. O campo SharingConversations no MQCXP é um valor booleano, que indica se a instância do canal é atualmente compartilhada.

Em um ambiente no qual conversas de compartilhamento não são permitidas, novas conexões do cliente que especificam MQCDs idênticos não compartilharão uma instância do canal.

Uma nova conexão de aplicativo cliente compartilhará a instância do canal quando as condições a seguir forem verdadeiras:

- Ambas as extremidades de conexão do cliente e de conexão do servidor da instância do canal são configuradas para conversas de compartilhamento e esses valores não são substituídos pelas saídas do canal.
- O valor de MQCD da conexão do cliente (fornecido na chamada MQCONNX do cliente ou a partir da tabela de definição de canal de cliente (CCDT)) corresponde exatamente ao valor de MQCD da conexão do cliente fornecido na chamada MQCONNX do cliente ou a partir da CCDT quando a instância do canal existente foi estabelecida pela primeira vez. Observe que o MQCD original pode ter sido alterado subsequentemente pelas saídas ou pela negociação do canal, mas que a correspondência é feita com relação ao valor que foi fornecido ao sistema do cliente antes dessas mudanças serem feitas.
- O limite de conversas de compartilhamento no lado do servidor não é excedido.

Se uma nova conexão de aplicativo cliente corresponder aos critérios para executar o compartilhamento em uma instância do canal com outras conversas, essa decisão será feita antes de qualquer saída ser chamada nessa conversa. Saídas em uma conversa desse tipo não podem alterar o fato de que ela está compartilhando a instância do canal com outras conversas. Se não houver instâncias do canal existentes correspondentes à nova definição de canal, uma nova instância do canal será conectada.

Negociação de canal ocorre somente para a primeira conversa em uma instância do canal; os valores negociados para a instância do canal são fixados nesse estágio e não podem ser alterados quando conversas subsequentes forem iniciadas. Autenticação TLS também ocorre somente para a primeira conversa.

Se o valor de MQCD de SharingConversations for alterado durante a inicialização de qualquer saída de segurança, envio ou recebimento para a primeira conversa no soquete na extremidade de conexão do cliente ou da conexão do servidor da instância do canal, o novo valor que terá após todas essas saídas serem inicializadas será usado para determinar o valor de conversas de compartilhamento para a instância do canal (o valor mais baixo terá precedência).

Se o valor negociado para conversas de compartilhamento for zero, a instância do canal nunca será compartilhada. Programas de saída adicionais que configuram esse campo para zero são executados de forma similar em sua própria instância do canal.

Se o valor negociado para conversas de compartilhamento for maior que zero, então, SharingConversations de MQCXP será configurado para TRUE para chamadas subsequentes a saídas, indicando que outros programas de saída nesta instância do canal podem ser inseridos simultaneamente a este.

Quando você escrever um programa de saída de canal, considere se ele será executado em uma instância do canal que pode envolver conversas de compartilhamento. Se a instância do canal precisar envolver conversas de compartilhamento, considere o efeito nas outras instâncias da saída do canal de campos de MQCD em mudança; todos os campos de MQCD têm valores comuns entre todas as conversas de compartilhamento. Após a instância do canal ser estabelecida, se programas de saída tentarem alterar campos de MQCD, eles poderão encontrar problemas porque outras instâncias dos programas de saída em execução na instância do canal poderão estar tentando alterar os mesmos campos ao mesmo tempo. Se essa situação puder surgir para seus programas de saída, você deverá serializar o acesso ao MQCD em seu código de saída.

Se estiver trabalhando com um canal definido para compartilhar conversas, mas não desejar que o compartilhamento ocorra em uma instância do canal específica, configure o valor de MQCD de SharingConversations para 1 ou 0 ao inicializar uma saída do canal na primeira conversa na instância do canal. Consulte SharingConversations para obter uma explicação dos valores de SharingConversations.

exemplo

Conversas de compartilhamento estão ativadas.

Você está usando uma definição de canal de conexão do cliente que especifica um programa de saída.

Na primeira vez que esse canal for iniciado, o programa de saída altera alguns dos parâmetros de MQCD quando ele é inicializado. Isso é influenciado pelo canal, portanto, a definição com a qual o canal está em execução agora é diferente daquela que foi originalmente fornecida. O parâmetro MQCXP SharingConversations é configurado para TRUE.

Na próxima vez que o aplicativo se conectar usando este canal, a conversa será executada na instância do canal que foi iniciada anteriormente, porque ela possui a mesma definição de canal original. A instância do canal à qual o aplicativo se conecta na segunda vez é a mesma instância à qual se conectou na primeira vez. Consequentemente, usa as definições que foram alteradas pelo programa de saída. Quando o programa de saída é inicializado para a segunda conversa, embora seja possível alterar campos MQCD, ele não é acionado pelo canal. Essas mesmas características se aplicam a qualquer conversa subsequente que compartilhe a instância do canal.

Usando MQCONNX

É possível usar a chamada MQCONNX para especificar uma estrutura de definição de canal (MQCD) na estrutura MQCNO.

Isto permite que o aplicativo cliente que está chamando especifique a definição do canal de conexão do cliente no tempo de execução. Para obter mais informações, consulte Criando um canal de conexão do cliente no IBM MQ MQI client usando MQCNO. Quando você usa MQCONNX, a chamada emitida no servidor depende do nível do servidor e da configuração do listener.

Quando você usa MQCONNX a partir de um cliente, as seguintes opções são ignoradas:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

A estrutura MQCD que é possível usar depende do número da versão do MQCD que você está usando. Para obter informações sobre versões do MQCD (MQCD_VERSION), consulte Versão do MQCD. É possível usar a estrutura MQCD, por exemplo, para transmitir programas de saída do canal ao servidor. Se você estiver usando o MQCD Versão 3 ou posterior, poderá usar a estrutura para transmitir uma matriz de saídas ao servidor. É possível usar esta função para executar mais de uma operação na mesma mensagem, como criptografia e compactação, incluindo uma saída para cada operação, em vez de modificar uma saída existente. Se você não especificar uma matriz na estrutura MQCD, os campos de saída únicos serão verificados. Para obter informações adicionais sobre programas de saída do canal, consulte “Programas de Saída de Canal para Canais de Mensagens” na página 973.

Manipulações de conexões compartilhadas no MQCONNX

É possível compartilhar identificadores entre diferentes encadeamentos no mesmo processo, usando manipulações de conexões compartilhadas.

Ao especificar uma manipulação de conexões compartilhada, a manipulação de conexões retornada da chamada MQCONNX poderá ser passada nas chamadas MQI subsequentes em qualquer encadeamento no processo.

Nota: É possível usar uma manipulação de conexões compartilhada em um IBM MQ MQI client para conectar-se a um gerenciador de filas do servidor que não suporte manipulações de conexões compartilhadas.

Construindo aplicativos para IBM MQ MQI clients

Os aplicativos podem ser construídos e executados no ambiente do IBM MQ MQI client. O aplicativo deve ser construído e vinculado ao IBM MQ MQI client usado. A maneira como os aplicativos são construídos e vinculados varia de acordo com a plataforma e a linguagem de programação usadas.

Se um aplicativo precisar ser executado em um ambiente do cliente, será possível gravá-lo nos idiomas mostrados na seguinte tabela:

Tabela 133. Linguagens de Programação Suportadas nos Ambientes do Cliente

Plataforma do cliente	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Sim	Sim	Sim			
IBM i	Sim		Sim		Sim	
Linux	Sim	Sim	Sim			
Windows	Sim	Sim	Sim			Sim

Multi *Vinculando aplicativos C ao código do IBM MQ MQI client*

Tendo escrito seu aplicativo IBM MQ que deseja executar no IBM MQ MQI client, deverá vinculá-lo ao código do IBM MQ MQI client.

É possível vincular seu aplicativo ao código do IBM MQ MQI client de duas maneiras:

1. Diretamente, conectando seu aplicativo a um gerenciador de filas, nesse caso, o gerenciador de filas deve estar na mesma máquina que seu aplicativo.
2. A um arquivo de biblioteca do cliente, que fornece acesso aos gerenciadores de filas na mesma ou em uma máquina diferente.

O IBM MQ fornece um arquivo de biblioteca de cliente para cada ambiente:

AIX **AIX**
A biblioteca libmqic.a para aplicativos não encadeados ou a biblioteca libmqic_r.a para aplicativos encadeados.

Linux **Linux**
A biblioteca libmqic.so para aplicativos não encadeados ou a biblioteca libmqic_r.so para aplicativos encadeados.

IBM i **IBM i**
Ligue o aplicativo cliente ao programa de serviços do cliente LIBMQIC para aplicativos não encadeados ou ao programa de serviços LIBMQIC_R para aplicativos encadeados.

Windows **Windows**
MQIC32.LIB.

ALW *Vinculando aplicativos C++ ao código do IBM MQ MQI client*

É possível escrever aplicativos para serem executados no cliente no C++. Os métodos de construção variam de acordo com o ambiente.

Para obter informações sobre como vincular seus aplicativos C++, consulte Construindo programas C++ do IBM MQ.

Para obter detalhes completos de todos os aspectos do uso de C++, consulte Usando C++

Multi *Vinculando aplicativos COBOL com o código de IBM MQ MQI client*

Tendo escrito um aplicativo COBOL que deseja executar no IBM MQ MQI client, deverá vinculá-lo a uma biblioteca apropriada.

O IBM MQ fornece um arquivo de biblioteca de cliente para cada ambiente:

AIX **AIX**

Vincule seu aplicativo COBOL não encadeado com a biblioteca libmqicb.a ou o aplicativo COBOL encadeado com libmqicb_r.a.

IBM i **IBM i**

Ligue o aplicativo cliente COBOL com o programa de serviços AMQCSTUB para aplicativos não encadeados ou o programa de serviços AMQCSTUB_R para aplicativos encadeados.

Windows **Windows**

Vincule seu código do aplicativo com a biblioteca MQICCBB para COBOL de 32 bits. O IBM MQ MQI client for Windows não suporta COBOL de 16 bits.

Windows *Vinculando aplicativos Visual Basic ao código do IBM MQ MQI client*

É possível vincular os aplicativos Microsoft Visual Basic com o código IBM MQ MQI client no Windows.

Deprecated

No IBM MQ 9.0, o suporte para o Microsoft Visual Basic 6.0 foi descontinuado. As classes do IBM MQ para .NET são a tecnologia de substituição recomendada. Para obter mais informações, consulte Desenvolvendo aplicativos .NET.

Vincule seu aplicativo Visual Basic aos arquivos include a seguir:

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
comandos PCF

CMQXB.bas
Canais

Configure `mqtype=2` para o cliente no compilador Visual Basic para assegurar a seleção automática correta do dll do cliente:

MQIC32.dll
Windows 7, Windows 8, Windows 2008 e Windows 2012

Conceitos relacionados

“Codificação no Visual Basic” na página 1065
Informações a serem consideradas ao codificar programas IBM MQ no Microsoft Visual Basic. O Visual Basic é suportado somente no Windows.

“Preparando programas Visual Basic no Windows” na página 1032
Informações a serem consideradas ao usar programas Microsoft Visual Basic no Windows.

Executando aplicativos no ambiente do IBM MQ MQI client

É possível executar um aplicativo IBM MQ em um ambiente integral do IBM MQ e em um ambiente do IBM MQ MQI client sem mudar seu código, desde que determinadas condições sejam atendidas.

Estas condições são que:

- O aplicativo não precisa se conectar a mais de um gerenciador de filas simultaneamente.
- O nome do gerenciador de filas não é prefixado com um asterisco (*) em uma chamada `MQCONN` ou `MQCONNX`.
- O aplicativo não precisa usar nenhuma das exceções listadas em Quais aplicativos são executados em um IBM MQ MQI client?

Nota: As bibliotecas usadas no momento de edição de link determinam o ambiente no qual seu aplicativo deve ser executado.

Ao trabalhar no ambiente do IBM MQ MQI client, lembre-se de que:

- Cada aplicativo em execução no ambiente do IBM MQ MQI client tem suas próprias conexões com servidores. Um aplicativo estabelece uma conexão com um servidor toda vez que emite uma chamada `MQCONN` ou `MQCONNX`.
- Um aplicativo envia e recebe mensagens de forma síncrona. Isto indica uma espera entre o momento em que a chamada é emitida no cliente e o retorno de um código de conclusão e de um código de razão pela rede.
- Toda conversão de dados é feita pelo servidor, mas consulte também MQCCSID para obter informações sobre como substituir o CCSID configurado da máquina.

Conectando aplicativos IBM MQ MQI client a gerenciadores de filas

Um aplicativo em execução em um ambiente do IBM MQ MQI client pode se conectar a um gerenciador de filas de várias maneiras. É possível usar variáveis de ambiente, a estrutura MQCNO ou uma tabela de definição do cliente.

Quando um aplicativo em execução em um ambiente do cliente IBM MQ emite uma chamada MQCONN ou MQCONNX, o cliente identifica como deve se fazer a conexão. Quando uma chamada MQCONNX é emitida por um aplicativo em um cliente IBM MQ, a biblioteca do cliente MQI procura as informações do canal do cliente na ordem a seguir:

1. Usando os conteúdos dos campos `ClientConnOffset` ou `ClientConnPtr` da estrutura MQCNO (se fornecida). Esses campos identificam a estrutura de definição de canal (MQCD) a ser usada como a definição do canal de conexão do cliente. Os detalhes da conexão podem ser substituídos usando uma saída de pré-conexão. Para obter informações adicionais, consulte “Fazendo referência a definições de conexão usando uma saída de pré-conexão de um repositório” na página 1005.
2. Se a variável de ambiente **MQSERVER** for configurada, o canal que ela define será usado
3. Se um arquivo `mqclient.ini` for definido e a sub-rotina Channels contiver um atributo **ServerConnectionParms** , o canal que ele define será usado.. Para obter mais informações, consulte IBM MQ MQI client arquivo de configuração, `mqclient.ini` e Sub-rotina de Canais do arquivo de configuração do cliente
4. Se as variáveis de ambiente **MQCHLLIB** e **MQCHLTAB** estiverem configuradas, a tabela de definição de canal do cliente para a qual elas apontam será usada.. Alternativamente, a variável de ambiente **MQCCDTURL** fornece a capacidade equivalente para configurar uma combinação das variáveis de ambiente **MQCHLLIB** e **MQCHLTAB** .. Se **MQCCDTURL** for configurado, a tabela de definição de canal do cliente para a qual ele aponta será usada... Para obter mais informações, consulte Acesso da URL para a CCDT
5. Se um arquivo `mqclient.ini` for definido e a sub-rotina Channels contiver atributos **ChannelDefinitionDirectory** e **ChannelDefinitionFile** , esses atributos serão usados para localizar a tabela de definições de canal do cliente.. Para obter mais informações, consulte IBM MQ MQI client arquivo de configuração, `mqclient.ini` e Sub-rotina de Canais do arquivo de configuração do cliente
6. Finalmente, se as variáveis de ambiente não forem configuradas, o cliente procurará uma tabela de definições de canal do cliente com um caminho e um nome que são estabelecidos a partir do atributo **DefaultPrefix** da sub-rotina AllQueueManagers no arquivo `mqs.ini` . Para obter mais informações, consulte a sub-rotina AllQueueManagers do arquivo mqs.ini..

   Se a procura por uma tabela de definição de canal de cliente falhar, o cliente usará os caminhos a seguir:

   - Linux AIX No AIX and Linux: `/var/mqm/AMQCLCHL.TAB`
   - Windows No Windows: `C:\Program Files\IBM\MQ\amqclchl.tab`
   - IBM i No IBM i: `/QIBM/UserData/mqm/@ipcc`
   - MQ Appliance No IBM MQ Appliance: *QMname*`_AMQCLCHL.TAB`. Eles aparecem sob o `mqbackup://` URI.

A primeira das opções descritas na lista anterior (usando o `ClientConnOffset` ou `ClientConnPtr` os campos de MQCNO) é suportada somente pela chamada MQCONNX. Se o aplicativo estiver usando MQCONN em vez de MQCONNX, as informações de canal serão procuradas das cinco maneiras restantes na ordem mostrada na lista. Se o cliente falhar para localizar as informações de canal, a chamada MQCONN ou MQCONNX falhará.

O nome do canal (para a conexão do cliente) deve corresponder ao nome do canal de conexão do servidor definido no servidor para que a chamada MQCONN ou MQCONNX seja bem-sucedida.

Conceitos relacionados

Acesso de endereço da web à tabela de definição de canal de cliente

Tarefas relacionadas

Configurando as Conexões entre o Servidor e o Cliente

Referências relacionadas

Tabela de Definições de Canal do Cliente

MQCNO-Opções de conexão

Os aplicativos cliente de conexão para gerenciadores de filas usando variáveis de ambiente

As informações de canal de cliente podem ser fornecidas para um aplicativo em execução em um ambiente do cliente por variáveis de ambiente.

Um aplicativo em execução em um ambiente do IBM MQ MQI client pode se conectar a um gerenciador de filas usando as variáveis de ambiente a seguir:

MQSERVER
: A variável de ambiente do **MQSERVER** é usada para definir um canal mínimo. **MQSERVER** especifica o local do servidor IBM MQ e o método de comunicação a ser usado..

MQCHLLIB
: A variável de ambiente **MQCHLLIB** especifica o caminho do diretório para o arquivo que contém a tabela de definições de canal do cliente (CCDT) O arquivo é criado no servidor, mas pode ser copiado na estação de trabalho do IBM MQ MQI client.

MQCHLTAB
: A variável de ambiente **MQCHLTAB** especifica o nome do arquivo contendo a tabela de definição de canal de cliente (CCDT).

A variável de ambiente **MQCCDTURL** fornece a capacidade equivalente para configurar uma combinação das variáveis de ambiente **MQCHLLIB** e **MQCHLTAB** . **MQCCDTURL** permite fornecer um arquivo, ftp ou URL http como um valor único a partir do qual uma tabela de definição de canal do cliente pode ser obtida. Para obter mais informações, veja Acesso de endereço da web à tabela de definição de canal do cliente.

Conectando aplicativos clientes a gerenciadores de filas usando a estrutura MQCNO

É possível especificar a definição do canal em uma estrutura de definição de canal (MQCD), que é fornecida usando a estrutura MQCNO da chamada MQCONNX.

Para obter mais informações, consulte Criando um canal de conexão do cliente no IBM MQ MQI client usando MQCNO.

Conectando aplicativos clientes a gerenciadores de filas usando uma tabela de definição de canal do cliente

Se você usar o comando MQSC DEFINE CHANNEL, os detalhes fornecidos serão colocados na tabela de definição de canal do cliente (ccdt). O conteúdo do parâmetro **QMgrName** da chamada MQCONN ou MQCONNX determina a qual gerenciador de filas o cliente se conecta.

Esse arquivo é acessado pelo cliente para determinar o canal que um aplicativo usará. Quando houver mais de uma definição de canal adequada, a opção de canal será influenciada pelos atributos do canal de peso do canal do cliente (CLNTWGHT) e de afinidade de conexão (AFFINITY).

Usando a reconexão automática do cliente

É possível fazer com que seus aplicativos clientes se reconectem automaticamente, sem gravar qualquer código adicional, configurando um número de componentes.

A reconexão do cliente automática é *sequencial*. A conexão é automaticamente restaurada em qualquer ponto no programa do aplicativo cliente, e as manipulações para abrir todos os objetos são restauradas.

Em contraste, reconexão manual requer que o aplicativo cliente recrie uma conexão utilizando MQCONN ou MQCONNX e reabra os objetos. A reconexão de cliente automática é adequada para muitos, mas não todos os aplicativos clientes.

Para obter mais informações, consulte Reconexão automática do cliente.

Função da tabela de definição de canal do cliente
A tabela de definição de canal do cliente (CCDT) contém definições de canais de conexão do cliente. Ela é especialmente útil se seus aplicativos clientes puderem precisar se conectar a diversos gerenciadores de fila alternativos.

A tabela de definição de canal do cliente é criada quando você define um gerenciador de filas. O mesmo arquivo pode ser usado por mais de um cliente IBM MQ.

Há várias maneiras para um aplicativo cliente utilizar um tabela de definição de canal de cliente. A tabela de definição de canal de cliente pode ser copiada para o computador cliente. É possível copiar o tabela de definição de canal de cliente para um local compartilhado por mais de um cliente. É possível disponibilizar o tabela de definição de canal de cliente acessível ao cliente como um arquivo compartilhado, enquanto ele permanece localizado no servidor.

A CCDT pode ser hospedada em um local central acessível por meio de um URI, removendo a necessidade de atualizar individualmente a CCDT para cada cliente implementado.

Conceitos relacionados
Acesso de endereço da web à tabela de definição de canal de cliente
Tarefas relacionadas
Acessando Definições de Canal de Conexão do Cliente
Referências relacionadas
Tabela de Definições de Canal do Cliente

Grupos de gerenciadores de filas na CCDT
É possível definir um conjunto de conexões na tabela de definição de canal de cliente (CCDT) como um *grupo de gerenciadores de filas*. É possível conectar um aplicativo a um gerenciador de filas que faz parte de um grupo de gerenciadores de filas. Isso pode ser feito colocando um prefixo no nome do gerenciador de filas em uma chamada MQCONN ou MQCONNX um asterisco.

Você pode optar por para definir conexões para mais de uma máquina servidor porque:

- Deseja se conectar a um cliente em qualquer um de um conjunto de gerenciadores de filas que está em execução, para melhorar a disponibilidade.
- Deseja reconectar um cliente ao mesmo gerenciador de filas ao qual ele se conectou com sucesso na última vez, mas se conectar a um gerenciador de filas diferente se a conexão falhar.
- Deseja poder tentar novamente uma conexão do cliente com um gerenciador de filas diferente se a conexão falhar, emitindo MQCONN no programa cliente novamente.
- Deseja reconectar automaticamente uma conexão do cliente com outro gerenciador de filas se a conexão falhar, sem escrever qualquer código do cliente.
- Deseja reconectar automaticamente uma conexão do cliente a uma instância diferente de um gerenciador de filas multi-instância se uma instância em espera assumir, sem escrever qualquer código do cliente.
- Deseja equilibrar suas conexões do cliente entre vários gerenciadores de filas, com mais clientes se conectando a alguns gerenciadores de filas que outros.
- Deseja difundir a reconexão de muitas conexões do cliente por vários gerenciadores de filas e ao longo do tempo, caso o alto volume de conexões cause uma falha.
- Deseja poder mover seus gerenciadores de filas sem mudar qualquer código do aplicativo cliente.

- Deseja gravar programas ao aplicativo cliente que não precisam conhecer nomes de gerenciadores de filas.

Nem sempre é apropriado se conectar a gerenciadores de filas diferentes. Um cliente transacional estendido ou um cliente Java no WebSphere Application Server, por exemplo, pode precisar se conectar a uma instância do gerenciador de filas previsível. A reconexão do cliente automática não é suportada pelo IBM MQ classes for Java.

Um grupo de gerenciadores de filas é um conjunto de conexões definidas na tabela de definição de canal de cliente (CCDT). O conjunto é definido por seus membros tendo o mesmo valor do atributo **QMNAME** em suas definições de canal.

Figura 97 na página 936 é uma representação gráfica de uma tabela de conexões do cliente, mostrando três grupos de gerenciadores de filas, dois grupos de gerenciadores de filas denominados gravados na CCDT como **QMNAME** (QM1) e **QMNAME** (QMGrp1), e um grupo em branco ou padrão gravado como **QMNAME** (' ').

1. O grupo de gerenciadores de filas QM1 tem três canais de conexão do cliente, conectando-o aos gerenciadores de filas QM1 e QM2. QM1 pode ser um gerenciador de filas multi-instância localizado em dois servidores diferentes.
2. O grupo de gerenciadores de filas padrão tem seis canais de conexão do cliente que o conectam a todos os gerenciadores de filas.
3. QMGrp1 tem canais de conexão do cliente para dois gerenciadores de filas, QM4 e QM5.

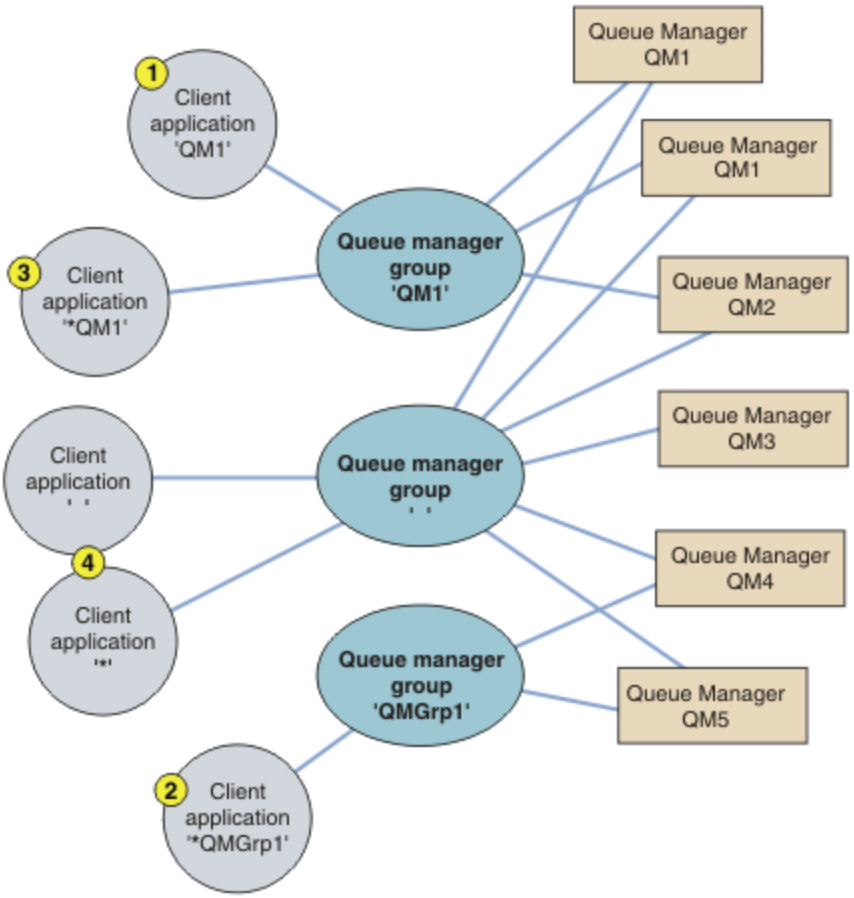


Figura 97. Grupos de gerenciadores de filas

Quatro exemplos de como usar essa tabela de conexões do cliente são descritos com a ajuda dos aplicativos clientes numerados em Figura 97 na página 936.

1. No primeiro exemplo, o aplicativo cliente passa um nome do gerenciador de filas, QM1, como o parâmetro **QmgrName** para sua chamada MQI MQCONN ou MQCONNX. O código do cliente IBM MQ

seleciona o grupo de gerenciadores de filas correspondente, QM1. O grupo contém três canais de conexão e o IBM MQ MQI client tenta se conecta ao QM1 usando cada um desses canais na sua vez até localizar um listener do IBM MQ para a conexão conectada a um gerenciador de filas em execução chamado QM1.

A ordem de tentativas de conexão depende do valor do atributo AFFINITY da conexão do cliente e dos pesos de canal do cliente. Dentro dessas restrições, a ordem de tentativas de conexão é aleatória, nas três conexões possíveis e ao longo do tempo, para difundir a carga de criação de conexões.

A chamada MQCONN ou MQCONNX emitida pelo aplicativo cliente é bem-sucedida quando uma conexão é estabelecida com uma instância em execução do QM1.

2. No segundo exemplo, o aplicativo cliente passa um nome do gerenciador de filas com um asterisco como prefixo, *QMGrp1, como o parâmetro **QmgrName** para sua chamada MQI MQCONN ou MQCONNX. O cliente IBM MQ seleciona o grupo de gerenciadores de filas correspondente, QMGrp1. Esse grupo contém dois canais de conexão de cliente e o IBM MQ MQI client tenta se conectar a *qualquer* gerenciador de filas usando cada canal em sua vez. Neste exemplo, o IBM MQ MQI client precisa fazer uma conexão bem-sucedida; o nome do gerenciador de filas ao qual ele se conecta não importa.

   A regra para a ordem de fazer tentativas de conexão é a mesma que antes. A única diferença é que ao colocar como prefixo no nome do gerenciador de filas um asterisco, o cliente indica que o nome do gerenciador de filas não é relevante.

   A chamada MQCONN ou MQCONNX emitida pelo aplicativo cliente é bem-sucedida quando uma conexão é estabelecida com uma instância em execução de qualquer gerenciador de filas conectado pelos canais no grupo de gerenciadores de filas QMGrp1.

3. O terceiro exemplo é essencialmente o mesmo que o segundo porque o parâmetro **QmgrName** tem como prefixo um asterisco, *QM1. O exemplo ilustra que não é possível determinar a qual gerenciador de filas uma conexão de canal do cliente irá se conectar inspecionando o atributo QMNAME em uma definição de canal por si só. O fato de que o atributo **QMNAME** da definição de canal é QM1 não é suficiente para requerer que uma conexão seja feita com um gerenciador de filas chamado QM1. Se seu aplicativo cliente colocar como prefixo de seu parâmetro **QmgrName** um asterisco, então, qualquer gerenciador de filas será um destino de conexão possível.

   Neste caso, as chamadas MQCONN ou MQCONNX emitidas pelo aplicativo cliente são bem-sucedidas quando uma conexão é estabelecida com uma instância em execução de QM1 ou QM2.

4. O quarto exemplo ilustra o uso do grupo padrão. Neste caso, o aplicativo cliente passa um asterisco, '*', ou um espaço em branco ' ', como o parâmetro **QmgrName** para sua chamada MQI MQCONN ou MQCONNX. Por convenção, na definição de canal do cliente, um atributo **QMNAME** significa que o grupo de gerenciadores de filas padrão e um parâmetro **QmgrName** em branco ou com asterisco corresponde a um atributo **QMNAME** em branco.

   Neste exemplo, o grupo de gerenciadores de filas padrão tem conexões de canal do cliente com todos os gerenciadores de filas. Selecionando o grupo de gerenciadores de filas padrão, o aplicativo pode ser conectado a qualquer gerenciador de filas no grupo.

   A chamada MQCONN ou MQCONNX emitida pelo aplicativo cliente é bem-sucedida quando uma conexão é estabelecida com uma instância em execução de qualquer gerenciador de filas.

   Nota: O grupo padrão é diferente de um gerenciador de filas padrão, embora um aplicativo use um parâmetro **QmgrName** em branco para se conectar ao grupo de gerenciadores de filas padrão ou ao gerenciador de filas padrão. O conceito de um grupo de gerenciadores de filas padrão é relevante somente para um aplicativo cliente e um gerenciador de filas padrão para um aplicativo do servidor.

Defina seus canais de conexão do cliente em somente um gerenciador de filas, incluindo os canais que se conectam a um segundo ou terceiro gerenciador de filas. Não os defina em dois gerenciadores de filas e, em seguida, tente mesclar as duas tabelas de definição de canal do cliente. Somente uma tabela de definição de canal de cliente pode ser acessada pelo cliente.

Exemplos

Consulte novamente a lista de razões para usar grupos de gerenciadores de filas no início do tópico. Como usar um grupo de gerenciadores de filas fornece esses recursos?

Conecte-se a qualquer um de um conjunto de gerenciadores de filas.
Defina um grupo de gerenciadores de filas com conexões para todos os gerenciadores de filas no conjunto e conecte-se ao grupo usando o parâmetro **QmgrName** com um asterisco como prefixo.

Reconecte-se ao mesmo gerenciador de filas, mas conecte-se a um diferente se o gerenciador de filas conectado na última vez estiver indisponível.
Defina um grupo de gerenciadores de filas como antes, mas configure o atributo **AFFINITY** (PREFERRED) em cada definição de canal do cliente.

Tente novamente uma conexão com outro gerenciador de filas se uma conexão falhar.
Conecte-se a um grupo de gerenciadores de filas e emita novamente a chamada MQI MQCONN ou MQCONNX se a conexão for interrompida ou se o gerenciador de filas falhar.

Reconecte-se automaticamente a outro gerenciador de filas se uma conexão falhar.
Conecte-se a um grupo de gerenciadores de filas usando a opção MQCONNX **MQCNO** MQCNO_RECONNECT.

Reconecte-se automaticamente a uma instância diferente de um gerenciador de filas multi-instância.
Faça o mesmo que no exemplo anterior. Neste caso, se desejar restringir o grupo de gerenciadores de filas para conectar-se às instâncias de um determinado gerenciador de filas multi-instância, defina o grupo com conexões somente para as instâncias do gerenciador de filas multi-instância.

Também é possível solicitar ao aplicativo cliente para emitir sua chamada MQI MQCONN ou MQCONNX sem asterisco como prefixo no parâmetro **QmgrName**. Dessa maneira, o aplicativo cliente pode se conectar somente ao gerenciador de filas denominado. Por fim, é possível configurar a opção **MQCNO** para MQCNO_RECONNECT_Q_MGR. Essa opção aceita reconexões com o mesmo gerenciador de filas que foi conectado anteriormente. Também é possível usar esse valor para restringir reconexões com a mesma instância de um gerenciador de filas normal.

Balanceie as conexões do cliente entre os gerenciadores de filas, com mais clientes conectados a alguns gerenciadores de filas que outros.
Defina um grupo de gerenciadores de filas e configure o atributo **CLNTWGHT** em cada definição de canal do cliente para distribuir as conexões desigualmente.

Difunda a carga de reconexão do cliente desigualmente e ao longo do tempo após uma falha de conexão ou do gerenciador de filas.
Faça o mesmo que no exemplo anterior. O IBM MQ MQI client escolhe a esmo reconexões entre gerenciadores de filas e espalha as reconexões ao longo do tempo.

Mova seu gerenciador de filas sem mudar qualquer código do cliente.
A CCDT isola seu aplicativo cliente do local do gerenciador de filas. A CCDT é um arquivo de dados que pode ser definido no cliente, lida a partir de um local compartilhado ou buscada a partir de um servidor da web. Para obter mais informações, veja Tabela de definição de canal de cliente.

Escreva um aplicativo cliente que não conheça os nomes dos gerenciadores de filas.
Use os nomes do grupo de gerenciadores de filas e estabeleça uma convenção de nomenclatura para os nomes do grupo de gerenciadores de filas que seja relevante para seus aplicativos clientes em sua organização e reflita a arquitetura de suas soluções em vez da nomenclatura dos gerenciadores de filas.

z/OS *Connecting to queue sharing groups*

You can connect your application to a queue manager that is part of a queue sharing group. This can be done by using the queue sharing group name instead of the queue manager name on the MQCONN or MQCONNX call.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

The client channel definition should use the queue sharing group generic interface to connect to an available queue manager in the group. For more information, see Connecting a client to a queue sharing group. A check is made to ensure that the queue manager the listener connects to is a member of the queue sharing group.

For more information on shared queues, see Shared queues and queue sharing groups.

Exemplos de peso e afinidade do canal
Esses exemplos ilustram como canais de conexão do cliente são selecionados quando `ClientChannelWeights` diferentes de zero são usados.

Os atributos de canal `ClientChannelWeight` e `ConnectionAffinity` controlam como canais de conexão do cliente são selecionados quando mais de um canal adequado está disponível para uma conexão. Esses canais são configurados para se conectarem a diferentes gerenciadores de filas para fornecer maior disponibilidade, balanceamento de carga de trabalho ou ambos. As chamadas MQCONN que poderiam resultar em uma conexão com um de vários gerenciadores de filas devem prefixar o nome do gerenciador de filas com um asterisco conforme descrito em: Exemplos de chamadas MQCONN: Exemplo 1. O nome do gerenciador de filas inclui um asterisco (*).

Os canais candidatos aplicáveis para uma conexão são aqueles em que o atributo QMNAME corresponde ao nome do gerenciador de filas especificado na chamada MQCONN. Se todos os canais aplicáveis para uma conexão tiverem um `ClientChannelWeight` de zero (o padrão) então eles são selecionados em ordem alfabética como no exemplo: Exemplos de chamadas MQCONN: Exemplo 1. O nome do gerenciador de filas inclui um asterisco (*).

Os exemplos a seguir ilustram o que acontece quando `ClientChannelWeights` diferentes de zero são usados. Observe que, como esse recurso envolve seleção de canal pseudoaleatória, os exemplos mostram uma sequência de ações que podem ocorrer em vez do que definitivamente ocorrerá.

Exemplo 1. Seleção de canais quando a ConnectionAffinity estiver configurado como PREFERRED
Esse exemplo ilustra como um IBM MQ MQI client seleciona um canal a partir de uma CCDT, em que o ConnectionAffinity está configurado para PREFERRED.

Neste exemplo, diversas máquinas clientes usam uma tabela de definição de canal de cliente (CCDT) fornecida por um gerenciador de filas. A CCDT inclui canais de conexão de cliente com os atributos a seguir (mostrados usando a sintaxe do comando DEFINE CHANNEL):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

O aplicativo emite MQCONN(*CORE)

O canal A não é um candidato para essa conexão, porque o atributo QMNAME não corresponde. Canais B, C e D são identificados como candidatos e são colocados em uma ordem de preferência baseada em seus pesos. Neste exemplo, a ordem pode ser C, B, D. O cliente tenta conectar-se ao gerenciador de filas em core2.ops.company.example. O nome do gerenciador de filas nesse endereço não é verificado, porque a chamada MQCONN incluiu um asterisco no nome do gerenciador de filas.

É importante observar que, com `AFFINITY(PREFERRED)`, toda vez que esta máquina cliente específica se conectar, colocará os canais na mesma ordem preferencial inicial. Isso se aplica mesmo quando as conexões são de processos diferentes ou em momentos diferentes.

Neste exemplo, o gerenciador de filas em core.2.ops.company.example não pode ser atingido. O cliente tenta se conectar a core1.ops.company.example porque o canal B é o próximo na ordem de preferência. Além disso, o canal C será rebaixado para se tornar o menos preferencial.

Uma segunda chamada MQCONN(*CORE) é emitida pelo mesmo aplicativo. O canal C foi rebaixado pela conexão anterior, portanto, o canal mais preferencial agora é B. Esta conexão é feita para core1.ops.company.example.

Uma segunda máquina que compartilha a mesma Tabela de definição de canal de cliente poderá colocar os canais em uma ordem de preferência inicial diferente. Por exemplo, D, B, C. Em circunstâncias normais, com todos os canais funcionando, os aplicativos nesta máquina estão conectados a core3.ops.company.example enquanto aqueles na primeira máquina estão conectados a core2.ops.company.example. Isso permite o balanceamento de carga de trabalho de um grande número de clientes entre vários gerenciadores de filas enquanto permite que cada cliente individual se conecte ao mesmo gerenciador de filas se ele estiver disponível.

Exemplo 2. Selecionando canais quando o ConnectionAffinity for configurado como NONE
Esse exemplo ilustra como um IBM MQ MQI client seleciona um canal a partir de uma CCDT, em que o ConnectionAffinity está configurado para NONE.

Neste exemplo, diversos clientes usam uma tabela de definição de canal de cliente (CCDT) fornecida por um gerenciador de filas. A CCDT inclui canais de conexão de cliente com os atributos a seguir (mostrados usando a sintaxe do comando DEFINE CHANNEL):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

O aplicativo emite MQCONN(*CORE). Como no exemplo anterior, o canal A não é considerado porque o QMNAME não corresponde. Canal B, C ou D são selecionados com base em seu peso, com probabilidades de 50%, 30% ou 20%. Neste exemplo, o canal B pode ser selecionado. Não há ordem de preferência persistente criada.

Uma segunda chamada MQCONN(*CORE) é feita. Novamente, um dos três canais aplicáveis é selecionado, com as mesmas probabilidades. Neste exemplo, o canal C é escolhido. No entanto, core2.ops.company.example não responde, portanto, outra opção é feita entre os canais candidatos restantes. O canal B é selecionado e o aplicativo é conectado a core1.ops.company.example.

Com AFFINITY(NONE), cada chamada MQCONN é independente de qualquer outra. Portanto, quando este aplicativo de exemplo faz uma terceira MQCONN(*CORE), ele pode mais uma vez tentar se conectar por meio do canal C interrompido, antes de escolher um dos B ou D.

Exemplos de chamadas MQCONN
Exemplos de uso do MQCONN para conectar-se a um gerenciador de filas específico ou para um de um grupo de gerenciadores de filas.

Em cada um dos exemplos a seguir, a rede é a mesma; há uma conexão definida para dois servidores do mesmo IBM MQ MQI client. (Nestes exemplos, a chamada MQCONNX poderia ser usada no lugar da chamada MQCONN.)

Existem dois gerenciadores de filas em execução nas máquinas servidores, um denominado SALE e o outro denominado SALE_BACKUP.

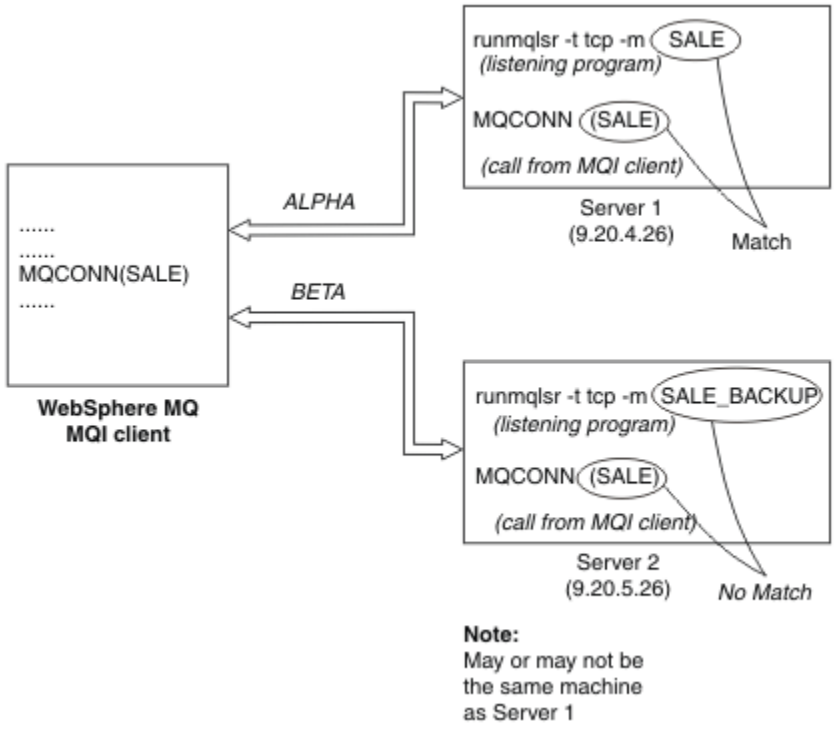


Figura 98. Exemplo de MQCONN

As definições para os canais nestes exemplos são:

Definições de SALE:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Definição de SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

As definições de canal do cliente podem ser resumidas conforme segue:

Nome	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

O que os exemplos de MQCONN demonstram
Os exemplos demonstram o uso de vários gerenciadores de filas como um sistema de backup.

Suponha que o link de comunicação para o Servidor 1 esteja temporariamente interrompido. O uso de vários gerenciadores de filas como um sistema de backup é demonstrado.

Cada exemplo cobre uma chamada MQCONN diferente e fornece uma explicação do que ocorre no exemplo específico apresentado, aplicando as regras a seguir:

1. A tabela de definição de canal de cliente (CCDT) é verificada em ordem alfabética de nome de canal para um nome de gerenciador de filas (campo QMNAME) correspondente àquele fornecido na chamada MQCONN.
2. Se uma correspondência for localizada, a definição de canal será usada.
3. Foi feita uma tentativa de iniciar o canal para na máquina identificada pelo nome de conexão (CONNAME). Se ela for bem-sucedida, o aplicativo continuará. Ele requer:
   - Um listener em execução no servidor.
   - O listener conectado ao mesmo gerenciador de filas que aquele ao qual o cliente deseja se conectar (se especificado).
4. Se a tentativa de iniciar o canal falhar e houver mais de uma entrada na tabela de definição de canal de cliente (neste exemplo, há duas entradas), o arquivo será procurado para uma correspondência adicional. Se uma correspondência for localizada, o processamento continuará na etapa 1.
5. Se nenhuma correspondência for localizada ou não houver mais entradas na tabela de definição de canal de cliente e o canal falhou ao ser iniciado, o aplicativo será incapaz de se conectar. Um código de razão apropriado e um código de conclusão são retornados na chamada MQCONN. O aplicativo pode executar uma ação com base nos códigos de razão e de conclusão retornados.

Exemplo 1. Nome do gerenciador de filas inclui um asterisco (*)
Neste exemplo, o aplicativo não está preocupado a qual gerenciador de filas ele se conecta. O aplicativo emite uma chamada MQCONN para um nome de gerenciador de filas incluindo um asterisco. Um canal adequado é escolhido.

O aplicativo emite:

```
MQCONN (*SALE)
```

Seguindo as regras, isto é o que acontece nessa instância:

1. A tabela de definição de canal de cliente (CCDT) é verificada para o nome de gerenciador de filas SALE, correspondente à chamada MQCONN do aplicativo.
2. Definições de canal para ALPHA e BETA são localizadas.
3. Se um canal tiver um valor de CLNTWGHT igual a 0, esse canal será selecionado. Se ambos tiverem um valor de CLNTWGHT igual a 0, o canal ALPHA será selecionado porque é o primeiro na sequência alfabética. Se ambos os canais tiverem um valor de CLNTWGHT diferente de zero, um canal será selecionado aleatoriamente, com base em seu peso.
4. Uma tentativa de iniciar o canal é feita.
5. Se o canal BETA foi selecionado, a tentativa de iniciá-lo será bem-sucedida.
6. Se o canal ALPHA foi selecionado, a tentativa de iniciá-lo NÃO será bem-sucedida porque o link de comunicação está interrompido. As etapas a seguir se aplicam:
   a. O único outro canal para o nome do gerenciador de filas SALE é BETA.
   b. Uma tentativa de iniciar esse canal é feita - ela é bem-sucedida.
7. Uma verificação para ver se um listener está em execução mostrar que há um em execução. Ele não está conectado ao gerenciador de filas SALE, mas como o parâmetro de chamada MQI tem um asterisco (*) incluído nele, nenhuma verificação é feita. O aplicativo é conectado ao gerenciador de filas SALE_BACKUP e continua o processamento.

Exemplo 2. Nome do gerenciador de filas especificado
Neste exemplo o aplicativo deve se conectar a um gerenciador de filas específico. O aplicativo emite uma chamada MQCONN para esse nome do gerenciador de filas. Um canal adequado é escolhido.

O aplicativo requer uma conexão com um gerenciador de filas específico, denominado SALE, conforme visto na chamada MQI:

```
MQCONN (SALE)
```

Seguindo as regras, isto é o que acontece nessa instância:

1. A tabela de definição de canal de cliente (CCDT) é verificada na sequência alfabética de nome de canal para o nome de gerenciador de filas SALE, correspondente à chamada MQCONN do aplicativo.
2. A primeira definição de canal localizada para correspondência é ALPHA.
3. É feita uma tentativa de iniciar o canal. Ela não é bem-sucedida porque o link de comunicação está quebrado.
4. A tabela de definição de canal de cliente é novamente verificada para o nome do gerenciador de filas SALE e o nome do canal BETA é localizado.
5. Uma tentativa de iniciar o canal é feita - ela é bem-sucedida.
6. Uma verificação para ver se um listener está em execução mostra que há um em execução, mas ele não está conectado ao gerenciador de filas SALE.
7. Não há entradas adicionais na tabela de definição de canal de cliente. O aplicativo não pode continuar e recebe o código de retorno MQRC_Q_MGR_NOT_AVAILABLE.

Exemplo 3. Nome do gerenciador de filas está em branco ou tem um asterisco (*)
Neste exemplo, o aplicativo não está preocupado a qual gerenciador de filas ele se conecta. O aplicativo emite uma chamada MQCONN especificando um nome de gerenciador de filas em branco ou um asterisco. Um canal adequado é escolhido.

Ele é tratado da mesma maneira que “Exemplo 1. Nome do gerenciador de filas inclui um asterisco (*)” na página 942.

Nota: Se esse aplicativo estava em execução em um ambiente diferente de um IBM MQ MQI client e o nome estava em branco, ele estaria tentando se conectar ao gerenciador de filas padrão. Esse não é o caso quando ele é executado de um ambiente do cliente; o gerenciador de filas acessado é aquele associado ao listener ao qual o canal se conecta.

O aplicativo emite:

```
MQCONN ("")
```

ou

```
MQCONN (*)
```

Seguindo as regras, isto é o que acontece nessa instância:

1. A tabela de definição de canal de cliente (CCDT) é verificada na sequência alfabética de nome de canal para um nome de gerenciador de filas que está em branco, correspondente à chamada MQCONN do aplicativo.
2. A entrada para o nome do canal ALPHA tem um nome de gerenciador de filas na definição de SALE. Isso não corresponde ao parâmetro de chamada MQCONN, que requer que o nome do gerenciador de filas esteja em branco.
3. A próxima entrada é para o nome do canal BETA.
4. O `queue manager name` na definição é SALE. Mais uma vez, isso não corresponde ao parâmetro de chamada MQCONN, que requer que o nome do gerenciador de filas esteja em branco.
5. Não há entradas adicionais na tabela de definição de canal de cliente. O aplicativo não pode continuar e recebe o código de retorno MQRC_Q_MGR_NOT_AVAILABLE.

Acionando no ambiente do cliente

As mensagens enviadas por aplicativos IBM MQ em execução no IBM MQ MQI clients contribuem para o acionamento exatamente da mesma maneira que qualquer outra mensagem e elas podem ser usadas para acionar programas no servidor e no cliente.

Acionamento é explicado em detalhe em Canais de acionamento.

O monitor acionador e o aplicativo a serem iniciados devem estar no mesmo sistema.

As características padrão da fila acionada são as mesmas que as no ambiente do servidor. Em particular, se nenhuma opção de controle de ponto de sincronização MQPMO for especificada em um aplicativo cliente colocando mensagens em uma fila acionada que seja local para um gerenciador de filas do z/OS , as mensagens serão colocadas em uma unidade de trabalho Se a condição de acionamento for então atendida, a mensagem do acionador será colocada na fila de inicialização na mesma unidade de trabalho e não poderá ser recuperada pelo monitor acionador até a unidade de trabalho ser finalizada. O processo que deve ser acionado não é iniciado até que a unidade de trabalho seja finalizada.

Definição de processo

Deve-se definir a definição do processo no servidor, pois isso está associado à fila que tem o acionamento configurado.

O objeto do processo define o que deve ser acionado. Se o cliente e o servidor não estiverem em execução na mesma plataforma, qualquer processo iniciado pelo monitor acionador deverá definir *ApplType*, caso contrário, o servidor usará suas definições padrão (ou seja, o tipo de aplicativo que está normalmente associado à máquina servidor) e causará uma falha.

Por exemplo, se o monitor acionador estiver sendo executado em um IBM MQ MQI client e deseja enviar uma solicitação para um servidor em outro sistema operacional, o MQAT_WINDOWS_NT deve ser definido, caso contrário o outro sistema operacional usa suas definições padrão e o processo falha.

Multi *Monitor acionador*

O monitor acionador fornecido pelo IBM MQ for Multiplatforms é executado nos ambientes clientes para sistemas Multiplatforms.

Para executar o monitor acionador, emita um destes comandos:

- IBM i No IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW Nas plataformas AIX, Linux, and Windows:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

A fila de inicialização padrão é SYSTEM.DEFAULT.INITIATION.QUEUE no gerenciador de filas padrão. A fila de inicialização é onde o monitor acionador procura mensagens do acionador. Ela então chama programas para as mensagens do acionador apropriadas. Esse monitor acionador suporta o tipo de aplicativo padrão e é o mesmo que `runmqtrm`, exceto que vincula as bibliotecas do cliente.

A sequência de comandos, construída pelo monitor acionador, é a seguinte:

1. O *ApplicId* da definição de processo relevante. *ApplicId* é o nome do programa a executar, como seria inserido na linha de comandos.
2. A estrutura MQTMC2, entre aspas, obtida a partir da fila de inicialização. Uma sequência de comandos é iniciada contendo essa sequência, exatamente conforme fornecida, entre aspas na ordem que o comando do sistema a aceita como um parâmetro.
3. O *EnvrData* da definição de processo relevante.

O monitor acionador não verifica se há outra mensagem na fila de inicialização até a conclusão do aplicativo que iniciou. Se o aplicativo tiver muito processamento a executar, o monitor acionador pode

não acompanhar o número de mensagens do acionador que chegam. Há duas maneiras de lidar com esta situação:

1. Ter mais monitores acionadores em execução

   Se optar por ter mais monitores acionadores em execução, será possível controlar o número máximo de aplicativos que podem ser executados a qualquer momento.

2. Executar os aplicativos iniciados em segundo plano

   Se optar por executar aplicativos em segundo plano, o IBM MQ não impõe nenhuma restrição quanto ao número de aplicativos que podem ser executados.

Para executar o aplicativo iniciado em segundo plano em sistemas AIX and Linux , você deve colocar um & (e comercial) no final do *EnvrData* da definição de processo.

aplicativos CICS (não z/OS)
Um programa de aplicativo não z/OS CICS que emite uma chamada MQCONN ou MQCONNX deve ser definido para CEDA como RESIDENT. Se você vincular novamente um aplicativo do servidor CICS como um cliente, você corre o risco de perder o suporte do ponto de sincronização.

Um programa de aplicativo não z/OS CICS que emite uma chamada MQCONN ou MQCONNX deve ser definido para CEDA como RESIDENT. Para tornar o código residente o menor possível, é possível vincular-se a um programa separado para emitir a chamada MQCONN ou MQCONNX.

Se a variável de ambiente MQSERVER for usada para definir a conexão do cliente, ela deverá ser especificada no arquivo CICSENV.CMD

Os aplicativosIBM MQ podem ser executados em um ambiente do servidor IBM MQ ou em um cliente IBM MQ sem mudar o código. No entanto, em um ambiente do servidor IBM MQ, o CICS pode agir como coordenador do ponto de sincronização e você usa EXEC CICS SYNCPOINT e EXEC CICS SYNCPOINT ROLLBACK em vez de **MQCMIT** e **MQBACK**. Se um aplicativo CICS for simplesmente revinculado como um cliente, o suporte ao ponto de sincronização será perdido. **MQCMIT** e **MQBACK** devem ser usados para o aplicativo em execução em um IBM MQ MQI client.

ALW Preparando e executando aplicativos CICS e Tuxedo

Para executar os aplicativos CICS e Tuxedo como aplicativos clientes, você usa bibliotecas diferentes daquelas usadas com aplicativos de servidor. O ID do usuário sob o qual o aplicativo é executado também é diferente.

Para preparar aplicativos CICS e Tuxedo para executar como aplicativos IBM MQ MQI client, siga as instruções em Configurando um cliente transacional estendido.

Observe, entretanto, que as informações que lidam especificamente com a preparação de aplicativos CICS e Tuxedo, incluindo os programas de amostra fornecidos com o IBM MQ, presumem que você esteja preparando aplicativos para executar em um sistema de servidor IBM MQ. Como resultado, as informações se referem apenas às bibliotecas do IBM MQ que são destinadas para uso em um sistema do servidor. Quando você estiver preparando seus aplicativos clientes, deve-se executar as ações a seguir:

- Use a biblioteca do sistema do cliente apropriada para as ligações de idioma que seu aplicativo usa. Por exemplo:
  - Linux AIX Para aplicativos escritos em C no AIX and Linux, use a biblioteca libmqic em vez de libmqm.
  - Windows Em sistemas Windows, use a biblioteca mqic.lib em vez de mqm.lib.
- Em vez das bibliotecas do sistema do servidor mostradas em Tabela 134 na página 946 e Tabela 135 na página 946, use as bibliotecas do sistema do cliente equivalentes. Se uma biblioteca do sistema do servidor não estiver listada nestas tabelas, use a mesma biblioteca em um sistema do cliente.

Tabela 134. Bibliotecas do sistema do cliente no AIX and Linux	
Biblioteca para um sistema do servidor IBM MQ	**Biblioteca equivalente para uso em um sistema do cliente IBM MQ**
libmqmxa	libmqcxa
V 9.4.0 V 9.4.0 libmqmxa64	V 9.4.0 V 9.4.0 libmqcxa64

Tabela 135. Bibliotecas do sistema do cliente em sistemas Windows	
Biblioteca para um sistema do servidor IBM MQ	**Biblioteca equivalente para uso em um sistema do cliente IBM MQ**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

O ID do usuário usado por um aplicativo cliente

Ao executar um aplicativo do servidor IBM MQ sob o CICS, normalmente ele alterna do usuário do CICS para o ID do usuário da transação. No entanto, quando você executa um aplicativo IBM MQ MQI client em CICS, ele retém a autoridade privilegiada do CICS.

ALW *Programas de amostra CICS e Tuxedo*

programas de amostra do CICS e Tuxedo para uso em sistemas AIX, Linux, and Windows.

Tabela 136 na página 946 lista os programas de amostra CICS e Tuxedo que são fornecidos para uso nos sistemas do cliente AIX and Linux. Tabela 137 na página 947 lista as informações equivalentes par a sistemas cliente Windows. As tabelas também listam os arquivos que são usados para preparar e executar os programas. Para obter uma descrição dos programas de amostra, consulte “A transação de amostra do CICS” na página 1094 e “Usando as amostras TUXEDO no AIX, Linux, and Windows” na página 1139.

Tabela 136. Programas de amostra para sistemas cliente AIX and Linux		
Descrição	**Origem**	**Módulo executável**
Programa CICS	amqscic0.ccs	amqscicc
Arquivo de cabeçalho para o programa CICS	amqscih0.h	-
Programa cliente Tuxedo para colocar mensagens	amqstxpx.c	-
Programa cliente Tuxedo para obter mensagens	amqstxgx.c	-
Programa do servidor Tuxedo para os dois programas clientes	amqstxsx.c	-
Arquivo UBBCONFIG para os programas Tuxedo	ubbstxcx.cfg	-
Arquivo de tabela do campo para os programas Tuxedo	amqstxvx.flds	-
Visualizar arquivo de descrição para os programas Tuxedo	amqstxvx.v	-

Tabela 137. Programas de amostra para sistemas cliente Windows		
Descrição	**Origem**	**Módulo executável**
Transação do CICS	amqscic0.ccs	amqscicc
Arquivo de cabeçalho para a transação do CICS	amqscih0.h	-
Programa cliente Tuxedo para colocar mensagens	amqstxpx.c	-
Programa cliente Tuxedo para obter mensagens	amqstxgx.c	-
Programa do servidor Tuxedo para os dois programas clientes	amqstxsx.c	-
Arquivo UBBCONFIG para os programas Tuxedo	ubbstxcx.cfg	-
Arquivo de tabela do campo para os programas Tuxedo	amqstxvx.fld	-
Visualizar arquivo de descrição para os programas Tuxedo	amqstxvx.v	-
Makefile para os programas Tuxedo	amqstxmc.mak	-
Arquivo ENVFILE para os programas Tuxedo	amqstxen.env	-

ALW *A mensagem de erro AMQ5203, conforme modificada para aplicativos CICS e Tuxedo*

Quando você executar aplicativos CICS ou Tuxedo que usam um cliente transacional estendido, você pode ver mensagens de diagnóstico padrão. Uma delas foi modificada para uso com um cliente transacional estendido

As mensagens que você pode ver nos arquivos de log de erro do IBM MQ são documentadas em Mensagens de diagnóstico: AMQ4000-9999. A mensagem AMQ5203 foi modificada para uso com um cliente transacional estendido. Aqui está o texto da mensagem modificada:

AMQ5203: Ocorreu um erro ao chamar a interface XA.

Explicação

O número do erro é &2, em que um valor de 1 indica que o valor de sinalizações fornecido de &1 era inválido, 2 indica que houve uma tentativa de usar bibliotecas encadeadas e não encadeadas no mesmo processo, 3 indica que houve um erro com o nome do gerenciador de filas fornecido '&3', 4 indica que o ID do gerenciador de recursos de &1 era inválido, 5 indica que uma tentativa foi feita para usar um segundo gerenciador de filas chamado '&3' quando outro gerenciador de filas já estava conectado, 6 indica que o Gerenciador de Transações foi chamado quando o aplicativo não está conectado a um gerenciador de filas, 7 indica que a chamada XA foi feita enquanto outra chamada estava em andamento, 8 indica que a sequência xa_info '&4' na chamada xa_open continha um valor de parâmetro inválido para o nome do parâmetro '&5' e 9 indica que a sequência xa_info '&4' na chamada xa_open está omitindo um parâmetro necessário, nome do parâmetro '&5'.

Resposta do usuário

Corrija o erro e tente a operação novamente.

Windows Preparando e executando aplicativos Microsoft Transaction Server

Para preparar um aplicativo MTS para que seja executado como um aplicativo IBM MQ MQI client, siga estas instruções conforme apropriado para o seu ambiente.

Para obter informações gerais sobre como desenvolver aplicações do Microsoft Transaction Server (MTS) que acessam os recursos do IBM MQ, consulte a seção sobre MTS no IBM MQ Help Center.

Para preparar um aplicativo MTS para que seja executado como um aplicativo IBM MQ MQI client, faça o seguinte para cada componente do aplicativo:

- Se o componente usar as ligações de linguagem C para o MQI, siga as instruções no "Preparando programas C no Windows" na página 1029, mas vincule o componente à biblioteca mqicxa.lib em vez da mqic.lib.
- Se o componente usar as classes IBM MQ C++, siga as instruções em "Construindo programas C++ no Windows" na página 561, mas vincule o componente à biblioteca imqx23vn.lib em vez da imqc23vn.lib.
- Se o componente usar as ligações de linguagem Visual Basic para o MQI, siga as instruções no "Preparando programas Visual Basic no Windows" na página 1032, mas quando definir o projeto Visual Basic, digite MqType=3 no campo **Argumentos de compilação condicional**.

Preparando e executando aplicativos IBM MQ JMS

É possível executar aplicativos IBM MQ JMS no modo cliente, com WebSphere Application Server como seu gerenciador de transações. Você pode ver certas mensagens de aviso.

Para preparar e executar aplicativos IBM MQ JMS no modo cliente, com o WebSphere Application Server como seu gerenciador de transações, siga as instruções em "Usando IBM MQ classes for JMS/Jakarta Messaging" na página 83.

Ao executar um aplicativo cliente IBM MQ JMS, você poderá ver as mensagens de aviso a seguir:

MQJE080
Unidades de licença insuficientes – execute setmqcap

MQJE081
O arquivo que contém as informações da unidade de licença está no formato errado – execute setmqcap

MQJE082
O arquivo que contém as informações da unidade de licença não pode ser localizado – execute setmqcap

Saídas de usuário, saídas de API e serviços instaláveis do IBM MQ

Este tópico contém links para informações sobre como usar e desenvolver esses programas.

Para uma introdução sobre como você pode usar saídas de usuário, saídas de API e serviços instaláveis para estender as instalações do gerenciador de filas, consulte Ampliando as instalações do gerenciador.

Para obter informações sobre como gravar e compilar saídas e serviços instaláveis, consulte os subtópicos.

Conceitos relacionados
Programas de Saída de Canal para Canais MQI
Referências relacionadas
Referência de saída de API
Informações de referência da interface de serviços instaláveis
IBM i Informações de referência da interface de serviços instaláveis no IBM i

ALW Composição de saídas e serviços instaláveis em AIX, Linux, and Windows

É possível escrever e compilar saídas sem vinculação a qualquer bibliotecas IBM MQ no AIX, Linux, and Windows.

Sobre esta tarefa

Este tópico se aplica aos sistemas AIX, Linux, and Windows somente. Para obter detalhes sobre a composição de saídas e serviços instaláveis para outras plataformas, consulte os tópicos específicos da plataforma relevante.

Se o IBM MQ for instalado em um local não padrão, deve-se escrever e compilar suas saídas sem vinculação a qualquer biblioteca do IBM MQ.

É possível escrever e compilar saídas nos sistemas AIX, Linux, and Windows sem vincular qualquer uma dessas bibliotecas do IBM MQ:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

Saídas existentes vinculadas a essas bibliotecas continuam funcionando, contanto que nos sistemas AIX and Linux o IBM MQ esteja instalado no local padrão.

Procedimento

1. Inclua o arquivo de cabeçalho cmqec.h.

   Incluir esse arquivo de cabeçalho inclui automaticamente os arquivos de cabeçalho cmqc.h, cmqxc.h e cmqzc.h.
2. Escreva a saída de forma que as chamadas MQI e DCI sejam feitas por meio da estrutura MQIEP. Para obter mais informações sobre a estrutura MQIEP, consulte Estrutura MQIEP.
   - Serviços instaláveis
     - Use o parâmetro **Hconfig** para apontar para a chamada MQZEP.
     - Deve-se verificar se os primeiros 4 bytes de **Hconfig** correspondem ao **StrucId** da estrutura MQIEP antes de usar o parâmetro **Hconfig**.
     - Para obter mais informações sobre como escrever componentes de serviço instaláveis, consulte MQIEP.
   - Saídas de API
     - Use o parâmetro **Hconfig** para apontar para a chamada MQXEP.
     - Deve-se verificar se os primeiros 4 bytes de **Hconfig** correspondem ao **StrucId** da estrutura MQIEP antes de usar o parâmetro **Hconfig**.
     - Para obter mais informações sobre como escrever saídas de API, consulte “Escrevendo saídas de API” na página 966.
   - Saídas do canal
     - Use o parâmetro **pEntryPoints** da estrutura MQCXP para apontar para as chamadas MQI e DCI.
     - Deve-se verificar se o número da versão de MQCXP está na versão 8 ou superior antes de usar **pEntryPoints**.
     - Para obter mais informações sobre como escrever saídas de canal, consulte “Gravando programas de saída do canal” na página 976.
   - Saídas de conversão de dados
     - Use o parâmetro **pEntryPoints** da estrutura MQDXP para apontar para as chamadas MQI e DCI.
     - Deve-se verificar se o número da versão de MQDXP está na versão 2 ou superior antes de usar **pEntryPoints**.
     - É possível usar o comando **crtmqcvx** e o arquivo de origem amqsvfc0.c para criar o código de conversão de dados que usa o parâmetro **pEntryPoints**. Consulte “Gravando uma saída de conversão de dados para IBM MQ for Windows” na página 1003 e “Escrevendo uma saída de conversão de dados para sistemas IBM MQ for AIX or Linux” na página 1000.

- Se houver saídas de conversão de dados existentes que foram geradas usando o comando **crtmqcvx**, deve-se gerar novamente a saída usando o comando atualizado.
- Para obter mais informações sobre como escrever saídas de conversão de dados, consulte "Escrevendo saídas de conversão de dados" na página 995.

- Saídas de pré-conexão
  - Use o parâmetro **pEntryPoints** da estrutura MQNXP para apontar para chamadas MQI e DCI.
  - Deve-se verificar se o número da versão MQNXP está na versão 2 ou superior antes de usar **pEntryPoints**.
  - Para obter mais informações sobre como escrever saídas de pré-conexão, consulte "Fazendo referência a definições de conexão usando uma saída de pré-conexão de um repositório" na página 1005.
- Saídas de publicação
  - Use o parâmetro **pEntryPoints** da estrutura MQPSXP para apontar para as chamadas MQI e DCI.
  - Deve-se verificar se o número da versão MQPSXP está na versão 2 ou superior antes de usar **pEntryPoints**.
  - Para obter mais informações sobre como escrever saídas de publicação, consulte "Escrevendo e compilando saídas de publicação" na página 1007.
- Saídas de carga de trabalho do cluster
  - Use o parâmetro **pEntryPoints** da estrutura MQWXP para apontar para chamadas MQXCLWLN.
  - Deve-se verificar se o número da versão de MQWXP está na versão 4 ou superior antes de usar **pEntryPoints**.
  - Para obter mais informações sobre como escrever saídas de carga de trabalho de cluster, consulte "Gravando e Compilando Saídas de Carga de Trabalho do Cluster" na página 1009.

Por exemplo, em uma saída do canal que chama MQPUT:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Exemplos adicionais podem ser vistos em "Usando os programas processuais de amostra do IBM MQ" na página 1070.

3. Compile a saída:
   - Não vincule às bibliotecas do IBM MQ.
   - Não inclua um RPath integrado em qualquer biblioteca do IBM MQ em sua saída.
   - Para obter mais informações sobre como compilar sua saída, consulte um dos tópicos a seguir:
     - Saídas de API: "Compilando saídas de API" na página 968.
     - Saídas de canal, saídas de publicação, saídas de carga de trabalho do cluster: "Compilando programas de saída de canais em sistemas AIX, Linux, and Windows" na página 994.
     - Saídas de conversão de dados: "Escrevendo saídas de conversão de dados" na página 995.
4. Coloque a saída em um dos locais a seguir:
   - Um caminho de sua escolha que você qualifique totalmente ao configurar a saída
   - O caminho de saída padrão, em um diretório de instalação específico. Por exemplo, *MQ_DATA_PATH*/exits/installation2.
   - O caminho de saída padrão

O caminho de saída padrão é *MQ_DATA_PATH*/exits para saídas de 32 bits e *MQ_DATA_PATH*/exits64 para saídas de 64 bits. É possível mudar esses caminhos no arquivo qm.ini ou mqclient.ini. Para obter mais informações, consulte Saída do caminho. No Windows e no Linux, é possível usar o IBM MQ Explorer para mudar o caminho:

a. Clique com o botão direito no nome do gerenciador de filas
b. Clique em **Propriedades...**
c. Clique em **Saídas**
d. No campo do caminho padrão de saídas, especifique o nome do caminho do diretório que retém o programa de saída.

Se uma saída for colocada em um diretório de instalação específico e no diretório de caminho padrão, a saída do diretório de instalação específico será usada pela instalação do IBM MQ denominada no caminho. Por exemplo, a saída é colocada em /exits/installation2 e em /exits, mas não em /exits/installation1. A instalação do IBM MQ installation2 usa a saída de /exits/installation2. A instalação do IBM MQ installation1 usa a saída do diretório /exits.

5. Se necessário, configure a saída:
   - Serviços instaláveis: "Configurando serviços e componentes" na página 959.
   - Saídas de API: "Configurando saídas de API" na página 971.
   - Saídas do canal: "Configurando saídas do canal" na página 995.
   - Saídas de publicação: "Configurando saídas de publicação" na página 1008.
   - Saídas de pré-conexão: Sub-rotina PreConnect do arquivo de configuração do cliente.

ALW *Saídas de API não vinculadas a uma biblioteca do MQI*

Em determinadas circunstâncias, é necessário vincular sua saída de API existente, que não pode ser recodificada para usar os ponteiros de função MQIEP, a uma biblioteca de API do IBM MQ.

Isso é necessário para que a sua saída de API existente possa ser carregada com sucesso pelo vinculador de tempo de execução do seu sistema em programas que ainda não possuem ponteiros de função carregados.

Nota: Essas informações são limitadas às saídas de API existentes que fazem chamadas MQI diretamente. Ou seja, as saídas que não usam MQIEP. Onde possível, é necessário planejar recodificar a saída para usar os pontos de entrada de MQIEP.

runmqsc é um exemplo de um programa que não se vincula diretamente a uma biblioteca MQI..

Portanto, uma saída de API que não foi vinculada à sua biblioteca de API necessária do IBM MQ ou recodificada para usar o MQIEP falha ao ser carregada no **runmqsc**.

Você vê erros no log de erro do gerenciador de filas, por exemplo, AMQ6175: o sistema não pôde carregar dinamicamente a biblioteca compartilhada, junto com o texto de qualificação, como undefined symbol: MQCONN.

e AMQ7214: o módulo para a Saída de API 'myexitname' não pôde ser carregado.

Tarefas relacionadas

"Composição de saídas e serviços instaláveis em AIX, Linux, and Windows" na página 948
É possível escrever e compilar saídas sem vinculação a qualquer bibliotecas IBM MQ no AIX, Linux, and Windows.

ALW Serviços e componentes instaláveis para o AIX, Linux, and Windows

Esta seção apresenta os serviços instaláveis e as funções e os componentes associados a eles. A interface para essas funções é documentada para que você ou os fornecedores de software possam fornecer os componentes.

As principais razões para fornecer serviços instaláveis do IBM MQ são:

- Para fornecer a flexibilidade de escolher se deseja usar os componentes fornecidos por produtos IBM MQ ou substituir ou aumentá-los com outras pessoas.
- Para permitir que os fornecedores participem, fornecendo componentes que podem usar novas tecnologias, sem fazer mudanças internas em produtos IBM MQ.
- Permitir que o IBM MQ explore novas tecnologias de forma mais rápida e mais barata e, assim, fornecer produtos antes e a preços mais baixos.

Serviços instaláveis e *componentes de serviço* fazem parte da estrutura do produto IBM MQ. No centro desta estrutura está a parte do gerenciador de filas que implementa a função e as regras associadas ao Message Queue Interface (MQI). Essa parte central requer inúmeras funções de serviço, denominadas *serviços instaláveis*, para executar seu trabalho. Os serviços instaláveis são:

- Serviço de autorização
- Serviço de Nomes

Cada serviço instalável é um conjunto relacionado de funções implementadas usando um ou mais *componentes de serviços*. Cada componente é chamado usando uma interface publicamente disponível e corretamente arquitetada. Isso permite que fornecedores de software independentes e outros terceiros forneçam componentes instaláveis para aumentar ou substituir os fornecidos pelos produtos IBM MQ. Tabela 138 na página 952 resume os serviços e componentes que podem ser usados.

Tabela 138. Resumo dos componentes de serviços instaláveis

Serviço instalável	**Componente fornecido**	**Função**	**Requisitos**
Serviço de autorização	object authority manager (OAM)	Fornece verificação de autorização sobre os comandos e as chamadas MQI. Os usuários podem gravar seu próprio componente para aumentar ou substituir o OAM. Por exemplo, para verificar se um ID do usuário tem autoridade para abrir uma fila.	(Instalações de autorização de plataforma apropriadas são presumidas)
Serviço de Nomes	Nenhum	Fornece suporte para o gerenciador de filas para consultar o nome do gerenciador de filas que possui uma fila especificada. • Definido pelo usuário	• Um gerenciador de nomes gravado pelo usuário ou terceiros

A interface de serviços instaláveis é descrita em Informações de referência da interface de serviços instaláveis.

Tarefas relacionadas

Configurando serviços instaláveis

Gravando um componente de serviço

Esta seção descreve o relacionamento entre os serviços, componentes, pontos de entrada e códigos de retorno.

Funções e componentes

Cada serviço consiste em um conjunto de funções relacionadas. Por exemplo, o serviço de nomes contém funções para:

- Consultar um nome de filas e retornar o nome do gerenciador de filas no qual a fila está definida
- Inserir um nome da fila no diretório de serviço
- Excluir um nome da fila do diretório do serviço

Ele também contém as funções de inicialização e finalização.

É fornecido um serviço instalável por um ou mais componentes de serviço. Cada componente pode executar algumas ou todas as funções que são definidas para esse serviço. Por exemplo, no IBM MQ for AIX, o componente de serviço de autorização fornecido, o OAM, executa todas as funções disponíveis. Consulte “Interface de serviço de autorização” na página 956 para obter informações adicionais. O componente também é responsável por gerenciar quaisquer recursos subjacentes ou software (por exemplo, um diretório LDAP) que precisarem implementar o serviço. Os arquivos de configuração fornecem uma maneira padrão de carregar o componente e de determinar os endereços das rotinas funcionais que fornece.

O Figura 99 na página 953 mostra como os serviços e os componentes estão relacionados:

- Um serviço é definido para um gerenciador de filas por sub-rotinas em um arquivo de configuração.
- Cada serviço é suportado pelo código fornecido no gerenciador de filas. Os usuários não podem mudar esse código e, portanto, não podem criar seus próprios serviços.
- Cada serviço é implementado por um ou mais componentes; eles podem ser fornecidos com o produto ou gravados pelo usuário. Podem ser chamados diversos componentes para um serviço, cada um suportando diferentes recursos no serviço.
- Os pontos de entrada conectam os componentes de serviço ao código de suporte no gerenciador de filas.

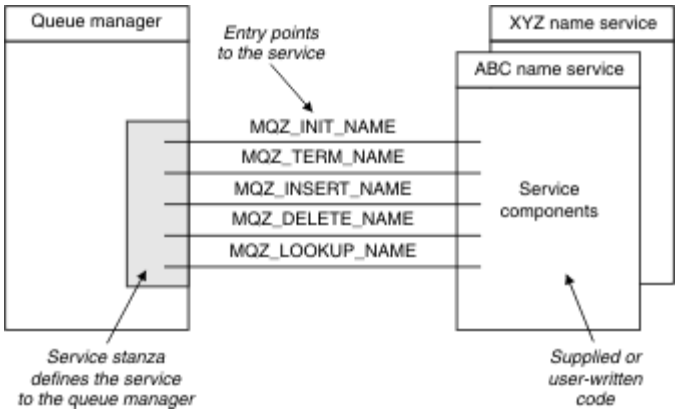


Figura 99. Entendendo serviços, componentes e pontos de entrada

Pontos de entrada

Cada componente de serviço é representado por uma lista de endereços de ponto de entrada das rotinas que suportam um determinado serviço instalável. O serviço instalável define a função a ser executada por cada rotina.

A ordenação dos componentes de serviço quando eles são configurados define a ordem na qual os pontos de entrada são chamados em uma tentativa de atender a uma solicitação para o serviço.

No arquivo de cabeçalho `cmqzc.h` fornecido, os pontos de entrada fornecidos para cada serviço possuem um prefixo MQZID_.

Se os serviços estiverem presentes, os serviços serão carregados em uma ordem predefinida. A lista a seguir mostra os serviços e a ordem na qual são inicializados.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

O serviço AuthorizationService é o único que está configurado por padrão. Configure o NameService e o UserIdentifierService manualmente, se desejar usá-los.

Serviços e os componentes de serviço possuem um mapeamento de um-para-um ou um-para-vários. Diversos componentes de serviço podem ser definidos para cada serviço. Nos sistemas AIX and Linux, o valor de serviço da sub-rotina ServiceComponent deve corresponder ao valor do nome da sub-rotina de serviço no arquivo qm.ini. No Windows, o valor da chave de registro de serviço do ServiceComponent deve corresponder ao valor da chave de registro de nome e é definido como: HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation*MQ_INSTALLATION_NAME*\Configuration\QueueManager*qmname*\ em que *qmname* é o nome do gerenciador de filas.

Para sistemas AIX and Linux, os componentes de serviço são iniciados na ordem em que são definidos no arquivo qm.ini. No Windows, como o registro do Windows é usado, o IBM MQ emite uma chamada **RegEnumKey** que retorna os valores em ordem alfabética. Portanto, no Windows os serviços são chamados em ordem alfabética, conforme são definidos no registro.

A ordem das definições de ServiceComponent é significativa. Essa ordem dita a ordem na qual os componentes são executados para um determinado serviço. Por exemplo, o AuthorizationService no Windows é configurado com o componente OAM padrão chamado MQSeries.WindowsNT.auth.service. Componentes adicionais podem ser definidos para este serviço para substituir o OAM padrão. A menos que MQCACF_SERVICE_COMPONENT seja especificado, o primeiro componente encontrado na ordem alfabética será usado para processar a solicitação e o nome para esse componente será usado.

Códigos de retorno

Os componentes de serviço fornecem códigos de retorno para o gerenciador de filas relatar em várias condições. Eles relatam o sucesso ou falha da operação e indicam se o gerenciador de filas continuará para o próximo componente de serviço. Um parâmetro *Continuation* separado transporta essa indicação.

Dados do componente

Um único componente de serviço pode requerer que os dados sejam compartilhados entre as suas várias funções. Serviços instaláveis fornecem uma área de dados opcional a ser passada em cada chamada de um componente de serviço. Essa área de dados é para uso exclusivo do componente de serviço. Ela é compartilhada por todas as chamadas de uma determinada função, mesmo se forem feitas a partir de processos ou espaços de endereço diferentes. É garantido que isso seja endereçável a partir do componente de serviço sempre que for chamado. Deve-se declarar o tamanho dessa área na sub-rotina *ServiceComponent*.

Inicialização e finalização de componentes
O uso das opções de inicialização e finalização de componentes.

Quando a rotina de inicialização do componente é chamada, ela deve chamar a função **MQZEP** do gerenciador de filas para cada ponto de entrada suportado pelo componente. **MQZEP** define um ponto de entrada para o serviço. Todos os pontos de saída indefinidos são presumidos como NULL.

Um componente é sempre chamado uma vez com a opção de inicialização primária, antes de ser chamado de qualquer outra maneira.

Um componente pode ser chamado com a opção de inicialização secundária em determinadas plataformas. Por exemplo, ele pode ser chamado uma vez para cada processo de sistema operacional, encadeamento ou tarefa pelo qual o serviço é acessado.

Se a inicialização secundária for usada:

- O componente pode ser chamado mais de uma vez para a inicialização secundária. Para cada chamada desse tipo, uma chamada correspondente para a finalização secundária é emitida quando o serviço não é mais necessário.

  Para serviços de nomenclatura, essa é a chamada MQZ_TERM_NAME.

Para serviços de autorização, essa é a chamada MQZ_TERM_AUTHORITY.

- Os pontos de entrada devem ser especificados novamente (chamando MQZEP) toda vez que o componente for chamado para inicialização primária e secundária.
- Somente uma cópia de dados do componente é usada para o componente; não há uma cópia diferente para cada inicialização secundária.
- O componente não é chamado para qualquer outra chamada ao serviço (do processo de sistema operacional, encadeamento ou tarefa, conforme apropriado) antes da realização da inicialização secundária.
- O componente deve configurar o parâmetro **Version** para o mesmo valor para a inicialização primária e secundária.

O componente é sempre chamado com a opção de finalização primária uma vez, quando não for mais necessário. Nenhuma chamada adicional será feita para esse componente.

O componente é chamado com a opção de finalização secundária se tiver sido chamado para inicialização secundária.

Gerenciador de autoridade de objeto (OAM)
O componente de serviço de autorização fornecido com os produtos IBM MQ é chamado de Gerenciador de Autoridade de Objeto (OAM).

Por padrão, o OAM fica ativo e funciona com os comandos de controle **dspmqaut** (exibir autoridade), **dmpmqaut** (fazer dump de autoridade) e **setmqaut** (configurar ou reconfigurar autoridade).

A sintaxe desses comandos e como usá-los são descritos em Administrando IBM MQ for Multiplatforms usando comandos de controle

O OAM trabalha com a *entidade* de um diretor ou grupo:

- Linux AIX Nos sistemas AIX and Linux, um diretor é um ID do usuário ou um ID associado a um programa de aplicativo em execução em nome de um usuário; um grupo é uma coleção definida pelo sistema de diretores.
- Windows Nos sistemas Windows, um diretor é um ID do usuário do Windows ou um ID associado a um programa de aplicativo em execução em nome de um usuário; um grupo é um grupo do Windows.

Autorizações podem ser concedidas ou revogadas no nível do diretor ou do grupo.

Quando uma solicitação MQI é feita ou um comando é emitido, o OAM verifica se a entidade associada à operação tem autorização para executar a operação solicitada e para acessar os recursos do gerenciador de filas especificado.

O serviço de autorização permite aumentar ou substituir a verificação de autoridade fornecida para gerenciadores de filas, gravando seu próprio componente de serviço de autorização.

Serviço de Nomes
O serviço de nomes é um serviço instalável que fornece suporte para o gerenciador de filas consultar o nome do gerenciador de filas que possui uma fila especificada. Nenhum outro atributo de fila pode ser recuperado de um serviço de nomes.

O serviço de nomes permite que um aplicativo abra filas remotas para saída como se fossem filas locais. Um serviço de nomes não é chamado para objetos diferentes de filas.

Nota: As filas remotas devem ter seu atributo **Scope** configurado como CELL.

Quando um aplicativo abre uma fila, ele procura o nome da primeira fila no diretório do gerenciador de filas. Se não a localizar lá, ele procura em todos os serviços de nomes que foram configurados, até que localize um que reconheça o nome da fila. Se nenhum reconhecer o nome, a abertura falhará.

O serviço de nomes retorna o gerenciador de filas proprietário dessa fila. O gerenciador de filas continua, então, com a solicitação MQOPEN como se o comando tivesse especificado a fila e nome do gerenciador de filas na solicitação original.

A interface de serviço de nomes (NSI) faz parte da estrutura do IBM MQ.

Como o serviço de nomes funciona

Se uma definição de fila especificar o atributo **Scope** como gerenciador de filas, ou seja, SCOPE(QMGR) no MQSC, a definição de fila (juntamente com todos os atributos da fila) será armazenada somente no diretório do gerenciador de filas. Isso não pode ser substituído por um serviço instalável.

Se uma definição de fila especificar o atributo **Scope** como célula, ou seja, SCOPE(CELL) no MQSC, a definição de fila será novamente armazenada no diretório do gerenciador de filas, juntamente com todos os atributos da fila. No entanto, o nome da fila e do gerenciador de filas também são armazenados em um serviço de nomes. Se não houver nenhum serviço disponível que possa armazenar essas informações, uma fila com o *Scope* célula não poderá ser definida.

O diretório no qual as informações são armazenadas pode ser gerenciado pelo serviço ou o serviço pode usar um serviço subjacente, por exemplo, um diretório LDAP, para esse propósito. Em ambos os casos, as definições armazenadas no diretório devem persistir, mesmo após o componente e o gerenciador de filas serem finalizados, até serem excluídos explicitamente.

Nota:

1. Para enviar uma mensagem a uma definição de fila local de um host remoto (com um escopo de CELL) em um gerenciador de filas diferente em uma célula de diretório de nomenclatura, será necessário definir um canal.
2. Não é possível obter mensagens diretamente da fila remota, mesmo quando ela tem um escopo de CELL.
3. Nenhuma definição de fila remota é necessária ao enviar para uma fila com um escopo de CELL.
4. O serviço de nomenclatura define centralmente a fila de destino, embora você ainda precise de uma fila de transmissão para o gerenciador de filas de destino e um par de definições de canal. Além disso, a fila de transmissão no sistema local deve ter o mesmo nome que o gerenciador de filas proprietário da fila de destino, com o escopo de célula, no sistema remoto.

   Por exemplo, se o gerenciador de filas remotas tiver o nome QM01, a fila de transmissão no sistema local também deve ter o nome QM01.

Interface de serviço de autorização
O serviço de autorização fornece pontos de entrada para uso pelo gerenciador de filas.

Os pontos de entrada são os seguintes:

MQZ_AUTHENTICATE_USER
Autentica um ID do usuário e senha e pode configurar campos de contexto de identidade.

MQZ_CHECK_AUTHORITY
Verifica se uma entidade possui autoridade para desempenhar uma ou mais operações em um objeto especificado.

MQZ_CHECK_PRIVILEGED
Verifica se um usuário especificado é um usuário privilegiado.

MQZ_COPY_ALL_AUTHORITY
Copia todas as autorizações atuais que existem para um objeto referenciado para outro objeto.

MQZ_DELETE_AUTHORITY
Exclui todas as autorizações associadas a um objeto especificado.

MQZ_ENUMERATE_AUTHORITY_DATA
Recupera todos os dados de autoridade que correspondem aos critérios de seleção especificados.

MQZ_FREE_USER
Libera recursos alocados associados.

MQZ_GET_AUTHORITY
Obtém a autoridade que uma entidade tem para acessar um objeto especificado.

MQZ_GET_EXPLICIT_AUTHORITY
Obtém a autoridade que um grupo denominado possui para acessar um objeto especificado (mas sem a autoridade adicional do grupo **nobody**) ou a autoridade que o grupo principal do proprietário nomeado possui para acessar um objeto especificado.

MQZ_INIT_AUTHORITY
Inicializa componente de serviço de autorização.

MQZ_INQUIRE
Consulta a funcionalidade suportada do serviço de autorização.

MQZ_REFRESH_CACHE
Atualizar todas as autorizações.

MQZ_SET_AUTHORITY
Define a autoridade que uma entidade tem para um objeto especificado.

MQZ_TERM_AUTHORITY
Finaliza o componente de serviço de autorização.

Além disso, no IBM MQ for Windows, o serviço de autorização fornece os pontos de entrada a seguir para uso pelo gerenciador de filas:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

Esses pontos de entrada suportam o uso do Windows Security Identifier (NT SID).

Estes nomes são definidos como **typedef**, no arquivo de cabeçalho `cmqzc.h`, que pode ser usado para o protótipo das funções do componente.

A função de inicialização (**MQZ_INIT_AUTHORITY**) deve ser o ponto de entrada principal para o componente. As outras funções são chamadas por meio do endereço do ponto de entrada que a função de inicialização incluiu no vetor do ponto de entrada do componente.

Interface de serviço de nomes
Um nome do serviço fornece pontos de entrada para uso pelo gerenciador de filas.

Os pontos de entrada a seguir são fornecidos:

MQZ_INIT_NAME
Inicializar o nome do componente de serviço.

MQZ_TERM_NAME
Finalizar o nome do componente de serviço.

MQZ_LOOKUP_NAME
Consulte o nome do gerenciador de filas para a fila especificada.

MQZ_INSERT_NAME
Insira uma entrada contendo o nome do gerenciador de filas proprietário da fila especificada no diretório usado pelo serviço.

MQZ_DELETE_NAME
Excluir a entrada para a fila especificada do diretório usado pelo serviço.

Se houver mais de um serviço de nomes configurado:

- Para consulta, a função MQZ_LOOKUP_NAME é chamada para cada serviço na lista até que o nome da fila seja resolvido (a menos que algum componente indique que a procura deve parar).
- Para inserir, a função MQZ_INSERT_NAME é chamada para o primeiro serviço na lista que suporta essa função.
- Para excluir, a função MQZ_DELETE_NAME é chamada para o primeiro serviço na lista que suporta essa função.

Não tenha mais de um componente que suporte as funções inserir e excluir. Entretanto, um componente que suporta somente consulta é viável e pode ser usado, por exemplo, como o último componente na lista para resolver qualquer nome que não seja conhecido por nenhum outro componente de serviço de nomes para um gerenciador de filas no qual o nome possa ser definido.

Na linguagem de programação C, os nomes são definidos como tipos de dados de função usando a instrução typedef. Eles podem ser usados para criar protótipo das funções de serviço, para assegurar que os parâmetros estejam corretos.

O arquivo de cabeçalho que contém todo o material específico para serviços instaláveis é `cmqzc.h` para a linguagem C.

Além da função de inicialização (MQZ_INIT_NAME), que deve ser o ponto de entrada principal do componente, funções são chamadas pelo endereço do ponto de entrada que a função de inicialização incluiu, usando a chamada MQZEP.

Usando diversos componentes de serviço
É possível instalar mais de um componente para um serviço. Isso permite que os componentes forneçam somente implementações parciais do serviço e dependam de outros componentes para fornecer as funções restantes.

Exemplo de uso de vários componentes

Suponha que você crie dois um componentes de serviços de nomes chamados `ABC_name_serv` e `XYZ_name_serv`.

`ABC_name_serv`
Esse componente suporta a inserção ou a exclusão de um nome do diretório de serviço, mas não suporta a procura de um nome de fila.

`XYZ_name_serv`
Esse componente suporta a procura de um nome de fila, mas não suporta a inserção ou a exclusão de um nome do diretório de serviço.

O componente `ABC_name_serv` retém um banco de dados de nomes de filas e usa dois algoritmos simples para inserir ou excluir um nome do diretório de serviço.

O componente `XYZ_name_serv` usa um algoritmo simples que retorna um nome de gerenciador de filas fixo para qualquer nome de fila com o qual ele é chamado. Ele não mantém um banco de dados de nomes de filas e, portanto, não suporta as funções de inserção e exclusão.

Os componentes são instalados no mesmo gerenciador de filas. As sub-rotinas *`ServiceComponent`* são ordenadas para que o componente `ABC_name_serv` seja chamado primeiro. Quaisquer chamadas para inserir ou excluir uma fila em um diretório de componentes são manipuladas pelo componente `ABC_name_serv`; ele é o único que implementa essas funções. No entanto, uma chamada de consulta que o componente `ABC_name_serv` não pode resolver é transmitida para o componente somente de consulta, `XYZ_name_serv`. Esse componente fornece um nome de gerenciador de filas por meio de seu algoritmo simples.

Omitir pontos de entrada ao usar vários componentes

Se decidir usar vários componentes para fornecer um serviço, será possível projetar um componente de serviço que não implementa determinadas funções. A estrutura de serviços instaláveis não coloca restrições sobre o que é possível omitir. No entanto, para serviços instaláveis específicos, omissão de um ou mais funções pode ser logicamente inconsistente com o propósito do serviço.

Exemplo de pontos de entrada usados com vários componentes

Tabela 139 na página 959 mostra um exemplo do serviço de nomes instalável para o qual os dois componentes foram instalados. Cada um suporta um conjunto diferente de funções associadas a esse determinado serviço instalável. Para a função de inserção, o ponto de entrada do componente ABC é chamado primeiro. Pontos de entrada que não foram definidos para o serviço (usando **MQZEP**) são

assumidos como NULL. Um ponto de entrada para inicialização é fornecido na tabela, mas isso não é necessário porque a inicialização é executada pelo ponto de entrada principal do componente.

Quando o gerenciador de filas precisar usar um serviço instalável, ele usa os pontos de entrada definidos para esse serviço (as colunas em Tabela 139 na página 959). Tomando cada componente em sua vez, o gerenciador de filas determina o endereço da rotina que implementa a função necessária. Em seguida, chama a rotina, se ela existir. Se a operação for bem-sucedida, quaisquer resultados e informações de status serão usados pelo gerenciador de filas.

Tabela 139. Exemplo de pontos de entrada para um serviço instalável

Número da função	**Componente de serviço de nomes ABC**	**Componente de serviço de nomes XYZ**
MQZID_INIT_NAME (Initialize)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (Terminate)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (Insert)	ABC_Insert()	NULL
MQZID_DELETE_NAME (Delete)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (Lookup)	NULL	XYZ_Lookup()

Se a rotina não existir, o gerenciador de filas repete esse processo para o próximo componente na lista. Além disso, se a rotina existir, mas retornar um código indicando que ela não pôde executar a operação, a tentativa continuará com o próximo componente disponível. Rotinas em componentes de serviço podem retornar um código que indica que não deve ser feita nenhuma tentativa adicional para executar a operação.

Configurando serviços e componentes

É possível configurar componentes de serviço usando os arquivos de configuração do gerenciador de filas, exceto em sistemas Windows, em que cada gerenciador de filas tem sua própria sub-rotina no Registro.

Procedimento

1. Inclua sub-rotinas no arquivo de configuração do gerenciador de filas, `qm.ini`, para definir o serviço para o gerenciador de filas e especificar o local do módulo:
   - Cada serviço usado deve ter uma sub-rotina `Service`, que define o serviço para o gerenciador de filas. Para obter mais informações, consulte Sub-rotina de serviço do arquivo qm.ini
   - Para cada componente dentro de um serviço, deve haver uma sub-rotina `ServiceComponent`. Essa sub-rotina identifica o nome e o caminho do módulo que contém o código para esse componente. Para obter mais informações, consulte a sub-rotina ServiceComponent do arquivo qm.ini

   O componente do serviço de autorização, conhecido como Gerenciador de autoridade de objeto (OAM), é fornecido com o produto. Ao criar um gerenciador de filas, o arquivo de configuração do gerenciador de filas (ou o Registro nos sistemas Windows) é automaticamente atualizado para incluir as sub-rotinas apropriadas para o serviço de autorização e para o componente padrão (o OAM). Para outros componentes, deve-se configurar o arquivo de configuração do gerenciador de filas manualmente.

   O código para cada componente de serviço é carregado no gerenciador de filas quando gerenciador de filas é iniciado, usando a ligação dinâmica, quando isso for suportado na plataforma.
2. Pare e reinicie o gerenciador de filas para ativar o componente.

Referências relacionadas

Sub-rotina Service do arquivo qm.ini

Sub-rotina ServiceComponent do arquivo qm.ini

Atualizando o OAM após mudar a autorização de um usuário
No IBM MQ, é possível atualizar a informação do grupo de autorização OAM imediatamente após mudar a associação ao grupo de autorização do usuário, refletindo as mudanças feitas no nível do sistema operacional sem precisar parar e reiniciar o gerenciador de filas. Para fazer isso, emita o comando **REFRESH SECURITY**.

Nota: Ao mudar autorizações com o comando `setmqaut`, o OAM implementa tais mudanças imediatamente.

Os gerenciadores de filas armazenam dados de autorização em uma fila local chamada SYSTEM.AUTH.DATA.QUEUE. Esses dados são gerenciados pelo **amqzfuma.exe**.

Referências relacionadas
REFRESH SECURITY

IBM i Serviços e componentes instaláveis no IBM i

Use estas informações para aprender sobre os serviços instaláveis e as funções e os componentes associados a eles. A interface para essas funções é documentada para que você ou os fornecedores de software possam fornecer os componentes.

As principais razões para fornecer serviços instaláveis do IBM MQ são:

- Fornecer a você a flexibilidade de escolher se deseja usar componentes fornecidos pelo IBM MQ for IBM i ou substituir ou aumentar os mesmos com outros.
- Permitir que fornecedores participem, fornecendo componentes que possam usar novas tecnologias, sem fazer mudanças internas no IBM MQ for IBM i.
- Permitir que o IBM MQ explore novas tecnologias de forma mais rápida e mais barata e, assim, fornecer produtos antes e a preços mais baixos.

Serviços instaláveis e *componentes de serviço* fazem parte da estrutura do produto IBM MQ. No centro desta estrutura está a parte do gerenciador de filas que implementa a função e as regras associadas ao Message Queue Interface (MQI). Essa parte central requer inúmeras funções de serviço, denominadas *serviços instaláveis*, para executar seu trabalho. O serviço instalável disponível no IBM MQ for IBM i é o serviço de autorização.

Cada serviço instalável é um conjunto relacionado de funções implementadas usando um ou mais *componentes de serviços*. Cada componente é chamado usando uma interface publicamente disponível e corretamente arquitetada. Isso permite que fornecedores de software independentes e outros terceiros forneçam componentes instaláveis para aumentar ou substituir os fornecidos pelo IBM MQ for IBM i. Tabela 140 na página 960 resume o suporte para o serviço de autorização.

Tabela 140. Resumo de componentes do serviço de autorização

Componente fornecido	Função	Requisitos
Gerenciador de autoridade de objeto (OAM)	Fornece verificação de autorização sobre os comandos e as chamadas MQI. Os usuários podem gravar seu próprio componente para aumentar ou substituir o OAM.	(Instalações de autorização de plataforma apropriadas são presumidas)
Componente de serviço de nomes do DCE **Nota:** O DCE é suportado somente nas versões do IBM MQ anteriores à versão V6.0.	• Permite que os gerenciadores de filas compartilhem as filas ou • Definido pelo usuário **Nota:** As filas compartilhadas devem ter seu atributo **Scope** configurado para CELL.	• O DCE é necessário para o componente fornecido ou • Um gerenciador de nomes gravado pelo usuário ou terceiros

IBM i *Funções e componentes no IBM i*

Use estas informações para entender as funções e os componentes, pontos de entrada, códigos de retorno e dados dos componentes que você pode usar no IBM MQ for IBM i.

Cada serviço consiste em um conjunto de funções relacionadas. Por exemplo, o serviço de nomes contém funções para:

- Consultar um nome de filas e retornar o nome do gerenciador de filas no qual a fila está definida
- Inserir um nome da fila no diretório de serviço
- Excluir um nome da fila do diretório do serviço

Ele também contém as funções de inicialização e finalização.

É fornecido um serviço instalável por um ou mais componentes de serviço. Cada componente pode executar algumas ou todas as funções que são definidas para esse serviço. O componente também é responsável por gerenciar quaisquer recursos subjacentes ou software de que precisa para implementar o serviço. Os arquivos de configuração fornecem uma maneira padrão de carregar o componente e de determinar os endereços das rotinas funcionais que fornece.

Os serviços e os componentes estão relacionados, como a seguir:

- Um serviço é definido para um gerenciador de filas por sub-rotinas em um arquivo de configuração.
- Cada serviço é suportado pelo código fornecido no gerenciador de filas. Os usuários não podem mudar esse código e, portanto, não podem criar seus próprios serviços.
- Cada serviço é implementado por um ou mais componentes; eles podem ser fornecidos com o produto ou gravados pelo usuário. Podem ser chamados diversos componentes para um serviço, cada um suportando diferentes recursos no serviço.
- Os pontos de entrada conectam os componentes de serviço ao código de suporte no gerenciador de filas.

Pontos de entrada

Cada componente de serviço é representado por uma lista de endereços de ponto de entrada das rotinas que suportam um determinado serviço instalável. O serviço instalável define a função a ser executada por cada rotina. A ordenação dos componentes de serviço quando eles são configurados define a ordem na qual os pontos de entrada são chamados em uma tentativa de atender a uma solicitação para o serviço. No arquivo de cabeçalho `cmqzc.h` fornecido, os pontos de entrada fornecidos para cada serviço possuem um prefixo MQZID_.

Códigos de retorno

Os componentes de serviço fornecem códigos de retorno para o gerenciador de filas para relatar uma variedade de condições. Eles relatam o sucesso ou falha da operação e indicam se o gerenciador de filas continuará para o próximo componente de serviço. Um parâmetro *Continuation* separado transporta essa indicação.

Dados do componente

Um único componente de serviço pode requerer que os dados sejam compartilhados entre as suas várias funções. Os serviços instaláveis fornecem uma área de dados opcional a ser transmitida em cada chamada de um determinado componente de serviço. Essa área de dados é para uso exclusivo do componente de serviço. Ela é compartilhada por todas as chamadas de uma determinada função, mesmo se forem feitas a partir de processos ou espaços de endereço diferentes. É garantido que isso seja endereçável a partir do componente de serviço sempre que for chamado. Deve-se declarar o tamanho dessa área na sub-rotina *`ServiceComponent`*.

IBM i *Inicialização no IBM i*

Quando a rotina de inicialização do componente é chamada, ela deve chamar a função MQZEP do gerenciador de filas para cada ponto de entrada suportado pelo componente. MQZEP define um ponto de entrada para o serviço. Todos os pontos de saída indefinidos são presumidos como NULL.

Inicialização primária
Um componente é sempre chamado com esta opção uma vez, antes de ser chamado de qualquer outra maneira.

Inicialização secundária
Um componente pode ser chamado com essa opção em determinadas plataformas. Por exemplo, ele pode ser chamado uma vez para cada processo de sistema operacional, encadeamento ou tarefa pelo qual o serviço é acessado.

Se a inicialização secundária for usada:

- O componente pode ser chamado mais de uma vez para a inicialização secundária. Para cada chamada desse tipo, uma chamada correspondente para a finalização secundária é emitida quando o serviço não é mais necessário.

  Para serviços de autorização, essa é a chamada MQZ_TERM_AUTHORITY.
- Os pontos de entrada devem ser especificados novamente (chamando MQZEP) toda vez que o componente for chamado para inicialização primária e secundária.
- Somente uma cópia de dados do componente é usada para o componente; não há uma cópia diferente para cada inicialização secundária.
- O componente não é chamado para qualquer outra chamada ao serviço (do processo de sistema operacional, encadeamento ou tarefa, conforme apropriado) antes da realização da inicialização secundária.
- O componente deve configurar o parâmetro **Version** para o mesmo valor para a inicialização primária e secundária.

Finalização primária
O componente é sempre iniciado com esta opção uma vez, quando ele não é mais necessário. Nenhuma chamada adicional será feita para esse componente.

Finalização secundária
O componente é iniciado com esta opção, se ele tiver sido iniciado para a inicialização secundária.

IBM i *Configurando serviços e componentes no IBM i*

Configure componentes de serviço usando os arquivos de configuração do gerenciador de filas.

Procedimento

1. Inclua sub-rotinas no arquivo de configuração do gerenciador de filas, `qm.ini`, para definir o serviço para o gerenciador de filas e especificar o local do módulo:

   - Cada serviço usado deve ter uma sub-rotina `Service`, que define o serviço para o gerenciador de filas. Para obter mais informações, consulte Sub-rotina de serviço do arquivo qm.ini
   - Para cada componente dentro de um serviço, deve haver uma sub-rotina `ServiceComponent`. Essa sub-rotina identifica o nome e o caminho do módulo que contém o código para esse componente. Para obter mais informações, consulte a sub-rotina ServiceComponent do arquivo qm.ini

   O componente do serviço de autorização, conhecido como Gerenciador de autoridade de objeto (OAM), é fornecido com o produto. Ao criar um gerenciador de filas, o arquivo de configuração do gerenciador de filas é automaticamente atualizado para incluir as sub-rotinas apropriadas para o serviço de autorização e para o componente padrão (o OAM). Para outros componentes, deve-se configurar o arquivo de configuração do gerenciador de filas manualmente.

   O código para cada componente de serviço é carregado no gerenciador de filas quando gerenciador de filas é iniciado, usando a ligação dinâmica, quando isso for suportado na plataforma.

2.

IBM i *Criando seu próprio componente de serviço no IBM i*

Use estas informações para saber como criar um componente de serviço no IBM MQ for IBM i.

Para criar seu próprio componente de serviço:

- Assegure que o arquivo de cabeçalho `cmqzc.h` esteja incluído em seu programa.
- Crie a biblioteca compartilhada compilando o programa e vinculando-o às bibliotecas compartilhadas `libmqm*` e `libmqmzf*`.

  Nota: Como o agente pode ser executado em um ambiente encadeado, deve-se construir o OAM para ser executado em um ambiente encadeado. Isso inclui usar as versões encadeadas de `libmqm` e `libmqmzf`.
- Inclua sub-rotinas no arquivo de configuração do gerenciador de filas para definir o serviço para o gerenciador de filas e para especificar o local do módulo.
- Pare e reinicie o gerenciador de filas para ativar o componente.

IBM i *Serviço de autorização no IBM i*

O serviço de autorização é um serviço instalável que permite que os gerenciadores de filas chamem instalações de autorização, por exemplo, verificando se uma ID do usuário tem autoridade para abrir uma fila.

Esse serviço é um componente do security enabling interface (SEI) do IBM MQ, que faz parte da estrutura do IBM MQ. Os seguintes assuntos são discutidos:

- "Gerenciador de autoridade de objeto (OAM)" na página 963
- "Definindo o serviço para o sistema operacional" na página 964
- "Configurando as sub-rotinas do serviço de autorização" na página 964
- "Interface de serviço de autorização no IBM i" na página 964

Gerenciador de autoridade de objeto (OAM)

O componente de serviço de autorização fornecido com os produtos IBM MQ é chamado de Gerenciador de Autoridade de Objeto (OAM). Por padrão, o OAM fica ativo e funciona com os seguintes comandos de controle:

- **WRKMQMAUT** - trabalhar com autoridade
- **WRKMQMAUTD** - trabalhar com dados de autoridade
- **DSPMQMAUT** - exibir a autoridade do objeto
- **GRTMQMAUT** - conceder autoridade do objeto
- **RVKMQMAUT** - revogar a autoridade do objeto
- **RFRMQMAUT** - atualizar segurança

A sintaxe destes comandos e o modo de usá-los são descritos na ajuda do comando de CL. O OAM trabalha com a *entidade* de um diretor ou grupo.

Quando uma solicitação de MQI é feita ou um comando é emitido, o OAM verifica a autorização da entidade associada à operação para ver se ele pode executar as seguintes ações:

- Executar a operação solicitada.
- Acessar os recursos do gerenciador de filas especificado.

O serviço de autorização permite aumentar ou substituir a verificação de autoridade fornecida para gerenciadores de filas, gravando seu próprio componente de serviço de autorização.

Definindo o serviço para o sistema operacional

As sub-rotinas do serviço de autorização no arquivo de configuração do gerenciador de filas `qm.ini` definem o serviço de autorização para o gerenciador de filas. Consulte “Configurando serviços e componentes no IBM i” na página 962 para obter informações sobre os tipos de sub-rotina.

Configurando as sub-rotinas do serviço de autorização

No IBM MQ for IBM i:

Diretor
É um perfil de usuário do sistema IBM i.

Group
É um perfil de grupo do sistema IBM i.

Autorizações podem ser concedidas ou revogadas apenas no nível do grupo. Um pedido para conceder ou revogar a autoridade de um usuário atualiza o grupo primário para esse usuário.

Cada gerenciador de filas possui seu próprio arquivo de configuração do gerenciador de filas. Por exemplo, o caminho e o nome de arquivo padrão do arquivo de configuração do gerenciador de filas para o gerenciador de filas QMNAME é `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

As sub-rotinas *`Service`* e *`ServiceComponent`* para o componente de autorização padrão são incluídas no `qm.ini` automaticamente, mas podem ser substituídas por `WRKENVVAR`. Qualquer outra sub-rotina *`ServiceComponent`* deve ser incluída manualmente.

Por exemplo, as seguintes sub-rotinas no arquivo de configuração do gerenciador de filas definem dois componentes de serviço de autorização:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

Figura 100. Sub-rotinas do serviço de autorização em qm.ini no IBM i

A primeira sub-rotina do componente de serviço `MQ.UNIX.authorization.service` define o componente de serviço de autorização padrão, o OAM. Se essa sub-rotina for removida e o gerenciador de filas reiniciado, o OAM será desativado e nenhuma verificação de autorização será realizada.

IBM i *Interface de serviço de autorização no IBM i*

A interface de serviço de autorização fornece vários pontos de entrada para uso pelo gerenciador de filas.

MQZ_AUTHENTICATE_USER
Autentica um ID do usuário e senha e pode configurar campos de contexto de identidade.

MQZ_CHECK_AUTHORITY
Verifica se uma entidade possui autoridade para desempenhar uma ou mais operações em um objeto especificado.

MQZ_COPY_ALL_AUTHORITY
Copia todas as autorizações atuais que existem para um objeto referenciado para outro objeto.

MQZ_DELETE_AUTHORITY
Exclui todas as autorizações associadas a um objeto especificado.

MQZ_ENUMERATE_AUTHORITY_DATA
Recupera todos os dados de autoridade que correspondem aos critérios de seleção especificados.

MQZ_FREE_USER
Libera recursos alocados associados.

MQZ_GET_AUTHORITY
Obtém a autoridade que uma entidade tem para acessar um objeto especificado.

MQZ_GET_EXPLICIT_AUTHORITY
Obtém a autoridade que um grupo denominado possui para acessar um objeto especificado (mas sem a autoridade adicional do grupo **nobody**) ou a autoridade que o grupo principal do proprietário nomeado possui para acessar um objeto especificado.

MQZ_INIT_AUTHORITY
Inicializa componente de serviço de autorização.

MQZ_INQUIRE
Consulta a funcionalidade suportada do serviço de autorização.

MQZ_REFRESH_CACHE
Atualizar todas as autorizações.

MQZ_SET_AUTHORITY
Define a autoridade que uma entidade tem para um objeto especificado.

MQZ_TERM_AUTHORITY
Finaliza o componente de serviço de autorização.

Esses pontos de entrada suportam o uso do Windows Security Identifier (NT SID).

Estes nomes são definidos como **typedef**, no arquivo de cabeçalho `cmqzc.h`, que pode ser usado para o protótipo das funções do componente.

A função de inicialização (**MQZ_INIT_AUTHORITY**) deve ser o ponto de entrada principal para o componente. As outras funções são chamadas por meio do endereço do ponto de entrada que a função de inicialização incluiu no vetor do ponto de entrada do componente.

Consulte “Criando seu próprio componente de serviço no IBM i” na página 963 para obter mais informações.

Multi Gravando e compilando saídas de API em Multiplataformas

As saídas de API permitem gravar o código que altera o comportamento das chamadas de API do IBM MQ, como MQPUT e MQGET, e, em seguida, inserem esse código imediatamente antes ou depois dessas chamadas.

Nota: z/OS Não suportado em IBM MQ for z/OS.

Por que usar saídas de API?

Cada um de seus aplicativos tem uma tarefa específica a executar e seu código deve executar essa tarefa com a maior eficiência possível. Em um nível superior, talvez você queira aplicar padrões ou processos de negócios a um determinado gerenciador de filas para todos os aplicativos que usam esse gerenciador de filas. É mais eficiente fazer isso acima do nível de aplicativos individuais e, assim, não precisar mudar o código de cada aplicativo afetado.

Eis algumas sugestões de áreas nas quais as saídas de API podem ser úteis:

Segurança
Para segurança, é possível fornecer autenticação, verificando se os aplicativos estão autorizados a acessar uma fila ou gerenciador de filas. Também é possível policiar o uso de aplicativos da API, autenticando as chamadas de API individuais ou mesmo os parâmetros que usam.

Flexibilidade
Para flexibilidade, é possível responder às rápidas mudanças no seu ambiente de negócios sem mudar os aplicativos que dependem dos dados nesse ambiente. Você poderia, por exemplo, ter

saídas de API que respondem a mudanças nas taxas de juros, taxas de câmbio de moedas ou preço dos componentes em um ambiente de fabricação.

Monitorando o uso de uma fila ou de um gerenciador de filas
Para monitorar o uso de uma fila ou de um gerenciador de filas, é possível rastrear o fluxo de aplicativos e mensagens, registrar os erros nas chamadas de API, configurar as trilhas de auditoria para propósitos contábeis ou coletar estatísticas de uso para propósitos de planejamento.

O que acontece quando uma saída de API é executada?

Após escrever um programa de saída e identificá-lo para o IBM MQ, o gerenciador de filas chama seu código de saída automaticamente nos pontos registrados.

As rotinas de saída da API para execução são identificadas em sub-rotinas no Multiplatforms. Este tópico abrange as sub-rotinas nos arquivos de configuração `mqs.ini` e `qm.ini`.

A definição das rotinas pode ocorrer em três locais:

1. ApiExitCommon, no arquivo mqs.ini, identifica as rotinas, para todo o IBM MQ, aplicadas quando gerenciadores de filas são iniciados. Elas podem ser substituídas por rotinas definidas para gerenciadores de filas individuais (consulte o item “3” na página 966 nesta lista).
2. O modelo ApiExit, no arquivo mqs.ini , identifica rotinas, para todo o IBM MQ, copiadas para o conjunto local ApiExit(consulte o item “3” na página 966 nesta lista) quando um novo gerenciador de filas é criado..
3. ApiExitLocal, no arquivo qm.ini, identifica as rotinas que se aplicam a um gerenciador de filas específico.

Quando um novo gerenciador de filas é criado, as definições ApiExitTemplate no mqs.ini são copiadas para as definições ApiExitLocal em qm.ini para o novo gerenciador de filas. Quando um gerenciador de filas é iniciado, as definições ApiExitCommon e ApiExitLocal são usadas. As definições ApiExitLocal substituem as definições ApiExitCommon se ambas identificarem uma rotina com o mesmo nome. O atributo `Sequence` descrito em “Configurando saídas de API” na página 971 determina a ordem na qual as rotinas definidas nas sub-rotinas são executadas.

Usando saídas de API entre diversas instalações do IBM MQ

Certifique-se de que as saídas de API escritas para a versão anterior do IBM MQ sejam usadas para funcionar com todas as versões, pois mudanças feitas em saídas na IBM WebSphere MQ 7.1 podem não funcionar com uma versão anterior. Para obter mais informações sobre as mudanças feitas nas saídas, consulte “Composição de saídas e serviços instaláveis em AIX, Linux, and Windows” na página 948

As amostras fornecidas para as saídas de API amqsaem e amqsaxe refletem as mudanças necessárias ao gravar saídas. O aplicativo cliente deve assegurar que as bibliotecas corretas do IBM MQ que correspondem à instalação do gerenciador de filas ao qual o aplicativo está associado estejam vinculadas a ele antes do lançamento do aplicativo.

Multi *Escrevendo saídas de API*

É possível escrever saídas para cada chamada de API usando a linguagem de programação C.

Saídas disponíveis

Saídas estão disponíveis para cada chamada de API da seguinte forma:

- MQCB, para registrar novamente um retorno de chamada para a manipulação de objetos especificada e controlar ativação e mudanças no retorno de chamada
- MQCTL, para executar ações de controle nos identificadores de objeto abertos para uma conexão
- MQCONN/MQCONNX, para fornecer uma manipulação de conexões do gerenciador de filas para uso em chamadas de API subsequentes
- MQDISC, para desconectar-se de um gerenciador de filas

- MQBEGIN, para iniciar uma unidade de trabalho (UOW) global
- MQBACK, para restaurar uma UOW
- MQCMIT, para confirmar uma UOW
- MQOPEN, para abrir um recurso do IBM MQ para acesso subsequente
- MQCLOSE, para fechar um recurso do IBM MQ que havia sido aberto anteriormente para acesso
- MQGET, para recuperar uma mensagem de uma fila que havia sido aberta anteriormente para acesso
- MQPUT1, para colocar uma mensagem em uma fila
- MQPUT, para colocar uma mensagem em uma fila que havia sido aberta anteriormente para acesso
- MQINQ, para consultar sobre os atributos de um recurso do IBM MQ que havia sido aberto anteriormente para acesso
- MQSET, para configurar os atributos de uma fila que havia sido aberta anteriormente para acesso
- MQSTAT, para recuperar informações de status
- MQSUB, para registrar a assinatura de aplicativos para um determinado tópico
- MQSUBRQ, para fazer uma solicitação de uma assinatura

MQ_CALLBACK_EXIT fornece uma função de saída a ser executada antes e após o processamento de retorno de chamada. Para obter mais informações, veja Retorno de chamada - MQ_CALLBACK_EXIT.

Escrevendo saídas de API

Nas saídas de API, as chamadas assumem o formato geral:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

em que *call* é o nome da chamada MQI sem o prefixo MQ; por exemplo, PUT, GET. Os *parameters* controlam a função da saída, fornecendo primariamente a comunicação entre a saída e os blocos de controle externos MQAXP (a estrutura do parâmetro de saída de API) e MQAXC (estrutura de contexto de saída de API). *context* descreve o contexto no qual a saída de API foi chamada e *ApiCallParameters* representam os parâmetros para a chamada MQI.

Para ajudar a escrever sua saída de API, uma saída de amostra, amqsaxe0.c, é fornecida; essa saída gera as entradas para arquivo especificado. É possível usar essa amostra como seu ponto de início ao escrever saídas. Para obter mais informações sobre como usar a saída de amostra, consulte “O programa de amostra de saída de API” na página 1089.

Para obter mais informações sobre as chamadas de saída de API, blocos de controle externo e tópicos associados, consulte Referência de saída de API.

Para obter informações gerais sobre como escrever, compilar e configurar uma saída, consulte “Composição de saídas e serviços instaláveis em AIX, Linux, and Windows” na página 948.

Usando identificadores de mensagens em saídas de API

É possível controlar a quais propriedades de mensagens uma saída de API tem acesso. Propriedades estão associadas a um ExitMsgHandle. Propriedades configuradas em uma saída put são configuradas na mensagem que está sendo efetuado put, mas as propriedades recuperadas em uma saída get não são retornadas ao aplicativo.

Ao registrar uma função de saída MQ_INIT_EXIT usando a chamada MQI MQXEP com **Function** configurado para MQXF_INIT e **ExitReason** configurado como MQXR_CONNECTION, você passa uma estrutura MQXEPO como o parâmetro **ExitOpts**. A estrutura MQXEPO contém o campo ExitProperties, que especifica o conjunto de propriedades a ser disponibilizado para a saída. Ela é especificada como uma sequência de caracteres que representa o prefixo das propriedades, que corresponde a um nome de pasta MQRFH2.

Cada saída de API recebe uma estrutura MQAXP, que contém um campo ExitMsgHandle. Esse campo é configurado para um valor gerado pelo IBM MQ e é específico de uma conexão. Portanto, o identificador permanece inalterado entre as saídas de API de tipos iguais ou diferentes na mesma conexão.

Em um MQ_PUT_EXIT ou MQ_PUT1_EXIT com um **ExitReason** de MQXR_BEFORE, ou seja, uma saída de API executada antes de efetuar put de uma mensagem, quaisquer propriedades (diferentes de propriedades do descritor de mensagens) associadas a ExitMsgHandle quando a saída for concluída serão configuradas na mensagem para a qual put está sendo efetuado. Para evitar que isso aconteça, configure ExitMsgHandle para MQHM_NONE. Também é possível fornecer um identificador de mensagem diferente.

Em um MQ_GET_EXIT e MQ_CALLBACK_EXIT, o identificador ExitMsgé limpo das propriedades e preenchido com as propriedades especificadas no campo ExitProperties quando o MQ_INIT_EXIT foi registrado, diferente das propriedades do descritor de mensagem Essas propriedades não são disponibilizadas no aplicativo de get. Se o aplicativo de get tiver especificado um identificador de mensagens no campo MQGMO (opções de get message), quaisquer propriedades associadas a esse identificador, incluindo as propriedades do descritor de mensagens, estarão disponíveis para a saída de API. Para evitar que o ExitMsgHandle seja preenchido com as propriedades, configure-o para MQHM_NONE.

Um programa de amostra, amqsaem0.c, é fornecido para ilustrar o uso de identificadores de mensagens em saídas de API.

Referências relacionadas

Saídas de usuário, saídas de API e referência de serviços instaláveis

Multi *Compilando saídas de API*

Após ter escrito uma saída, você a compila e vincula da seguinte forma.

Os exemplos a seguir mostram os comandos usados para o programa de amostra descrito em "O programa de amostra de saída de API" na página 1089. Para plataformas diferentes de sistemas Windows, é possível encontrar o código de saída da API de amostra em *MQ_INSTALLATION_PATH*/samp e a biblioteca compartilhada compilada e vinculada em *MQ_INSTALLATION_PATH*/samp/bin.

Windows Para sistemas Windows, é possível localizar o código de saída da API de amostra em *MQ_INSTALLATION_PATH* \Tools\c\Samples. *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ foi instalado.

Nota: A orientação sobre programação de aplicativos de 64 bits está listada em Normas de codificação em plataformas de 64 bits.

Para clientes Multicast, as saídas de API e as saídas de conversão de dados precisam ser capazes de executar no lado do cliente, já que algumas mensagens podem não passar pelo gerenciador de filas. As bibliotecas a seguir fazem parte dos pacotes do cliente, assim como dos pacotes do servidor:

Tabela 141. Bibliotecas que estão nos pacotes do cliente e do servidor

Sistema operacional	Bibliotecas
AIX AIX	32 bits e 64 bits: libmqm.a e libmqm_r.a
IBM i IBM i	LIBMQM & LIBMQM_R
Linux Linux	32 bits e 64 bits: libmqm.s e libmqm_r.so
Windows Windows	32 bits e 64 bits: mqm.dll e mqm.pdb

Linux AIX *Compilando saídas de API em sistemas AIX and Linux*

Exemplos de como compilar Saídas de API em sistemas AIX and Linux.

Em todas as plataformas, o ponto de entrada para o módulo é MQStart.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

no AIX

AIX

Compile o código-fonte de saída de API emitindo um dos seguintes comandos:

Aplicativos de 32 bits

Não encadeado

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Encadeado

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Aplicativos de 64 bits

Não encadeado

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Encadeado

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

no Linux

Linux

Compile o código-fonte de saída de API emitindo um dos seguintes comandos:

Aplicativos de 31 bits

Não encadeado

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Encadeado

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Aplicativos de 32 bits

Não encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Aplicativos de 64 bits

Não encadeado

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Encadeado

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *Compilando saídas de API em sistemas Windows*

Compile e vincule o programa de saída da API de amostra, `amqsaxe0.c`, no Windows

Um arquivo de manifesto é um documento XML opcional que contém a informação de versão, ou qualquer outra, que possa ser integrada a um aplicativo compilado ou DLL.

Se você não tiver esse documento, omita o parâmetro -manifest *manifest.file* no comando **mt**.

Adapte os comandos nos exemplos em Figura 101 na página 970 ou Figura 102 na página 970 para compilar e vincular o `amqsaxe0.c` no Windows. Os comandos funcionam com o Microsoft Visual Studio 2008, 2010 ou 2012. Os exemplos supõem que o diretório `C:\Program Files\IBM\MQ\tools\c\samples` é o diretório atual.

32 bits

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

Figura 101. Compile e vincule o `amqsaxe0.c` no Windows de 32 bits

64 bits

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

Figura 102. Compile e vincule o `amqsaxe0.c` no Windows de 64 bits

Conceitos relacionados

“O programa de amostra de saída de API” na página 1089

A saída API de amostra gera um rastreio MQI para um arquivo especificado pelo usuário com um prefixo definido na variável de ambiente **MQAPI_TRACE_LOGFILE** .

IBM i *Compilando saídas de API no IBM i*
Compilando saídas de API no IBM i.

Uma saída é criada da seguinte forma (para um exemplo de linguagem C):

1. Crie um módulo usando CRTCMOD. Compile-o para usar teraspace, incluindo o parâmetro TERASPACE(*YES *TSIFC).
2. Crie um programa de serviços do módulo usando CRTSRVPGM. Deve-se ligá-lo ao programa de serviços QMQM/LIBMQMZF_R para saídas de API multiencadeadas.

Configurando saídas de API

Configure IBM MQ para ativar as saídas de API mudando as informações de configuração.

Para mudar as informações de configuração, deve-se mudar as sub-rotinas que definem as rotinas de saída e a sequência na qual elas são executadas. Essas informações podem ser mudadas das seguintes maneiras:

- Windows Linux Usando o IBM MQ Explorer no Windows e no Linux (plataformas x86 e x86-64).
- Windows Usando o comando **amqmdain** no Windows.
- Multi Usando os arquivos mqs.ini e qm.ini diretamente no Multiplataformas.

  O arquivo mqs.ini contém informações relevantes para todos os gerenciadores de filas em um determinado nó. É possível localizá-lo nos locais a seguir:

  – Linux AIX No diretório /var/mqm no AIX and Linux.

  – Windows No WorkPath especificado na chave HKLM\SOFTWARE\IBM\WebSphere MQ nos sistemas Windows ...

  – IBM i No diretório /QIBM/UserData/mqm no IBM i.

  O arquivo qm.ini contém informações relevantes para um gerenciador de filas específico. Há um arquivo de configuração do gerenciador de filas para cada gerenciador de filas, retido na raiz da árvore de diretórios ocupada pelo gerenciador de filas. Por exemplo, o caminho e o nome para um arquivo de configuração para um gerenciador de filas denominado QMNAME é:

  IBM i Nos sistemas IBM i:

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  Linux AIX Nos sistemas AIX and Linux:

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows Nos sistemas Windows:

  ```
  C:\ProgramData \IBM \MQ\qmgrs\QMNAME\qm.ini
  ```

  Antes de editar um arquivo de configuração, faça backup dele para que tenha uma cópia para a qual possa reverter se surgir a necessidade.

  É possível editar arquivos de configuração de uma das maneiras a seguir:

  – Automaticamente, usando os comandos que mudam a configuração dos gerenciadores de fila no nó.

– Manualmente, usando um editor de texto padrão.

Se você configurar um valor incorreto em um atributo do arquivo de configuração, o valor será ignorado e uma mensagem do operador será emitida para indicar o problema. O efeito é o mesmo que perder o atributo inteiramente.

Sub-rotinas para configuração

As sub-rotinas que devem ser mudadas são as seguintes:

ApiExitCommon

Definido em `mqs.ini` e no IBM MQ Explorer na página de propriedades IBM MQ, sob Saídas.

Quando qualquer gerenciador de filas se inicia, os atributos nesta sub-rotina são lidos e, em seguida, substituídos pelas saídas da API definidas em `qm.ini`.

ApiExitTemplate

Definido em `mqs.ini` e no IBM MQ Explorer na página de propriedades IBM MQ, sob Saídas.

Quando algum gerenciador de filas for criado, os atributos nesta sub-rotina serão copiados no arquivo qm.ini recém-criado na sub-rotina ApiExitLocal.

ApiExitLocal

Definido em `qm.ini` e no IBM MQ Explorer na página de propriedades do gerenciador de filas, sob Saídas.

Quando o gerenciador de filas se inicia, as saídas da API definidas aqui substituem os padrões definidos em `mqs.ini`.

Atributos para as sub-rotinas

- Nomeie a saída de API usando o seguinte atributo:

  Name=ApiExit_name
  O nome descritivo da saída de API passada para ela no campo ExitInfoName da estrutura MQAXP.

  Este nome deve ser exclusivo, sem ultrapassar 48 caracteres, e conter apenas caracteres válidos para os nomes de objetos do IBM MQ (por exemplo, nomes de fila).

- Identifique o módulo e o ponto de entrada do código de saída de API para execução usando os seguintes atributos:

  Function=function_name
  O nome do ponto de entrada da função no módulo que contém o código de saída de API. Este ponto de entrada é a função MQ_INIT_EXIT.

  O comprimento deste campo está limitado a MQ_EXIT_NAME_LENGTH.

  Module=module_name
  O módulo que contém o código de saída de API.

  Se esse campo contiver o nome de caminho completo do módulo, ele será utilizado dessa forma.

  Se este campo contiver apenas o nome do módulo, o módulo será localizado usando o atributo `ExitsDefaultPath` no `ExitPath` em qm.ini.

  Em plataformas que suportam bibliotecas encadeadas separadas, deve-se fornecer uma versão encadeada e não encadeada do módulo de saída de API. A versão encadeada deve ter um sufixo `_r`. A versão encadeada do stub de aplicativo IBM MQ anexa `_r` implicitamente ao nome do módulo fornecido antes de ser carregada.

  O comprimento deste campo é limitado ao comprimento máximo do caminho que a plataforma suporta.

- Opcionalmente, passe os dados com a saída que usa o seguinte atributo:

Data=data_name
Dados a serem passados para a saída de API no campo ExitData da estrutura MQAXP.

Se você incluir este atributo, espaços em branco iniciais e finais serão removidos, a sequência restante será truncada para 32 caracteres e o resultado será passado para a saída. Se você omitir este atributo, o valor padrão de 32 espaços em branco é passado para a saída.

O comprimento máximo deste campo é de 32 caracteres.

- Identifique a sequência desta saída em relação a outras saídas usando o seguinte atributo:

Sequence=sequence_number
A sequência na qual esta saída de API é chamada em relação a outras saídas de API. Uma saída com um baixo número de sequência é chamada antes de uma saída com um número de sequência mais alto. Não há necessidade para que a numeração de sequência de saídas seja contígua. Uma sequência de 1, 2, 3 possui o mesmo resultado que uma sequência de 7, 42, 1096. Se duas saídas tiverem o mesmo número de sequência, o gerenciador de filas decidirá qual chamar primeiro. É possível informar qual foi chamado após o evento colocando a hora ou um marcador no ExitChainArea indicado pelo ExitChainAreaPtr em MQAXP ou gravando seu próprio arquivo de log.

Este atributo é um valor numérico não assinado.

Sub-rotinas de amostra

O arquivo mqs.ini de amostra contém as seguintes sub-rotinas:

ApiExitTemplate
Esta sub-rotina define uma saída com o nome descritivo `OurPayrollQueueAuditor`, nome do módulo `auditor` e sequência número 2. Um valor de dados de 123 é transmitido para a saída.

ApiExitCommon
Esta sub-rotina define uma saída com o nome descritivo `MQPoliceman`, nome do módulo `tmqp` e sequência número 1. Os dados transmitidos são uma instrução (`CheckEverything`).

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

O seguinte arquivo qm.ini de amostra contém uma definição ApiExitLocal de uma saída com o nome descritivo `ClientApplicationAPIchecker`, nome do módulo `ClientAppChecker` e número de sequência 3.

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

Programas de Saída de Canal para Canais de Mensagens

Essa coleção de tópicos contém informações sobre os programas de saída do canal IBM MQ para canais do sistema de mensagens.

Message channel agents (MCAs) também podem chamar saídas de conversão de dados. Para obter mais informações sobre saídas de conversão de dados, veja “Escrevendo saídas de conversão de dados” na página 995.

Algumas dessas informações também se aplicam a saídas em canais MQI, que conectam IBM MQ MQI clients a gerenciadores de filas. Para obter mais informações, veja Programas de saída de canal para canais MQI.

Os programas de saída do canal são chamados nos locais definidos no processamento executado pelos programas MCA.

Alguns desses programas de saída do usuário trabalharam em pares complementares. Por exemplo, se um programa de saída de usuário for chamado pelo MCA de envio para criptografar as mensagens para transmissão, o processo complementar deve estar funcionando na extremidade de recebimento para reverter o processo.

O Tabela 142 na página 974 mostra os tipos de saída do canal que ficam disponíveis para cada tipo de canal.

Tabela 142. Saídas de Canal Disponíveis para cada tipo de canal

Tipo de Canal	**Saída de mensagen**	**saída de nova tentativa de mensagem**	**Saída de recepção**	**Saída de segurança**	**Saída de envio**	**saída de definição automática**
Canal Emissor	Sim		Sim	Sim	Sim	
Canal servidor	Sim		Sim	Sim	Sim	
canal do emissor de clusters	Sim		Sim	Sim	Sim	Sim
Canal receptor	Sim	Sim	Sim	Sim	Sim	Sim
Canal solicitador	Sim	Sim	Sim	Sim	Sim	
canal do receptor de clusters	Sim	Sim	Sim	Sim	Sim	Sim
canal de conexão do cliente			Sim	Sim	Sim	
canal de conexão do servidor			Sim	Sim	Sim	Sim

Notas: z/OS

1. No z/OS, a saída de definição automática se aplica apenas a canais de emissor de cluster e receptor de cluster.

Se você pretende executar as saídas de canal em um cliente, não será possível usar a variável de ambiente MQSERVER. Em vez disso, crie e faça referência a uma Client Channel Definition Table (CCDT) conforme descrito em Client Channel Definition Table.

Visão geral de processamento

Uma visão geral de como os MCAs usam programas de saída do canal.

Na inicialização, os MCAs trocam um diálogo de inicialização para sincronizar o processamento. Em seguida, alternam para uma troca de dados que inclui as saídas de segurança. Essas saídas devem terminar com sucesso para que a fase de inicialização seja concluída e para permitir que as mensagens sejam transferidas.

A fase de verificação de segurança é um loop, conforme mostrado em Figura 103 na página 975.

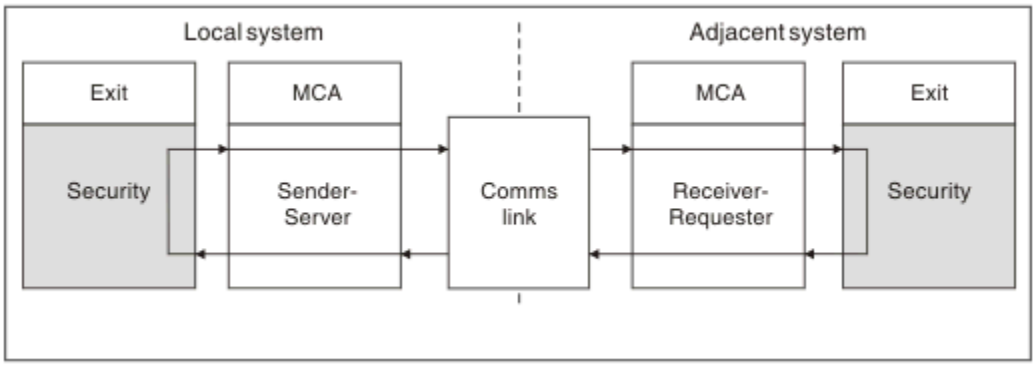


Figura 103. Loop da saída de segurança

Durante a fase de transferência de mensagem, o MCA de envio recebe as mensagens de uma fila de transmissão, chama a saída de mensagem, chama a saída de envio e, em seguida, envia a mensagem para o MCA de recebimento, conforme mostrado em Figura 104 na página 975.

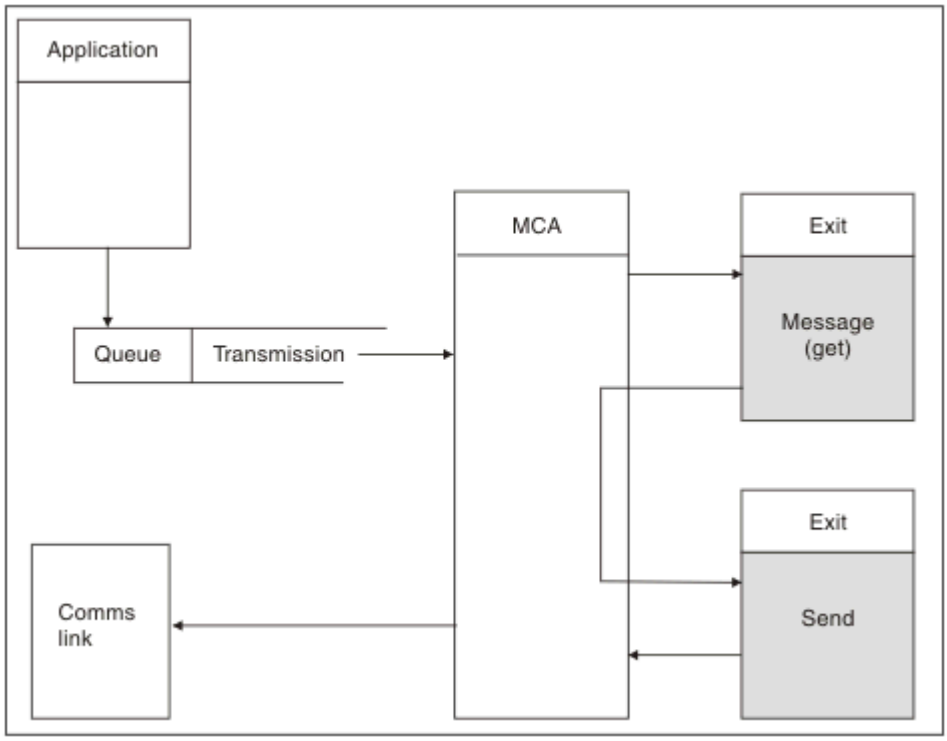


Figura 104. Exemplo de uma saída de envio na extremidade do emissor do canal de mensagens

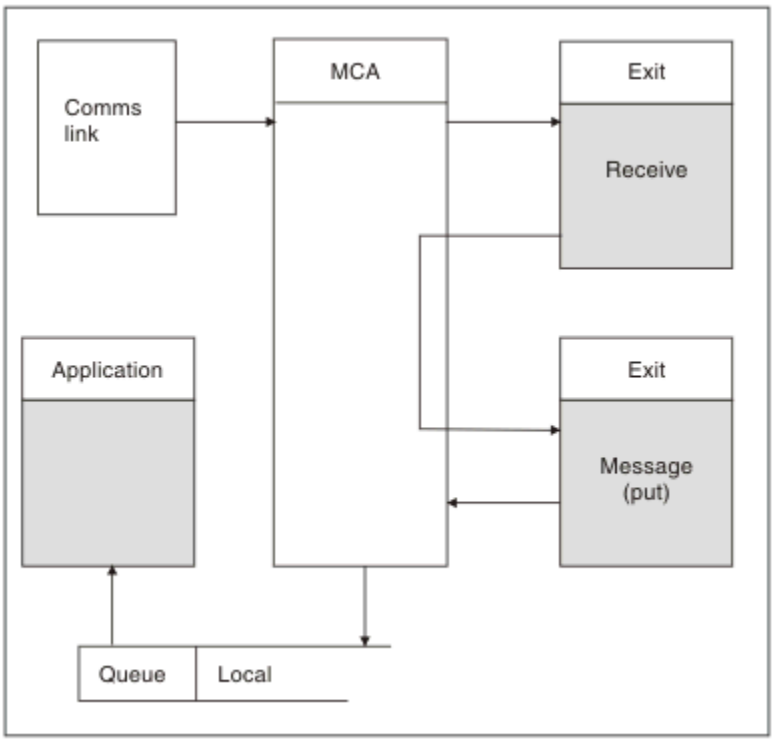


Figura 105. Exemplo de uma saída de recebimento no receptor final do canal de mensagens

O MCA de recebimento recebe uma mensagem do link de comunicações, chama a saída de recebimento, chama a saída de mensagem e, em seguida, coloca a mensagem na fila local, conforme mostrado em Figura 105 na página 976. (A saída de recebimento pode ser chamada mais de uma vez antes da saída de mensagem ser chamada.)

Gravando programas de saída do canal

É possível usar as seguintes informações para ajudá-lo a gravar programas de saída do canal.

As saídas de usuário e programas de saída do canal podem usar todas as chamadas MQI, exceto conforme o observado nas seções a seguir. Para MQ V7 e posterior, a estrutura MQCXP versão 7 e superior contém a manipulação de conexão hConn, que pode ser usada em vez de emitir o MQCONN. Para versões anteriores, para obter a manipulação de conexões, uma chamada MQCONN deverá ser emitida, embora um aviso MQRC_ALREADY_CONNECTED seja retornado porque o próprio canal está conectado ao gerenciador de filas.

Observe que a saída do canal deve ser thread-safe.

Para saídas nos canais de conexão do cliente, o gerenciador de filas ao qual a saída tenta se conectar depende de como a saída foi vinculada. Se a saída foi vinculada a MQM.LIB (ou QMQM/LIBMQM em IBM i) e não for especificado um nome do gerenciador de filas na chamada MQCONN, a saída tentará se conectar ao gerenciador de filas padrão do seu sistema. Se a saída foi vinculada com MQM.LIB (ou QMQM/LIBMQM no IBM i) e você especificar o nome do gerenciador de filas que foi transmitido para a saída por meio do campo QMgrName do MQCD, a saída tentará se conectar a esse gerenciador de filas. Se a saída tiver sido vinculada com MQIC.LIB ou qualquer outra biblioteca, a chamada MQCONN falhará especificando um nome do gerenciador de filas ou não.

É necessário evitar alterar o estado da transação associada ao hConn passado em uma saída de canal; não se deve usar os verbos MQCMIT, MQBACK ou MQDISC com o canal hConn e não é possível usar o verbo MQBEGIN especificando o canal hConn.

Se MQCONNX for usado especificando MQCNO_HANDLE_SHARE_BLOCK ou MQCNO_HANDLE_SHARE_NO_BLOCK para criar uma nova conexão do IBM MQ, então será sua responsabilidade assegurar que a conexão seja corretamente gerenciada e desconectar do gerenciador

de filas corretamente. Por exemplo, uma saída de canal que cria uma nova conexão ao gerenciador de filas em cada chamada sem desconectar resulta em um acúmulo de manipulações de conexão e um aumento no número de encadeamentos de agente.

Uma saída é executada no mesmo encadeamento que o MCA sozinho e usa a mesma manipulação de conexões. Portanto, ela é executada dentro do mesmo UOW que o MCA e quaisquer chamadas feitas no ponto de sincronização são confirmadas ou voltadas pelo canal ao final do lote.

Portanto, uma saída de mensagem de canal poderia enviar mensagens de notificação que são confirmadas apenas nessa fila quando o lote que contém a mensagem original for confirmado. Portanto, é possível emitir as chamadas MQI do ponto de sincronização a partir de uma saída de mensagem de canal.

Uma saída do canal pode mudar os campos no MQCD. No entanto, não há ação sobre essas mudanças, exceto nas circunstâncias listadas. Se um programa de saída de canal mudar um campo na estrutura de dados MQCD, o novo valor será ignorado pelo processo de canal do IBM MQ. No entanto, o novo valor permanece no MQCD e é passado a qualquer saída restante em uma sequência de saída e a qualquer conversa que compartilhando instância do canal. Para obter mais informações, consulte Mudando campos MQCD em uma saída de canal

Além disso, para programas escritos em C, a função de biblioteca C não reentrante não deve ser usada em um programa de saída de canal.

Linux AIX Se você usar diversas bibliotecas de saída de canal simultaneamente, podem surgir problemas em algumas plataformas UNIX and Linux se o código para duas saídas diferentes contiver funções nomeadas de forma idêntica. Quando uma saída do canal estiver carregada, o carregador dinâmico resolverá os nomes de função na biblioteca de saída para os endereços em que a biblioteca estiver carregada. Se duas bibliotecas de saída definirem funções separadas que por acaso possuem nomes idênticos, esse processo de resolução pode resolver incorretamente os nomes de função de uma biblioteca para usar as funções de outra. Se este problema ocorrer, especifique ao vinculador que ele deve exportar apenas a saída necessária e as funções MQStart, uma vez que essas funções não são afetadas. Outras funções devem receber visibilidade local para que não sejam usadas pelas funções fora de suas próprias bibliotecas de saída. Consulte a documentação do vinculador para obter informações adicionais.

Todas as saídas são chamadas com uma estrutura do parâmetro de saída do canal (MQCXP), uma estrutura de definição do canal (MQCD), um buffer de dados preparado, parâmetro de comprimento de dados e parâmetro de comprimento de buffer. O comprimento do buffer não deve ser excedido:

- Para saídas de mensagem, deve-se permitir que a maior mensagem necessária seja enviada através do canal, além do comprimento da estrutura MQXQH.
- Para as saídas de envio e recebimento, o maior buffer que deve ser permitido é o seguinte:

  LU6.2
  32 KB

  TCP:

  IBM i IBM i 16 KB

  IBM i Outros 32 KB

  Nota: O comprimento máximo utilizável pode ser 2 bytes a menos que esse comprimento. Verifique o valor retornado em **MaxSegmentLength** para obter detalhes. Para obter mais informações sobre o **MaxSegmentLength**, consulte MaxSegmentLength

  NetBIOS:
  64 KB

  SPX:
  64 KB

  Nota: As saídas de recebimento nos canais emissores e as saídas de emissor nos canais receptores usam buffers de 2 KB para TCP.
- Para saídas de segurança, o recurso de enfileiramento distribuído aloca um buffer de 4000 bytes.

É permissível para a saída retornar um buffer alternativo juntamente com os parâmetros relevantes. Consulte “Programas de Saída de Canal para Canais de Mensagens” na página 973 para os detalhes da chamada.

z/OS *Writing channel exit programs on z/OS*

You can use the following information to help you write and compile channel-exit programs for z/OS.

The exits are started as if by a z/OS LINK, in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31 bit addressing mode

The link-edited modules must be placed in the data set specified by the CSQXLIB DD statement of the channel initiator address space procedure; the names of the load modules are specified as the exit names in the channel definition.

When writing channel exits for z/OS, the following rules apply:

- Exits must be written in assembler or C; if C is used, it must conform to the C systems programming environment for system exits, described in the z/OS C/C++ Programming Guide.
- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the channel initiator is running. The new version is used when the channel is restarted.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment, on return, to that at entry.
- Exits must free any storage obtained, or ensure that it is freed by a subsequent exit invocation.

  For storage that is to persist between invocations, use the z/OS STORAGE service, or the 4kmalc library function for System Programming C.

  For more information about this function, see 4kmalc() -- Allocate Page-Aligned Storage.
- All IBM MQ MQI calls except MQCMIT or CSQBCMT and MQBACK or CSQBBAK can be used. They must be contained after MQCONN (with a blank queue manager name). If these calls are used, the exit must be link-edited with the stub CSQXSTUB.

  The exception to this rule is that security channel exits can issue commit and backout MQI calls. To issue such calls, code the verbs CSQXCMT and CSQXBAK in place of MQCMIT or CSQBCMT and MQBACK or CSQBBAK.
- All exits that use stub CSQXSTUB from IBM WebSphere MQ 7.0 or later must be link-edited in a CSQXLIB load library with format PDS-E.
- Exits must not use any system services that cause a wait, because using system services would severely affect the handling of some or all the other channels. Many channels are run under a single TCB typically. If you do something in an exit that causes a wait and you do not use MQXWAIT, it causes all these channels to wait. Causing channels to wait does not give any functional problems, but might have an adverse effect on performance. Most SVCs involve waits, so you must avoid them, except for the following SVCs:
  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

  In general, therefore, avoid SVCs, PCs, and I/O. Instead, use the MQXWAIT call.
- Exits do not issue ESTAEs or SPIEs, apart from in any subtasks they attach, because their error handling might interfere with the error handling performed by IBM MQ. This means that IBM MQ might not be able to recover from an error, or that your exit program might not receive all the error information.

- The MQXWAIT call (see MQXWAIT) provides a wait service that waits for I/O and other events; if this service is used, exits must not use the linkage stack.

  For I/O and other facilities that do not provide non-blocking facilities or an ECB to wait on, a separate subtask must be ATTACHed, and its completion waited for by MQXWAIT; because of the processing that this technique incurs, this facility must be used only by the security exit.

- The MQDISC MQI call does not cause an implicit commit to occur within the exit program. A commit of the channel process is performed only when the channel protocol dictates.

The following exit samples are provided with IBM MQ for z/OS:

CSQ4BAX0
This sample is written in assembler, and illustrates the use of MQXWAIT.

CSQ4BCX1 and CSQ4BCX2
These samples are written in C and illustrate how to access the parameters.

CSQ4BCX3 and CSQ4BAX3
These samples are written in C and assembler respectively.

The CSQ4BCX3 sample (which is pre-compiled into the SCSQAUTH LOADLIB, should function with no changes necessary on the exit itself. You can create a LOADLIB (for example, called MY.TEST.LOADLIB) and copy the SCSQAUTH(CSQ4BCX3) member to it.

To set up a security exit on a client connection, carry out the following procedure:

1. Establish a valid OMVS segment for the user ID that the channel initiator uses.

   This allows the IBM MQ for z/OS channel initiator to use TCP/IP with the z/OS UNIX System Services (z/OS UNIX) socket interface, in order to facilitate exit processing. Note that it is unnecessary to define an OMVS segment for the user ID of any connecting client.
2. Ensure that the exit code itself runs only in a program controlled environment.

   This means everything loaded into the CHINIT address space must be loaded from a program controlled library (meaning all libraries in the STEPLIB), and any libraries named on CSQXLIB and

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   To set a load library as program controlled, use a command similar to this example:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   Then you can activate or refresh the program controlled environment by issuing the command:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```
3. Add the exit LOADLIB to the CSQXLIB DD (in the CHINIT started procedure), by issuing the following command:

   ```
   ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
   ```

   This activates the exit for the named channel.
4. Your external security manager (ESM) lists any other libraries to be program controlled, but note that none of the ESM or C libraries needs to be under program control.

See IBM MQ for z/OS server connection channel for further information on setting up a security exit using the sample CSQ4BCX3.

CSQ4BCX4
This sample is written in C and demonstrates using the **RemoteProduct** and **RemoteVersion** fields in MQCXP.

Related concepts

"Gravando programas de saída de canal em IBM i" on page 980
É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para IBM i.

"Escrevendo programas de saída de canal no AIX, Linux, and Windows" on page 980
É possível usar as informações a seguir para ajudar a escrever programas de saída de canal para sistemas AIX, Linux, and Windows.

Related reference

IBM MQ for z/OS server connection channel

IBM i *Gravando programas de saída de canal em IBM i*

É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para IBM i.

A saída é um objeto de programa gravado na ILE C, ILE RPG ou linguagem ILE COBOL. Os nomes de programa de saída e suas bibliotecas são denominados na definição de canal.

Observe as condições a seguir ao criar e compilar um programa de saída:

- O programa deve ser feito thread-safe e criado com o ILE C, ILE RPG ou compilador ILE COBOL. Para ILE RPG, deve-se especificar a especificação de controle THREAD(*SERIALIZE) e para ILE COBOL deve-se especificar SERIALIZE para a opção THREAD da instrução PROCESS. Os programas também devem estar ligados às bibliotecas encadeadas do IBM MQ: QMQM/LIBMQM_R no caso de ILE C e ILE RPG e AMQ0STUB_R no caso de ILE COBOL. Para obter informações adicionais sobre aplicativos RPG ou COBOL thread-safe, consulte o Guia do Programador adequado para o idioma.
- O IBM MQ for IBM i requer que os programas de saída sejam ativados para suporte de teraspace. (O teraspace é uma forma de memória compartilhada introduzida no OS/400 V4R4.) Para os compiladores ILE RPG e COBOL, quaisquer programas compilados em OS/400 V4R4 ou posterior são ativados. Para C, os programas devem ser compilados com as opções TERASPACE(*YES *TSIFC) especificadas nos comandos CRTCMOD ou CRTBNDC.
- Uma saída que retorna um ponteiro para seu próprio espaço de buffer deve assegurar que o objeto apontado exista além do período de tempo do programa de saída de canal. O ponteiro não pode ser o endereço de uma variável na pilha de programa nem de uma variável no heap de programa. Em vez disso, o ponteiro deve ser obtido do sistema. Um exemplo é um espaço do usuário criado na saída de usuário. Para assegurar que qualquer área de dados alocada pelo programa de saída do canal ainda esteja disponível para o MCA quando o programa é encerrado, a saída do canal deve ser executada no grupo de ativação do responsável pela chamada ou um grupo de ativação nomeado. Faça isso configurando o parâmetro ACTGRP em CRTPGM para um valor definido pelo usuário ou *CALLER. Se o programa for criado dessa maneira, o programa de saída de canal pode alocar memória dinâmica e transmitir um ponteiro para essa memória de volta para o MCA.

Conceitos relacionados

"Escrevendo programas de saída de canal no AIX, Linux, and Windows" na página 980
É possível usar as informações a seguir para ajudar a escrever programas de saída de canal para sistemas AIX, Linux, and Windows.

"Writing channel exit programs on z/OS " na página 978
You can use the following information to help you write and compile channel-exit programs for z/OS.

ALW *Escrevendo programas de saída de canal no AIX, Linux, and Windows*

É possível usar as informações a seguir para ajudar a escrever programas de saída de canal para sistemas AIX, Linux, and Windows.

Siga as instruções descritas em "Composição de saídas e serviços instaláveis em AIX, Linux, and Windows" na página 948. Use as informações específicas da saída do canal a seguir, conforme apropriado:

A saída deve ser escrita em C e é um DLL no Windows.

Defina uma rotina MQStart() simulada na saída e especifique `MQStart` como o ponto de entrada na biblioteca. Figura 106 na página 981 mostra como configurar uma entrada para seu programa:

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

Figura 106. Código-fonte de amostra para uma saída do canal

Ao escrever saídas de canal para Windows usando Visual C++, deve-se escrever o seu próprio arquivo DEF. Um exemplo de como fazer isso é mostrado em Figura 107 na página 981. Para obter informações adicionais sobre como escrever programas de saída do canal, consulte "Gravando programas de saída do canal" na página 976.

```
EXPORTS
ChannelExit
```

Figura 107. Arquivo DEF de amostra para o Windows

Conceitos relacionados

"Gravando programas de saída de canal em IBM i" na página 980
É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para IBM i.

"Writing channel exit programs on z/OS " na página 978
You can use the following information to help you write and compile channel-exit programs for z/OS.

Programas de saída de segurança do canal
É possível usar programas de saída de segurança para verificar se o parceiro na outra extremidade de um canal é genuíno. Isso é conhecido como autenticação.

Para especificar que um canal deve usar uma saída de segurança, especifique o nome da saída no campo **SCYEXIT** da definição de canal..

Nota: A autenticação também pode ser atingida com registros de autenticação de canal. Registros de autenticação de canal fornecem grande flexibilidade para evitar acesso aos gerenciadores de filas por determinados usuários e canais e no mapeamento de usuários remotos para identificadores de usuários do IBM MQ. O suporte ao TLS também é fornecido pelo IBM MQ para autenticar os usuários e fornecer criptografia e verificações de integridade de dados para os dados. Para obter informações adicionais TLS, veja Protocolos de segurança TLS no IBM MQ. No entanto, se ainda requerer formas mais sofisticadas (ou diferentes) de processamento de segurança e outros tipos de verificações e estabelecimento de contexto de segurança, considere gravar as saídas de segurança.

Os atributos Subject e Issuer DN aparecem nos atributos de status do canal a seguir:

- SSLPEER (PCF selector MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (PCF selector MQCACH_SSL_CERT_ISSUER_NAME)

Esses valores são retornados pelos comandos de status do canal, assim como pelos dados passados às saídas de segurança do canal listadas, conforme mostrado:

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

Uma saída de segurança pode ser escrita em C ou Java.

Programas da saída de segurança de canal são chamados nos locais a seguir no ciclo de processamento de um MCA:

- Na inicialização e na finalização do MCA.
- Imediatamente após a negociação de dados inicial ser concluída na inicialização do canal. A extremidade do receptor ou do servidor do canal pode iniciar uma troca de mensagem de segurança com a extremidade remota, fornecendo uma mensagem a ser entregue à saída de segurança na extremidade remota. Também pode recusar a fazer isso. O programa de saída é iniciado novamente para processar qualquer mensagem de segurança recebida da extremidade remota.
- Imediatamente após a negociação de dados inicial ser concluída na inicialização do canal. A extremidade do emissor ou do solicitante do canal processa uma mensagem de segurança recebida da extremidade remota ou inicia uma troca de segurança quando a extremidade remota não puder. O programa de saída é iniciado novamente para processar todas as mensagens de segurança subsequentes que possam ser recebidas.

Um canal do solicitante nunca é chamado com MQXR_INIT_SEC. O canal notifica o servidor de que tem o programa de saída de segurança e o servidor tem, então, a oportunidade de iniciar uma saída de segurança. Se ele não tiver um, informará o solicitante e um fluxo de comprimento zero será retornado para o programa de saída.

Nota: Evite enviar mensagens de segurança de comprimento zero.

Exemplos dos dados trocados por programas de saída de segurança estão ilustrados nas figuras Figura 108 na página 983 a Figura 111 na página 985. Esses exemplos mostram a sequência de eventos que ocorrem envolvendo a saída de segurança do receptor e a saída de segurança do emissor. Linhas sucessivas nas figuras representam a passagem de tempo. Em alguns casos, os eventos no receptor e no emissor não são correlacionados e, portanto, podem ocorrer ao mesmo tempo ou em momentos diferentes. Em outros casos, um evento em um programa de saída resulta em um evento complementar que ocorre posteriormente no outro programa de saída. Por exemplo, emFigura 108 na página 983:

1. O receptor e o emissor são chamados, cada um, com MQXR_INIT, mas essas chamadas não são correlacionadas e, portanto, podem ocorrer ao mesmo tempo ou em momentos diferentes.
2. O receptor é chamado, em seguida, com MQXR_INIT_SEC, mas retorna MQXCC_OK que não requer evento complementar na saída do emissor.
3. O emissor é chamado, em seguida, com MQXR_INIT_SEC. Isso não está correlacionado à chamada do receptor com MQXR_INIT_SEC. O emissor retorna MQXCC_SEND_SEC_MSG, que causa um evento complementar na saída do receptor.
4. O receptor é, então, chamado com MQXR_SEC_MSG e retorna MQXCC_SEND_SEC_MSG, que causa um evento complementar na saída do emissor.
5. O emissor é, então, chamado com MQXR_SEC_MSG e retorna MQXCC_OK que não requer evento complementar na saída do receptor.

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Message transfer begins	

Figura 108. Troca iniciada pelo emissor com acordo

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figura 109. Troca iniciada pelo emissor sem acordo

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figura 110. Troca iniciada pelo receptor com acordo

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

Figura 111. Troca iniciada pelo receptor sem acordo

O programa da saída de segurança de canal é passado a um buffer de agente que contém dados de segurança, excluindo qualquer cabeçalho de transmissão, gerado pela saída de segurança. Esses dados podem ser quaisquer dados adequados para que a extremidade do canal seja capaz de executar a validação de segurança.

O programa de saída de segurança em ambas as extremidades de envio e de recebimento do canal de mensagens pode retornar um de dois códigos de resposta para qualquer chamada:

- Troca de segurança finalizada sem erros
- Suprimir o canal e fechar

Nota:

1. As saídas de segurança do canal geralmente funcionam em pares. Ao definir os canais apropriados, certifique-se de que os programas de saída compatíveis sejam denominados para ambas as extremidades do canal.
2. IBM i No IBM i, os programas de saída de segurança que foram compilados com `Use adopted authority` (USEADPAUT=*YES) podem adotar autoridade QMQM ou QMQMADM. Tome cuidado para que a saída não use esse recurso para constituir um risco de segurança ao seu sistema.
3. Em um canal TLS no qual a outra extremidade do canal fornece um certificado, a saída de segurança recebe o Nome distinto do sujeito deste certificado no campo MQCD acessado por SSLPeerNamePtr e o Nome distinto do emissor no campo MQCXP acessado por SSLRemCertIssNamePtr. Os usos desse nome podem ser:
   - Para restringir o acesso por meio do canal TLS.
   - Para mudar MQCD.MCAUserIdentifier com base no nome.

Conceitos relacionados

Conceitos de TLS (Transport Layer Security)

Referências relacionadas

Registros de Autenticação de Canal

Escrevendo uma saída de segurança

É possível escrever uma saída de segurança usando o código de estrutura básica da saída de segurança.

Figura 112 na página 986 ilustra como escrever uma saída de segurança.

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

Figura 112. Código de estrutura básica da saída de segurança

O IBM MQ Entry Point MQStart padrão deve existir, mas não é necessário para executar qualquer função. O nome da função (EntryPoint, neste exemplo) pode ser mudado, mas a função deve ser exportada quando a biblioteca for compilada e vinculada. Como no exemplo anterior, os ponteiros pChannelExitParms devem ter cast efetuado para PMQCXP e pChannelDefinition deve ter cast efetuado para PMQCD. Para obter informações gerais sobre como chamar saídas do canal e sobre o uso de parâmetros, consulte MQ_CHANNEL_EXIT. Esses parâmetros são usados em uma saída de segurança da seguinte forma:

PMQVOID pChannelExitParms
entrada/saída

Ponteiro para a estrutura MQCXP – efetuar cast para PMQCXP para acessar campos. Essa estrutura é usada para comunicação entre a Saída e o MCA. Os campos a seguir no MQCXP são de interesse especial para Saídas de segurança:

ExitReason
Informa a Saída de segurança o estado atual na troca de segurança e é usado ao decidir qual ação tomar.

ExitResponse
A resposta ao MCA que determina o próximo estágio na troca de segurança.

ExitResponse2
Sinalizações de controle extra para gerir como o MCA interpreta a resposta da Saída de segurança.

ExitUserArea
16 bytes (máximo) de armazenamento que podem ser usados pela Saída de segurança para manter o estado entre chamadas.

ExitData
Contém os dados especificados no campo SCYDATA da definição de canal (32 bytes preenchidos à direita com espaços em branco).

PMQVOID pChannelDefinition
entrada/saída

Ponteiro para a estrutura MQCD – efetuar cast para PMQCD para acessar campos. Esse parâmetro contém a definição do canal. Os campos a seguir no MQCD são de interesse especial para Saídas de segurança:

ChannelName
O nome do canal (20 bytes preenchidos à direita com espaços em branco).

ChannelType
Um código que define o tipo de canal.

Identificador de usuário do MCA
Esse grupo de três campos é inicializado para o valor do campo MCAUSER especificado na definição do canal. Qualquer identificador de usuário especificado pela Saída de segurança nesses campos é usado para controle de acesso (não aplicável a canais SDR, SVR, CLNTCONN ou CLUSSDR).

MCAUserIdentifier
Primeiros 12 bytes do identificador preenchidos à direita com espaços em branco.

LongMCAUserIdPtr
Ponteiro para um buffer que contém o identificador de comprimento integral (não garantida finalização nula) tem prioridade sobre MCAUserIdentifier.

LongMCAUserIdLength
Comprimento da sequência apontada por LongMCAUserIdPtr – deve ser configurado se LongMCAUserIdPtr estiver configurado.

Identificador de usuário remoto
Se aplica somente a pares de canais CLNTCONN/SVRCONN. Se nenhuma Saída de segurança CLNTCONN for definida, então, esses três campos serão inicializados pelo MCA do cliente, portanto, eles podem conter um identificador de usuário do ambiente do cliente que pode ser usado por uma Saída de segurança SVRCONN para autenticação e ao especificar o Identificador de usuário do MCA. Se uma Saída de segurança CLNTCONN for definida, então, esses campos não serão inicializados e podem ser configuradas pelo Saída de segurança CLNTCONN ou mensagens de segurança podem ser usadas para passar um identificador de usuário do Cliente para o Servidor.

Identificador de usuário remoto
Primeiros 12 bytes do identificador preenchidos à direita com espaços em branco.

LongRemoteUserIdPtr
Ponteiro para um buffer que contém o identificador de comprimento integral (não garantida finalização nula) tem prioridade sobre RemoteUserIdentifier.

LongRemoteUserIdLength
Comprimento da sequência apontada por LongRemoteUserIdPtr – deve ser configurado se LongRemoteUserIdPtr estiver configurado.

PMQLONG pDataLength
entrada/saída

Ponteiro para MQLONG. Contém o comprimento de qualquer Saída de segurança contida no AgentBuffer após a chamada da Saída de segurança. Deve ser configurado por uma Saída de segurança para o comprimento de qualquer mensagem que está sendo enviada no AgentBuffer ou ExitBuffer.

PMQLONG pAgentBufferLength
entrada

Ponteiro para MQLONG. O comprimento dos dados contidos no AgentBuffer na chamada da Saída de segurança.

PMQVOID pAgentBuffer
entrada/saída

Na chamada da Saída de segurança, aponta para qualquer mensagem enviada da saída do parceiro. Se ExitResponse2 na estrutura MQCXP tiver a sinalização MQXR2_USE_AGENT_BUFFER configurada (padrão), então, uma Saída de segurança precisa configurar esse parâmetro para apontar para qualquer dado de mensagem que esteja sendo enviado.

PMQLONG pExitBufferLength
entrada/saída

Ponteiro para MQLONG. Esse parâmetro é inicializado para 0 na primeira chamada de uma Saída de segurança e o valor retornado é mantido entre as chamadas à Saída de segurança durante uma troca de segurança.

PMQPTR pExitBufferAddr
entrada/saída

Esse parâmetro é inicializado para um ponteiro nulo na primeira chamada de uma Saída de segurança e o valor retornado é mantido entre as chamadas à Saída de segurança durante uma troca de segurança. Se a sinalização MQXR2_USE_EXIT_BUFFER for configurada no ExitResponse2 na estrutura MQCXP, então, uma Saída de segurança precisa configurar esse parâmetro para apontar para qualquer dado de mensagem que esteja sendo enviado.

Diferenças em comportamento entre saídas de segurança definidas em pares de canais CLNTCONN/SVRCONN e outros pares de canais
As saídas de segurança podem ser definidas em todos os tipos de canal. No entanto, o comportamento das saídas de segurança definidas nos pares de canal CLNTCONN/SVRCONN é um pouco diferente das saídas de segurança definidas em outros pares de canal.

Uma saída de segurança em um canal CLNTCONN pode definir o Identificador de usuário remoto na definição de canal para processamento por uma saída de SVRCONN parceiro ou para autorização OAM se a Saída de segurança SVRCONN não estiver definida e o campo MCAUSER de SVRCONN não estiver configurado.

Se nenhuma Saída de segurança CLNTCONN estiver definida, o Identificador de usuário remoto na definição de canal é configurado para um identificador de usuário a partir do ambiente do cliente (que pode estar em branco) pelo cliente MCA.

Uma troca de segurança entre as Saídas de segurança definidas em um par de canais CLNTCONN e SVRCONN é concluída com êxito quando a Saída de segurança SVRCONN retorna um ExitResponse de MQXCC_OK. Uma troca de segurança entre outros pares de canal é concluída com êxito quando a Saída de segurança que iniciou a troca retorna um ExitResponse de MQXCC_OK.

No entanto, o código de ExitResponse MQXCC_SEND_AND_REQUEST_SEC_MSG pode ser usado para forçar a continuação da troca de segurança: se um ExitResponse de MQXCC_SEND_AND_REQUEST_SEC_MSG é retornado por uma Saída de segurança CLNTCONN ou SVRCONN, em seguida, a saída de parceiro deve responder enviando uma mensagem de segurança (não MQXCC_OK ou uma resposta nula) ou o canal é encerrado. Para Saídas de segurança definidas em outros tipos de canal, um ExitResponse de MQXCC_OK retornado em resposta a um MQXCC_SEND_AND_REQUEST_SEC_MSG da Saída de segurança do parceiro resulta na continuação da troca de segurança como se uma resposta nula fosse retornada e não na finalização do canal.

Saída de segurança SSPI
O IBM MQ for Windows fornece uma saída de segurança que fornece autenticação para canais do IBM MQ usando a Security Services Programming Interface (SSPI). A SSPI fornece os recursos de segurança integrados do Windows.

Essa saída de segurança é para o cliente IBM MQ e o servidor IBM MQ.

Os pacotes de segurança são carregados a partir de security.dll ou secur32.dll. Esses DLLs são fornecidos com o seu sistema operacional.

Autenticação unilateral é fornecida no Windows, usando os serviços de autenticação NTLM. Autenticação de duas vias é fornecida em Windows 2000, usando os serviços de autenticação do Kerberos.

O programa de saída de segurança é fornecido no formato de origem e de objeto. É possível usar o código de objeto no estado em que se encontra ou usar o código-fonte como um ponto de início para criar seus próprios programas de saída de usuário. Para obter mais informações sobre como usar o objeto ou o código de origem da saída de segurança SSPI, consulte "Usando a saída de segurança SSPI no Windows" na página 1149

Programas de saída de envio e de recebimento do canal
É possível usar as saídas de envio e de recebimento para executar tarefas como compactação de dados e descompactação. É possível especificar uma lista de programas de saída de envio e de recebimento a ser executada em sucessão.

Programas de saída de envio e de recebimento do canal são chamados nos seguintes locais no ciclo de processamento de um MCA:

- Os programas de saída de envio e de recebimento do canal são chamados para inicialização na iniciação do MCA e para finalização na rescisão do MCA.
- O programa de saída de envio é chamado em um ou outro final do canal dependendo do final no qual uma transmissão para uma transferência de mensagem é enviada, imediatamente antes de uma transmissão ser enviada através do link. A nota 4 explica por que as saídas estão disponíveis em ambas as direções, ainda que os canais de mensagens enviem mensagens apenas em uma direção.
- O programa de saída de recebimento é chamado em um ou outro final do canal dependendo do final no qual uma transmissão para uma transferência de mensagem é recebida, imediatamente após uma transmissão ter sido obtida a partir do link. A nota 4 explica por que as saídas estão disponíveis em ambas as direções, ainda que os canais de mensagens enviem mensagens apenas em uma direção.

Pode haver várias transmissões para uma transferência de mensagem e pode haver várias iterações dos programas de saída de envio e recebimento antes de uma mensagem atingir a saída de mensagem no final receptor.

Aos programas de saída de envio e de recebimento do canal é passado um buffer de agente que contém dados de transmissão como enviados ou recebidos do link de comunicações. Para os programas de saída de envio, os primeiros 8 bytes do buffer são reservadas para uso pelo MCA e não devem ser mudados. Se o programa retornar um buffer diferente, então estes primeiros 8 bytes devem existir no novo buffer. O formato dos dados apresentados ao programa de saída não está definido.

Um bom código de resposta deve ser retornado pelos programas de saída de envio e recebimento. Qualquer outra resposta faz um fim anormal do MCA.

Nota: Não emita uma chamada MQGET, MQPUT ou MQPUT1 no ponto de sincronização a partir de uma saída de envio ou de recebimento.

Nota:

1. As saídas de envio e recebimento geralmente trabalham em pares. Por exemplo, uma saída de envio pode compactar os dados e uma saída de recebimento descompacte-los ou uma saída de envio pode criptografar os dados e uma saída de recebimento decriptografá-los. Ao definir os canais apropriados, certifique-se de que os programas de saída compatíveis sejam denominados para ambas as extremidades do canal.
2. Se a compactação estiver ativada para o canal, serão transmitidos dados compactados às saídas.
3. Saídas de envio e recebimento do canal podem ser chamadas para segmentos de mensagem diferentes dos segmentos dos dados do aplicativo, por exemplo, mensagens de status. Eles não são chamados durante o diálogo de inicialização nem durante a fase de verificação de segurança.
4. Embora canais de mensagens enviem mensagens apenas em uma direção, os dados do canal de controle, como pulsações e fim de processamentos em lote, flui em ambas as direções, e essas saídas estarão disponíveis em ambas as direções, também. No entanto, alguns dos fluxos de dados de inicialização do canal inicial são isentos de processamento por qualquer uma das saídas.
5. Há circunstâncias em que as saídas de envio e de recebimento podem ser invocadas fora de sequência; por exemplo, se estiver sendo executada uma série de programas de saída ou estiver também estiver executando saídas de segurança. Em seguida, quando a saída de recebimento é chamada pela primeira vez para processar dados, ela pode receber dados que não passaram pela saída de envio correspondente. Se a saída de recebimento apenas executasse a operação, por exemplo descompactação, sem antes verificar se era necessária, os resultados seriam inesperados.

   É necessário codificar suas saídas de envio e de recebimento de tal forma que a saída de recebimento possa verificar se os dados que está recebendo foram processados pela saída de envio correspondente. A maneira recomendada de fazer isso é codificar seus programas de saída de modo que:

   - A saída de envio configura o valor do nono byte de dados para 0 e desloca todos os dados juntos em 1 byte antes de executar a operação. (Os primeiros 8 bytes são reservados para uso pelo MCA.)
   - Se a saída de recebimento receber dados que possuem um 0 no byte 9, ela saberá que os dados vêm da saída de envio. Ela remove o 0, executa a operação complementar, e desloca os dados resultantes de volta em 1 byte.
   - Se a saída de recebimento receber dados que tenham algo diferente de 0 no byte 9, ela assumirá que a saída de envio não foi executada e enviará os dados de volta para o responsável pela chamada inalterados.

   Ao usar saídas de segurança, se o canal for encerrado pela saída de segurança, é possível que uma saída de envio seja chamada sem a saída de recebimento correspondente. Uma maneira de evitar esse problema é codificar a saída de segurança para configurar uma sinalização em MQCD.SecurityUserData ou MQCD.SendUserData, por exemplo, quando a saída decidir encerrar o canal. Em seguida, a saída de envio precisará verificar esse campo e processar os dados apenas se a sinalização não estiver configurada. Essa verificação evita que a saída de envio atere os dados sem necessidade e, portanto, impede qualquer conversão de erros que possa ocorrer se a saída de segurança tiver recebido dados alterados.

Programas de saída de envio do canal - reservando espaço
É possível usar saídas de envio e de recebimento para transformar os dados antes da transmissão. Programas de saída de envio do canal podem incluir seus próprios dados sobre a transformação reservando espaço no buffer de transmissão.

Esses dados são processados pelo programa de saída de recebimento e, em seguida, removidos do buffer. Por exemplo, você pode desejar criptografar os dados e incluir uma chave de segurança para decriptografia.

Como reservar espaço e usá-lo

Quando o programa de saída de envio é chamado para inicialização, configure o campo *ExitSpace* de MQXCP para o número de bytes a ser reservado. Consulte MQCXP para obter detalhes. *ExitSpace* pode

ser configurado somente durante a inicialização, ou seja, quando *ExitReason* tiver o valor MQXR_INIT. Quando a saída de envio é chamada imediatamente antes da transmissão, com *ExitReason* configurado para MQXR_XMIT, *ExitSpace* bytes são reservados no buffer de transmissão. *ExitSpace* não é suportado em z/OS.

A saída de envio não usa todo o espaço reservado. Ela pode usar menos que *ExitSpace* bytes ou, se o buffer de transmissão não estiver cheio, a saída pode usar mais do que a quantia reservada. Ao configurar o valor de *ExitSpace*, deve-se deixar pelo menos 1 KB para dados de mensagem no buffer de transmissão. O desempenho do canal pode ser afetado se o espaço reservado for usado para grandes quantias de dados.

O buffer de transmissão é normalmente 32KB longo. No entanto, se o canal usar TLS, o tamanho do buffer de transmissão será reduzido para 15.352 bytes para se ajustar dentro do comprimento de registro máximo definido pela RFC 6101 e a família de padrões TLS relacionada. Mais 1024 bytes são reservados para uso pelo IBM MQ, portanto, o espaço de buffer de transmissão máximo utilizável por saídas de envio é 14.328 bytes.

O que acontece na extremidade de recebimento do canal

Os programas de saída de recebimento do canal devem ser configurados para serem compatíveis com as saídas de envio correspondentes. As saídas de recebimento devem saber o número de bytes no espaço reservado e devem remover os dados nesse espaço.

Várias saídas de envio

É possível especificar uma lista de programas de saída de envio e de recebimento a ser executada em sucessão. O IBM MQ mantém um total para o espaço reservado por todas as saídas de envio. Esse espaço total deve deixar pelo menos 1 KB para dados de mensagem no buffer de transmissão.

O exemplo a seguir mostra como o espaço é alocado para três saídas de envio, chamadas em sucessão:

1. Quando chamadas para inicialização:
   - A saída de envio A reserva 1 KB.
   - A saída de envio B reserva 2 KB.
   - A saída de envio C reserva 3 KB.
2. O tamanho de transmissão máximo é 32 KB e os dados do usuário têm 5 KB de comprimento.
3. A saída A é chamada com 5 KB de dados; até 27 KB estão disponíveis, pois 5 KB são reservados para as saídas B e C. A saída A soma 1 KB, a quantia que reservou.
4. A saída B é chamada com 6 KB de dados; até 29 KB estão disponíveis, pois 3 KB estão reservados para a saída C. A saída B soma 1 KB, menos do que os 2 KB que reservou.
5. A saída C é chamada com 7 KB de dados; até 32 KB estão disponíveis. A saída C inclui 10K, mais do que os 3 KB que reservou. Essa quantia é válida, porque a quantia total de dados, 17 KB, é menor que o máximo de 32 KB.

O tamanho máximo do buffer de transmissão para um canal usando TLS é 15.352 bytes, não 32 Kb. Isso ocorre porque os segmentos de transmissão de soquete seguro subjacentes são limitados a 16 Kb e parte do espaço é necessária para as sobrecargas de registro TLS. Mais 1024 bytes são reservados para uso pelo IBM MQ, portanto, o espaço de buffer de transmissão máximo utilizável por saídas de envio é 14.328 bytes.

Programas de saída de mensagem do canal
É possível usar a saída de mensagem do canal para executar tarefas, como criptografia no link, validação ou substituição de IDs de usuário recebidos, conversão de dados de mensagem, utilização de diário e manipulação de mensagem de referência. É possível especificar uma lista de programas de saída de mensagem a serem executados em sucessão.

Programas de saída de mensagem do canal são chamados nos seguintes locais no ciclo de processamento do MCA:

- Na inicialização e na finalização do MCA
- Imediatamente após um MCA de envio ter emitido uma chamada MQGET
- Antes que o MCA de recebimento emita uma chamada MQPUT

A saída de mensagem é passada por um buffer de agente que contém o cabeçalho da fila de transmissão MQXQH e o texto da mensagem do aplicativo, conforme recuperado da fila. O formato de MQXQH é fornecido em Cabeçalho MQXQH - fila de transmissão.

Multi Se você usar mensagens de referência (ou seja, mensagens que contêm somente um cabeçalho que aponta para algum outro objeto que deve ser enviado), a saída de mensagem reconhecerá o cabeçalho, MQRMH. Ele identifica o objeto, recupera-o da maneira apropriada, anexa-o ao cabeçalho e, em seguida, passa-o para o MCA para transmissão para o MCA de recebimento. No MCA de recebimento, outra saída de mensagem reconhece que essa mensagem é uma mensagem de referência, extrai o objeto e passo o cabeçalho para a fila de destino. Consulte “Mensagens de referência e transferências de objetos grandes” na página 807 e “Executando as amostras Reference Message” na página 1120 para obter mais informações sobre mensagens de referência e algumas saídas de mensagem de amostra que as manipulam.

As saídas de mensagem podem retornar as respostas a seguir:

- Envie a mensagem (saída GET). A mensagem pode ter sido mudada pela saída. (Isso retorna MQXCC_OK.)
- Coloque a mensagem na fila (saída PUT). A mensagem pode ter sido mudada pela saída. (Isso retorna MQXCC_OK.)
- Não processe a mensagem. A mensagem é colocada na fila de mensagens não entregues pelo MCA.
- Feche o canal.
- Código de retorno inválido, o que faz com que o MCA seja finalizado de forma anormal.

Nota:

1. As saídas de mensagem são chamadas uma vez para cada mensagem completa transferida, mesmo quando a mensagem for dividida em partes.
2. Linux AIX Se você fornecer uma saída de mensagem no AIX ou Linux, a conversão automática de IDs de usuário para caracteres minúsculos (descrito aqui) não funcionará.
3. Uma saída é executada no mesmo encadeamento que o próprio MCA. Ela também é executada na mesma unidade de trabalho (UOW) que o MCA, pois usa a mesma manipulação de conexões. Assim, quaisquer chamadas feitas sob o ponto de sincronização são confirmadas ou restauradas pelo canal no fim do lote. Por exemplo, um programa de saída de mensagem do canal pode enviar mensagens de notificação para outro e essas mensagens são confirmadas na fila somente quando o lote que contém a mensagem original for confirmado.

   Portanto, é possível emitir as chamadas MQI do ponto de sincronização por meio de um programa de saída de mensagem do canal.

Conversão de mensagens fora da saída de mensagem
Antes de chamar a saída de mensagem, o MCA de recebimento executa algumas conversões na mensagem. Este tópico descreve os algoritmos usados para executar as conversões.

Quais cabeçalhos são processados

Uma rotina de conversão é executada no MCA do receptor antes que a saída de mensagem seja chamada. A rotina de conversão começa com o cabeçalho MQXQH no início da mensagem. A rotina de conversão então processa os cabeçalhos encadeados que seguem MQXQH, executando a conversão conforme necessário. Os cabeçalhos encadeados podem se estender além do deslocamento contido no parâmetro HeaderLength dos dados de MQCXP que são passados à saída de mensagem do receptor. Os cabeçalhos a seguir são convertidos no local:

- MQXQH (nome do formato "MQXMIT")
- MQMD (esse cabeçalho faz parte do MQXQH e não tem nenhum nome de formato)
- MQMDE (nome do formato "MQHMDE")
- MQDH (nome do formato "MQHDIST")
- MQWIH (nome do formato "MQHWIH")

Os cabeçalhos a seguir não são convertidos, mas são passados à medida que o MCA continua a processar os cabeçalhos encadeados:

- MQDLH (nome do formato "MQDEAD")
- quaisquer cabeçalhos com nomes de formato iniciados pelos três caracteres 'MQH' (por exemplo "MQHRF") que não tenham sido mencionados de outra forma

Como os cabeçalhos são processados

O parâmetro Format de cada cabeçalho do IBM MQ é lido pelo MCA. O parâmetro Format é 8 bytes dentro do cabeçalho, que são 8 caracteres de byte único que contém um nome.

O MCA então interpreta os dados após cada cabeçalho como sendo do tipo denominado. Se o Format for o nome de um tipo de cabeçalho elegível para conversão de dados do IBM MQ, ele será convertido. Se for outro nome indicando dados não MQ (por exemplo, MQFMT_NONE ou MQFMT_STRING), o MCA para o processamento dos cabeçalhos.

Qual é o HeaderLength de MQCXP?

O parâmetro HeaderLength nos dados de MQCXP fornecido para uma saída de mensagem é o comprimento total dos cabeçalhos MQXQH (que inclui o MQMD), MQMDE e MQDH no início da mensagem. Esses cabeçalhos são encadeados usando os nomes e comprimentos de 'Format'.

MQWIH

Cabeçalhos encadeados podem se estender além do HeaderLength até a área de dados do usuário. O cabeçalho MQWIH, se estiver presente, é um desses cabeçalhos que aparece além do HeaderLength.

Se houver um cabeçalho MQWIH nos cabeçalhos encadeados, ele será convertido no local antes da saída de mensagem do receptor ser chamada.

Programa de saída de nova tentativa de mensagem do canal
A saída de nova tentativa de mensagem do canal é chamada quando uma tentativa de abrir a fila de destino não é bem-sucedida. É possível usar a saída para determinar sob quais circunstâncias tentar novamente, quantas vezes tentar novamente e com que frequência.

Essa saída também é chamada na extremidade de recebimento na inicialização e na finalização do MCA.

A saída de nova tentativa de mensagem do canal tem um buffer de agente passado que contém o cabeçalho da fila de transmissão, MQXQH, e o texto da mensagem do aplicativo conforme recuperado da fila. O formato de MQXQH é fornecido em Visão geral de MQXQH.

A saída é chamada para todos os códigos de razão; a saída determina para quais códigos de razão deseja que o MCA tente novamente, quantas vezes e em quais intervalos. (O valor do conjunto de contagem de novas tentativas de mensagem quando o canal foi definido é passado para a saída no MQCD, mas a saída pode ignorar esse valor.)

O campo MsgRetryCount em MQCXP é incrementado pelo MCA cada vez que a saída é chamada e a saída retorna MQXCC_OK com o tempo de espera contido no campo MsgRetryInterval de MQCXP ou MQXCC_SUPPRESS_FUNCTION. Novas tentativas continuam indefinidamente até a saída retornar MQXCC_SUPPRESS_FUNCTION no campo ExitResponse de MQCXP. Consulte MQCXP para obter informações sobre a ação executada pelo MCA para esses códigos de conclusão.

Se todas as novas tentativas forem mal-sucedidas, a mensagem será gravada na fila de mensagens não entregues. Se não houver nenhuma fila de mensagens não entregues disponível, o canal para.

Se você não definir uma saída de nova tentativa de mensagem para um canal e ocorrer uma falha que é provavelmente temporária, por exemplo, MQRC_Q_FULL, o MCA usa a contagem de nova tentativa de mensagem e os intervalos de nova tentativa de mensagem configurados de quando o canal foi definido. Se a falha for de natureza mais permanente e você não tiver definido um programa de saída para manipulá-la, a mensagem será gravada na fila de mensagens não entregues.

Programa de saída de autodefinição do canal
A saída de autodefinição de canal pode ser usada quando uma solicitação for recebida para iniciar um receptor ou um canal de conexão do servidor, mas nenhuma definição para esse canal existe (não para o IBM MQ for z/OS). Também pode ser chamada em todas as plataformas para canais emissores de cluster e receptores de cluster para permitir modificação de definição para uma instância do canal.

A saída de autodefinição de canal pode ser chamada em todas as plataformas, exceto no z/OS, quando uma solicitação for recebida para iniciar um receptor ou canal de conexão do servidor, mas não existe nenhuma definição de canal. É possível usá-la para modificar a definição padrão fornecida para um receptor definido automaticamente ou canal de conexão do servidor, SYSTEM.AUTO.RECEIVER ou SYSTEM.AUTO.SVRCON. Consulte Preparando canais para obter uma descrição de como as definições de canal podem ser criadas automaticamente.

A saída de autodefinição de canal também pode ser chamada quando uma solicitação for recebida para iniciar um canal emissor de clusters. Ela pode ser chamada para canais emissores de clusters e receptores de clusters para permitir modificação da definição para essa instância do canal. Nesse caso, a saída também se aplica ao IBM MQ for z/OS. Um uso comum da saída de autodefinição de canal é mudar os nomes de saídas de mensagens (MSGEXIT, RCVEXIT, SCYEXIT e SENDEXIT), porque nomes de saídas têm formatos diferentes em plataformas diferentes. Se nenhuma saída de autodefinição de canal for especificada, o comportamento padrão no z/OS é examinar um nome de saída distribuído no formato *[path]/libraryname(function)* e ter até oito caracteres de function, se presente, ou libraryname. No z/OS, um programa de saída de autodefinição de canal deve alterar os campos endereçado por MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr e ReceiveUserDataPtr, em vez de pelos campos MsgExit, MsgUserData, SendExit, SendUserData, ReceiveExit e ReceiveUserData em si.

Para obter mais informações, consulte Trabalhando com canais autodefinidos.

Como com outras saídas de canal, a lista de parâmetros é:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

`ChannelExitParms` estão descritos em MQCXP. `ChannelDefinition` está descrito em MQCD.

MQCD contém os valores que são usados na definição de canal padrão, se eles não forem alterados pela saída. A saída pode modificar apenas um subconjunto dos campos; consulte MQ_CHANNEL_AUTO_DEF_EXIT. No entanto, a tentativa de mudar outros campos não causa um erro.

A saída de autodefinição de canal retorna uma resposta de MQXCC_OK ou MQXCC_SUPPRESS_FUNCTION. Se nenhuma dessas respostas for retornada, o MCA continua o processamento como se MQXCC_SUPPRESS_FUNCTION tivesse sido retornada. Ou seja, a autodefinição é abandonada, nenhuma nova definição de canal é criada e o canal não pode ser iniciado.

ALW *Compilando programas de saída de canais em sistemas AIX, Linux, and Windows*

Use os exemplos a seguir para ajudar a compilar programas de saída de canais para sistemas AIX, Linux, and Windows.

Windows

Windows

O comando do compilador e vinculador para programas de saída de canal no Windows:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

SistemasAIX and Linux

Linux AIX

Nesses exemplos, `exit` é o nome da biblioteca e `ChannelExit` é o nome da função. No AIX o arquivo de exportação é chamado `exit.exp`. Esses nomes são usados pela definição de canal para fazer referência ao programa de saída usando o formato descrito em Definição de canal MQCD. Consulte também o parâmetro MSGEXIT do comando DEFINE CHANNEL.

AIX Comandos de amostra do compilador e vinculador para saídas de canal no AIX:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Comandos de amostra do compilador e vinculador para saídas de canal no Linux em que o gerenciador de filas é de 32 bits:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Comandos de amostra do compilador e vinculador para saídas de canal no Linux em que o gerenciador de filas é de 64 bits:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

No cliente, uma saída de 32 bits ou 64 bits pode ser usada. Essa saída deve ser vinculada a mqic_r.

AIX No AIX, todas as funções que são chamadas por IBM MQ devem ser exportadas. Um arquivo de exportação de amostra para este make file:

```
#
!channelExit
MQStart
```

Configurando saídas do canal

Para chamar a saída do canal, deve-se denominá-la na definição de canal.

Saídas do canal devem ser denominadas na definição de canal. É possível realizar essa denominação quando definir os canais pela primeira vez ou é possível incluir as informações posteriormente usando, por exemplo, o comando ALTER CHANNEL do MQSC. Também é possível fornecer nomes de saída do canal na estrutura de dados do canal MQCD. O formato do nome de saída depende da plataforma do IBM MQ; consulte MQCD ou Comandos MQSC para obter informações.

Se a definição de canal não contiver um nome de programa de saída do usuário, a saída do usuário não será chamada.

A saída de autodefinição de canal é uma propriedade do gerenciador de filas, não do canal individual. Para que essa saída seja chamada, ela deve ser denominada na definição do gerenciador de filas. Para alterar uma definição do gerenciador de filas, use o comando ALTER QMGR do MQSC.

Escrevendo saídas de conversão de dados

Esta coleção de tópicos contém informações sobre como escrever saídas de conversão de dados.

Nota: Não suportado no MQSeries for VSE/ESA.

Ao executar um MQPUT, seu aplicativo cria o descritor de mensagens (MQMD) da mensagem. Como o IBM MQ precisa ser capaz de entender os conteúdos do MQMD independentemente da plataforma na qual é criado, ele é convertido automaticamente pelo sistema.

Dados do aplicativo, no entanto, não são automaticamente convertidos. Se os dados de caracteres estiverem sendo trocados entre plataformas em que os campos `CodedCharSetId` e `Encoding` diferem, por exemplo, entre ASCII e EBCDIC, o aplicativo deve providenciar a conversão da mensagem. A conversão de dados do aplicativo pode ser executada pelo gerenciador de filas em si ou por um programa de saída do usuário, referido como uma *saída de conversão de dados*. O gerenciador de filas pode executar a conversão de dados em si, usando uma de suas rotinas de conversão integradas, se os dados do aplicativo estiverem em um dos formatos integrados (como MQFMT_STRING). Este tópico contém informações sobre o recurso de saída de conversão de dados que o IBM MQ fornece para quando os dados do aplicativo não estão em um formato integrado.

O controle pode ser passado para a saída de conversão de dados durante uma chamada MQGET. Isso evita converter em diferentes plataformas antes de atingir o destino final. No entanto, se o destino final for uma plataforma que não suporte a conversão de dados no MQGET, deve-se especificar CONVERT(YES) no canal emissor que envia os dados para seu destino final. Isso assegura que o IBM MQ converta os dados durante a transmissão. Nesse caso, sua saída de conversão de dados deve residir no sistema em que o canal emissor está definido.

A chamada MQGET é emitida diretamente pelo aplicativo. Configure os campos `CodedCharSetId` e `Encoding` no MQMD para o conjunto de caracteres e codificação necessários. Se seu aplicativo usar o mesmo conjunto de caracteres e codificação que o gerenciador de filas, configure `CodedCharSetId` para MQCCSI_Q_MGR e `Encoding` para MQENC_NATIVE. Após a chamada MQGET ser concluída, esses campos têm os valores apropriados para os dados de mensagem retornados. Eles podem ser diferentes dos valores necessários, se a conversão não tiver sido bem-sucedida. Seu aplicativo deve reconfigurar esses campos para os valores necessários antes de cada chamada MQGET.

As condições necessárias para que a saída de conversão de dados seja chamada são definidas para a chamada MQGET em MQGET.

Para obter uma descrição dos parâmetros que são passados à saída de conversão de dados e observações de uso detalhadas, consulte Conversão de dados para a chamada MQ_DATA_CONV_EXIT e a estrutura MQDXP.

Programas que convertem dados do aplicativo entre codificações de máquina diferentes e CCSIDs devem ser estar em conformidade com a data conversion interface (DCI) do IBM MQ.

Para clientes Multicast, as saídas de API e as saídas de conversão de dados precisam ser capazes de executar no lado do cliente, já que algumas mensagens podem não passar pelo gerenciador de filas. As bibliotecas a seguir fazem parte dos pacotes do cliente, assim como dos pacotes do servidor:

Tabela 143. Bibliotecas que estão nos pacotes do cliente e do servidor

Sistema operacional	**Bibliotecas**
AIX	32 bits e 64 bits: libmqm.a e libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 bits e 64 bits: libmqm.s e libmqm_r.so
Windows	32 bits e 64 bits: mqm.dll e mqm.pdb

Chamando a saída de conversão de dados

Uma saída de conversão de dados é uma saída escrita pelo usuário que recebe controle durante o processamento de uma chamada MQGET.

A saída será chamada se as instruções a seguir forem verdadeiras:

- A opção MQGMO_CONVERT for especificada na chamada MQGET.
- Alguns ou todos os dados da mensagem não estiverem no conjunto de caracteres ou na codificação solicitado.
- O campo *Format* na estrutura MQMD associada à mensagem não for MQFMT_NONE.
- O *BufferLength* especificado na chamada MQGET não for zero.
- O comprimento dos dados da mensagem não for zero.
- A mensagem contiver dados que tenham um formato definido pelo usuário. O formato definido pelo usuário pode ocupar a mensagem inteira ou ser precedido por um ou mais formatos integrados. Por exemplo, o formato definido pelo usuário pode ser precedido por um formato MQFMT_DEAD_LETTER_HEADER. A saída for chamada para converter somente o formato definido pelo usuário; o gerenciador de filas converte quaisquer formatos integrados que precedam o formato definido pelo usuário.

  Uma saída escrita pelo usuário também pode ser chamada para converter um formato incorporado, mas isso acontece somente se as rotinas de conversão integradas não puderem converter o formato integrado com sucesso.

Há algumas outras condições, descritas integralmente na observações de uso da chamada MQ_DATA_CONV_EXIT em MQ_DATA_CONV_EXIT.

Consulte MQGET para obter detalhes da chamada MQGET. Saídas de conversão de dados não podem usar chamadas MQI, além de MQXCNVC.

Uma nova cópia da saída é carregada quando um aplicativo tenta recuperar a primeira mensagem que usa esse *Format* desde que o aplicativo conectou ao gerenciador de filas. Uma nova cópia também pode ser carregada em outros momentos se o gerenciador de filas tiver descartado uma cópia carregada anteriormente.

A saída de conversão de dados é executada em um ambiente como do programa que emitiu a chamada MQGET. Assim como aplicativos de usuário, o programa pode ser um MCA (agente do canal de mensagens) que está enviando mensagens a um gerenciador de filas de destino que não suporta a conversão de mensagens. O ambiente inclui espaço de endereço e perfil do usuário, conforme aplicável. A saída não pode comprometer a integridade do gerenciador de filas, porque ela não é executada no ambiente do gerenciador de filas.

Conversão de dados no z/OS

No z/OS, esteja ciente do seguinte:

- Programas de saída podem ser escritos somente na linguagem assembly.
- Programas de saída devem ser reentrantes e capazes de executar em qualquer lugar no armazenamento.
- Programas de saída devem restaurar o ambiente na saída para aquela entrada at e devem liberar qualquer armazenamento obtido.
- Programas de saída não devem efetuar WAIT ou emitir ESTAEs ou SPIEs.
- Programas de saída geralmente são chamados como se por z/OS LINK em:
  - Estado não autorizado do programa problemático
  - Modo de controle de espaço de endereço principal
  - Modo não de memória cruzada
  - Modo não de registro de acesso
  - Modo de endereçamento de 31 bits
  - Modo TCB-PRB

- Quando usada por um aplicativo CICS, a saída é chamada por EXEC CICS LINK e deve estar em conformidade com as convenções de programação do CICS. Os parâmetros são passados por ponteiros (endereços) na área de comunicação do CICS (COMMAREA).

  Embora não recomendado, os programas de saída de usuário também podem usar chamadas API do CICS, com o cuidado a seguir:

  – Não emita pontos de sincronização, pois os resultados podem influenciar unidades de trabalho declaradas pelo MCA.
  – Não atualize quaisquer recursos controlados por um gerenciador de recursos diferente do IBM MQ for z/OS, incluindo aqueles controlados pelo CICS Transaction Server.

  Para canais com CONVERT=YES, a saída é carregada do conjunto de dados referenciado pela instrução DD CSQXLIB. As saídas CSQCBDCI e CSQCBDCO fornecidas pelo MQ para a ponte IBM MQ CICS são SCSQAUTH.

Gravando um programa de saída de conversão de dados para o IBM i

Informações sobre as etapas a serem consideradas ao escrever programas de saída de conversão de dados do MQ para o IBM i.

Siga estas etapas:

1. Nomeie seu formato da mensagem. O nome deve se ajustar no campo *Format* do MQMD. O nome do *Format* não deve ter espaços em branco à esquerda integrados e os espaços em branco à direita são ignorados. O nome do objeto deve ter no máximo oito caracteres que não sejam espaços em branco, pois o *Format* tem somente oito caracteres de comprimento. Lembre-se de usar esse nome toda vez que enviar uma mensagem (o nosso exemplo usa o nome Formato).
2. Crie uma estrutura para representar a sua mensagem. Consulte Sintaxe válida para obter um exemplo.
3. Execute essa estrutura por meio do comando CVTMQMDTA para criar um fragmento de código para sua saída de conversão de dados.

   As funções geradas pelo comando CVTMQMDTA usam macros que são fornecidas no arquivo QMQM/H(AMQSVMHA). Essas macros são gravadas presumindo que todas as estruturas são compactadas; corrija-as se este não for o caso.
4. Faça uma cópia do arquivo de origem de estrutura básica fornecido, QMQMSAMP/QCSRC(AMQSVFC4) e renomeie-o. (Nosso exemplo usa o nome EXIT_MOD.)
5. Localize as caixas de comentário a seguir no arquivo de origem e insira código conforme descrito:

   a. Perto do fim do arquivo de origem, uma caixa de comentário inicia com:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Aqui, insira o fragmento de código gerado na etapa “3” na página 998.

   b. Perto do meio do arquivo de origem, uma caixa de comentário inicia com:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Isso é seguido por uma chamada comentada para a função `ConverttagSTRUCT`.

   Mude o nome da função para o nome da função incluída na etapa “5.a” na página 998. Remova os caracteres de comentário para ativar a função. Se houver várias funções, crie as chamadas para cada uma delas.

   c. Perto do início do arquivo de origem, uma caixa de comentário inicia com:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   Aqui, insira as instruções de protótipo de função para as funções incluídas na etapa “5.a” na página 998.

Se a mensagem contiver dados de caractere, o código gerado chamará MQXCNVC; isso pode ser resolvido ligando o programa de serviços QMQM/LIBMQM.

6. Compile o módulo de origem, EXIT_MOD, conforme a seguir:

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. Criar/vincular o programa.

   Para aplicativos não encadeados, use o seguinte:

```
CRTPGM PGM(library/Format)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

   Além de criar a saída de conversão de dados para o ambiente básico, outra é necessária no ambiente encadeado. Esse objeto carregável deve ser seguido por _R. Use a biblioteca LIBMQM_R para resolver as chamadas para o MQXCNVC. Ambos os objetos carregáveis são necessários para um ambiente encadeado.

```
CRTPGM PGM(library/Format_R)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM_R)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

8. Coloque a saída na lista de bibliotecas para a tarefa do IBM MQ. É recomendável que, para produção, os programas de saída de conversão de dados sejam armazenadas em QSYS.

Nota:

1. Se CVTMQMDTA usar estruturas compactadas, todos os aplicativos IBM MQ deverão usar o qualificador _Packed.
2. Os programas de saída de conversão de dados devem ser reentrantes.
3. MQXCNVC é a única chamada MQI que pode ser emitida por meio de uma saída de conversão de dados.
4. Compile o programa de saída com a opção do compilador do perfil do usuário configurado para *USER, de forma que a saída é executada com a autoridade do usuário.
5. A ativação da memória de teraspace é necessária para todas as saídas de usuário com o IBM MQ for IBM i; especifique TERASPACE(*YES *TSIFC) nos comandos CRTCMOD e CRTBNDC.

Writing a data-conversion exit program for IBM MQ for z/OS

Information about steps to consider when writing data-conversion exit programs for IBM MQ for z/OS.

Follow these steps:

1. Take the supplied source skeleton CSQ4BAX9 (for non-CICS environments) or CSQ4CAX9 (for CICS) as your starting point.
2. Run the CSQUCVX utility.
3. Follow the instructions in the prolog of CSQ4BAX9 or CSQ4CAX9 to incorporate the routines generated by the CSQUCVX utility, in the order that the structures occur in the message that you want to convert.
4. The utility assumes that the data structures are not packed, that the implied alignment of the data is honored, and that the structures start on a fullword boundary, with bytes being skipped as required (as between ID and VERSION in the example in Valid syntax). If the structures are packed, omit the CMQXCALA macros that are generated. Therefore, consider declaring your structures in such a way that all fields are named and no bytes are skipped; in the example in Valid syntax, add a field "MQBYTE DUMMY;" between ID and VERSION.

5. The supplied exit returns an error if the input buffer is shorter than the message format to be converted. Although the exit converts as many complete fields as possible, the error causes an unconverted message to be returned to the application. If you want to allow short input buffers to be converted as far as possible, including partial fields, change the TRUNC= value on the CSQXCDFA macro to YES: no error is returned, so the application receives a converted message. The application must handle the truncation.
6. Add any other special processing code that you need.
7. Rename the program to your data format name.
8. Compile and link-edit your program like a batch application program (unless it is for use with CICS applications). The macros in the code generated by the utility are in the library, **thlqual**.SCSQMACS.

   If the message contains character data, the generated code calls MQXCNVC. If your exit uses this call, link-edit it with the exit stub program CSQASTUB. The stub is language-independent and environment-independent. Alternatively, you can load the stub dynamically using the dynamic call name CSQXCNVC. See "Dynamically calling the IBM MQ stub" on page 1041 for more information.

   Place the link-edited module in your application load library, and in a data set that is referenced by the CSQXLIB DD statement of your task procedure started by your channel initiator.
9. If the exit is for use by CICS applications, compile and link-edit it like a CICS application program, including CSQASTUB if required. Place it in your CICS application program library. Define the program to CICS in the typical way, specifying EXECKEY(CICS) in the definition.

Note: Although the LE/370 runtime libraries are needed for running the CSQUCVX utility (see step "2" on page 999), they are not needed for link-editing or running the data-conversion exit itself (see steps "8" on page 1000 and "9" on page 1000).

See "Writing IMS bridge applications" on page 75 for information about data conversion within the IBM MQ - IMS bridge.

Linux AIX *Escrevendo uma saída de conversão de dados para sistemas IBM MQ for AIX or Linux*

Informações sobre etapas a serem consideradas ao escrever programas de saída de conversão de dados para sistemas IBM MQ for AIX or Linux.

Siga estas etapas:

1. Nomeie seu formato da mensagem. O nome deve se ajustar no campo *Format* do MQMD e estar em maiúsculas, por exemplo, MYFORMAT. O nome do *Format* não deve ter espaços em branco à esquerda. Espaços em branco à direita são ignorados. O nome do objeto deve ter no máximo oito caracteres que não sejam espaços em branco, pois o *Format* tem somente oito caracteres de comprimento. Lembre-se de usar esse nome toda vez que enviar uma mensagem.

   Se a saída de conversão de dados for usada em um ambiente encadeado, o objeto carregável deve ser seguido por _r para indicar que é uma versão encadeada.
2. Crie uma estrutura para representar a sua mensagem. Consulte Sintaxe válida para obter um exemplo.
3. Execute essa estrutura por meio do comando `crtmqcvx` para criar um fragmento de código para sua saída de conversão de dados.

   As funções geradas pelo comando `crtmqcvx` usam macros que assumem que todas as estruturas são compactadas; emende-as se este não for o caso.
4. Copie o arquivo de origem da estrutura básica fornecido, renomeando-o para o nome de seu formato de mensagem configurado na etapa "1" na página 1000. O arquivo de origem da estrutura básica e a cópia são somente leitura.

   O arquivo de origem da estrutura básica é chamado amqsvfc0.c.
5. No IBM MQ for AIX, um arquivo de exportação da estrutura básica chamado modelo amqsvfc.exp também é fornecido. Copie esse arquivo, renomeando-o para MYFORMAT.EXP.

6. A estrutura básica inclui um arquivo de cabeçalho de amostra, amqsvmha.h, no diretório *MQ_INSTALLATION_PATH*/inc, em que *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado. Certifique-se de que seu caminho de inclusão aponte para esse diretório para captar esse arquivo.

   O arquivo amqsvmha.h contém macros que são usadas pelo código gerado pelo comando `crtmqcvx`. Se a estrutura a ser convertida contiver dados de caractere, essas macros chamarão MQXCNVC.
7. Localize as caixas de comentário a seguir no arquivo de origem e insira código conforme descrito:
   a. Perto do fim do arquivo de origem, uma caixa de comentário inicia com:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      Aqui, insira o fragmento de código gerado na etapa "3" na página 1000.
   b. Perto do meio do arquivo de origem, uma caixa de comentário inicia com:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      Isso é seguido por uma chamada comentada para a função `ConverttagSTRUCT`.

      Mude o nome da função para o nome da função incluída na etapa "7.a" na página 1001. Remova os caracteres de comentário para ativar a função. Se houver várias funções, crie as chamadas para cada uma delas.
   c. Perto do início do arquivo de origem, uma caixa de comentário inicia com:

      ```
      /* Insert the function prototypes for the functions produced by */
      ```

      Aqui, insira as instruções de protótipo de função para as funções incluídas na etapa "3" na página 1000.
8. Compile a sua saída como uma biblioteca compartilhada, usando MQStart como o ponto de entrada. Para isso, consulte "Compilação das saídas de conversão de dados em sistemas AIX and Linux" na página 1001.
9. Coloque a saída no diretório de saída. O diretório de saída padrão é `/var/mqm/exits` para sistemas de 32 bits e `/var/mqm/exits64`, para sistemas de 64 bits. É possível mudar esses diretórios no arquivo qm.ini ou mqclient.ini. Esse caminho pode ser configurado para cada gerenciador de filas e a saída será procurada somente nesse caminho ou caminhos.

Nota:

1. Se `crtmqcvx` usar estruturas compactadas, todos os aplicativos IBM MQ deverão ser compilados dessa maneira.
2. Os programas de saída de conversão de dados devem ser reentrantes.
3. MQXCNVC é a única chamada MQI que pode ser emitida por meio de uma saída de conversão de dados.

Linux AIX *Compilação das saídas de conversão de dados em sistemas AIX and Linux*

Exemplos de como compilar uma saída de conversão de dados em sistemas AIX and Linux.

Em todas as plataformas, o ponto de entrada para o módulo é MQStart.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

AIX

AIX

Compile o código-fonte de saída emitindo um dos seguintes comandos:

Aplicativos de 32 bits

Não encadeado

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Encadeado

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Aplicativos de 64 bits

Não encadeado

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Encadeado

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

Compile o código-fonte de saída emitindo um dos seguintes comandos:

Aplicativos de 31 bits

Não encadeado

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

Encadeado

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Aplicativos de 32 bits

Não encadeado

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

Encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Aplicativos de 64 bits

Não encadeado

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Encadeado

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Windows *Gravando uma saída de conversão de dados para IBM MQ for Windows*

Informações sobre as etapas a serem consideradas ao gravar programas de saída de conversão de dados para o IBM MQ for Windows.

Siga estas etapas:

1. Nomeie seu formato da mensagem. O nome deve se ajustar no campo *Format* do MQMD. O nome do *Format* não deve ter espaços em branco à esquerda. Espaços em branco à direita são ignorados. O nome do objeto deve ter no máximo oito caracteres que não sejam espaços em branco, pois o *Format* tem somente oito caracteres de comprimento.

   Um arquivo .DEF chamado amqsvfcn.def também é fornecido no diretório de amostras, *MQ_INSTALLATION_PATH*\Tools\C\Samples *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado. Tire uma cópia desse arquivo e o renomeie, por exemplo, para MYFORMAT.DEF. Certifique-se de que o nome do DLL que está sendo criado e o nome especificado em MYFORMAT.DEF sejam iguais. Sobrescreva o nome FORMAT1 em MYFORMAT.DEF com o novo nome do formato.

   Lembre-se de usar esse nome toda vez que enviar uma mensagem.
2. Crie uma estrutura para representar a sua mensagem. Consulte Sintaxe válida para obter um exemplo.
3. Execute essa estrutura por meio do comando `crtmqcvx` para criar um fragmento de código para sua saída de conversão de dados.

   As funções geradas pelo comando CRTMQCVX usam macros que são gravadas presumindo que todas as estruturas são compactadas; conserte-as se este não for o caso.
4. Copie o arquivo de origem de estrutura básica fornecido, amqsvfc0.c, renomeando-o para o nome do formato de mensagem que você configurou na etapa “1” na página 1003.

   amqsvfc0.c está em *MQ_INSTALLATION_PATH*\Tools\C\Samples em que *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado. (O diretório de instalação padrão é C:\Program Files\IBM\MQ.)

   O esqueleto inclui um arquivo de cabeçalho de amostra amqsvmha.h no diretório *MQ_INSTALLATION_PATH*\Tools\C\include Certifique-se de que seu caminho de inclusão aponte para esse diretório para captar esse arquivo.

   O arquivo amqsvmha.h contém macros que são usadas pelo código gerado pelo comando CRTMQCVX. Se a estrutura a ser convertida contiver dados de caractere, essas macros chamarão MQXCNVC.
5. Localize as caixas de comentário a seguir no arquivo de origem e insira código conforme descrito:

   a. Perto do fim do arquivo de origem, uma caixa de comentário inicia com:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Aqui, insira o fragmento de código gerado na etapa “3” na página 1003.

b. Perto do meio do arquivo de origem, uma caixa de comentário inicia com:

```
/* Insert calls to the code fragments to convert the format's */
```

Isso é seguido por uma chamada comentada para a função ConverttagSTRUCT.

Mude o nome da função para o nome da função incluída na etapa “5.a” na página 1003. Remova os caracteres de comentário para ativar a função. Se houver várias funções, crie as chamadas para cada uma delas.

c. Perto do início do arquivo de origem, uma caixa de comentário inicia com:

```
/* Insert the function prototypes for the functions produced by */
```

Aqui, insira as instruções de protótipo de função para as funções incluídas na etapa “3” na página 1003.

6. Crie o arquivo de comando a seguir:

```
cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
MYFORMAT.C



MYFORMAT.DEF
```

em que *MQ_INSTALLATION_PATH* é o diretório no qual o IBM MQ está instalado.

7. Emita o arquivo de comando para compilar sua saída como um arquivo DLL.
8. Coloque a saída no subdiretório de saída abaixo do diretório de dados do IBM MQ. O diretório padrão para instalar suas saídas em sistemas de 32 bits é *MQ_DATA_PATH*\Exits e para sistemas de 64 bits é *MQ_DATA_PATH*\Exits64

O caminho usado para procurar pelas saídas de conversão de dados é fornecido no registro. A pasta de registro é:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

e a chave de registro é: ExitsDefaultPath. Esse caminho pode ser configurado para cada gerenciador de filas e a saída será procurada somente nesse caminho ou caminhos.

Nota:

1. Se CRTMQCVX usar estruturas compactadas, todos os aplicativos IBM MQ deverão ser compilados dessa maneira.
2. Os programas de saída de conversão de dados devem ser reentrantes.
3. MQXCNVC é a única chamada MQI que pode ser emitida por meio de uma saída de conversão de dados.

Windows *Arquivos de carregamento do comutador e saída em sistemas operacionais Windows*

Os processos do gerenciador de filas IBM WebSphere MQ for Windows 7.5 têm 32 bits. Como resultado, quando usar aplicativos 64-bit, alguns tipos de saída e arquivos de carregamento do comutador XA também precisa ter uma versão 32-bit disponível para uso pelo gerenciador de filas. Se a versão 32-bit da saída ou arquivo de carregamento do comutador XA for necessário e não estiver disponível, então o comando falhar ou chamada de API relevante.

Dois atributos são suportados no qm.ini file para *ExitPath*. Eles são ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits e ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64 O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado. Usá-los

assegura que a biblioteca apropriada pode ser localizada. Se uma saída for usada em um cluster do IBM MQ, isso também assegura que a biblioteca apropriada em um sistema remoto possa ser localizada.

A tabela a seguir lista os diferentes tipos de Saída e Comutador carregar arquivos e notas se 32-bit ou 64-bit versões, ou ambos, são necessários, de acordo com se 32-bit ou 64-bit aplicativos estão sendo utilizados:

Tipos de arquivo	Aplicativos de 32 bits	Aplicativos de 64 bits
saída API	32-bit e 64-bit	64 bits
Saída de conversão de dados	32 bits	64 bits
Saídas de canal do servidor (todos os tipos)	64 bits	64 bits
Saídas do Canal do Cliente (todos os tipos)	32 bits	64 bits
Saída de serviço instalável	64 bits	64 bits
Cluster do WLM de saída	64 bits	64 bits
Publicação/Assinatura de saída de roteamento	64 bits	64 bits
arquivos de carregamento do comutador do Banco	32-bit e 64-bit	64 bits
Bibliotecas AX do gerenciador de transações externo	32 bits	64 bits
Saída de pré-conexão	32 bits	64 bits

Fazendo referência a definições de conexão usando uma saída de pré-conexão de um repositório

O IBM MQ MQI clients pode ser configurado para consultar um repositório para obter definições de conexão usando uma biblioteca de saída de pré-conexão.

Introdução

Um aplicativo cliente pode se conectar a um gerenciador de filas usando tabelas de definição de canal do cliente (CCDT). Geralmente, o arquivo CCDT está localizado em um servidor de arquivos de rede central e possui clientes que fazem referência a ele. Como é difícil gerenciar e administrar vários aplicativos clientes que fazem referência ao arquivo CCDT, uma abordagem flexível é armazenar as definições do cliente em um repositório global como um diretório LDAP, um WebSphere Registry and Repository ou qualquer outro repositório. Armazenar as definições de conexão do cliente em um repositório facilita o gerenciamento das definições de conexão do cliente e os aplicativos podem acessar as definições de conexão do cliente corretas e mais atuais.

Durante a execução da chamada MQCONN/X, o IBM MQ MQI client carrega uma biblioteca de saída de pré-conexão especificada pelo aplicativo e chama uma função de saída para recuperar definições de conexão. As definições de conexão recuperadas são, então, usadas para estabelecer a conexão com um gerenciador de filas. Os detalhes da biblioteca de saída e da função a ser chamada são especificados no arquivo de configuração mqclient.ini.

Sintaxe

`void MQ_PRECONNECT_EXIT` (*pExitParms*, *pQMgrName*, *ppConnectOpts*, *pCompCode*, *pReason*);

Parâmetros

pExitParms
Tipo: entrada/saída PMQNXP

A estrutura do parâmetro de saída **PreConnection**.

A estrutura é alocada e mantida pelo responsável pela chamada da saída.

pQMgrName
Tipo: entrada/saída PMQCHAR

Nome do gerenciador de filas.

Na entrada, esse parâmetro é a cadeia de filtros fornecida para a chamada de API MQCONN por meio do parâmetro **QMgrName**. Esse campo pode ficar em branco, ser explícito ou conter determinados caracteres curingas. O campo é mudado pela saída. O parâmetro é NULL quando a saída é chamada com MQXR_TERM.

ppConnectOpts
Tipo: entrada/saída ppConnectOpts

Opções que controlam a ação de MQCONNX.

Esse é um ponteiro para uma estrutura de opções de conexão MQCNO que controla a ação da chamada de API MQCONN. O parâmetro é NULL quando a saída é chamada com MQXR_TERM. O cliente MQI sempre fornece uma estrutura MQCNO para a saída, mesmo que ela não tenha sido fornecida originalmente pelo aplicativo. Se um aplicativo fornecer uma estrutura MQCNO, o cliente fará uma duplicata para passá-la para a saída em que é modificado. O cliente retém a propriedade do MQCNO.

Um MQCD referenciado por meio do MQCNO tem precedência sobre qualquer definição de conexão fornecida por meio da matriz. O cliente usa a estrutura MQCNO para se conectar ao gerenciador de filas e os outros são ignorados.

pCompCode
Tipo: entrada/saída PMQLONG

Código de conclusão.

Ponteiro para um MQLONG que recebe o código de conclusão de saídas. Deve ser um dos valores a seguir:

- MQCC_OK - Conclusão bem-sucedida
- MQCC_WARNING - Aviso (conclusão parcial)
- MQCC_FAILED - Falha na chamada

pReason
Tipo: entrada/saída PMQLONG

Razão que qualifica pCompCode.

Ponteiro para um MQLONG que recebe o código de razão de saída. Se o código de conclusão for MQCC_OK, o único valor válido será:

- MQRC_NONE - (0, x'000') Nenhuma razão para o relatório.

Se o código de conclusão for MQCC_FAILED ou MQCC_WARNING, a função de saída poderá configurar o campo de código de razão como qualquer valor MQRC_* válido.

Chamada C

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

Escrevendo e compilando saídas de publicação

É possível configurar uma saída de publicação no gerenciador de filas para mudar o conteúdo de uma mensagem publicada antes que seja recebida pelos assinantes. Também é possível mudar o cabeçalho da mensagem ou não entregar a mensagem a uma assinatura.

Nota: Saídas de publicação não são suportadas no z/OS.

É possível usar a saída de publicação para inspecionar e alterar mensagens entregues aos assinantes:

- Examine o conteúdo de uma mensagem publicada para cada assinante
- Modifique os conteúdos de uma mensagem publicada para cada assinante
- Altere a fila na qual uma mensagem é colocada
- Pare a entrega de uma mensagem para um assinante

Escrevendo uma saída de publicação

Use as etapas em "Composição de saídas e serviços instaláveis em AIX, Linux, and Windows" na página 948, para ajudá-lo a escrever e compilar sua saída.

O provedor da saída de publicação define o que a saída faz. A saída, no entanto, deve estar em conformidade com as regras definidas em MQPSXP.

O IBM MQ não fornece uma implementação do ponto de entrada MQ_PUBLISH_EXIT. Ele fornece uma declaração typedef em linguagem C. Use o typedef para declarar os parâmetros para uma saída escrita pelo usuário corretamente. O exemplo a seguir ilustra como usar a declaração typedef:

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

A saída de publicação é executada no processo do gerenciador de filas, como resultado das operações a seguir:

- Uma operação de Publicação em que uma mensagem é entregue a um ou mais assinantes
- Uma operação de Publicação em que uma ou mais mensagens retidas são entregues
- Uma operação de Solicitação de assinatura em que uma ou mais mensagens retidas são entregues

Se a saída de publicação for chamada para uma conexão, na primeira vez em que for chamado, um código *ExitReason* de MQXR_INIT será configurado. Antes que a conexão seja desconectada depois de usar uma saída de publicação, a saída será chamada com um código *ExitReason* de MQXR_TERM.

Se a saída de publicação for configurada, mas não puder ser carregada quando o gerenciador de filas for iniciado, as operações de publicar/assinar mensagens serão inibidas para o gerenciador de filas. Deve-se corrigir o problema ou reiniciar o gerenciador de filas antes que o sistema de mensagens de publicar/assinar seja reativado.

Cada conexão do IBM MQ que requer a saída de publicação poderá falhar ao carregar ou inicializar a saída. Se a saída falhar ao carregar ou inicializar, as operações de publicar/assinar que requerem a saída de publicação serão desativadas para essa conexão. As operações falham com o código de razão do IBM MQ `MQRC_PUBLISH_EXIT_ERROR`.

O contexto no qual a saída de publicação é chamada é a conexão por um aplicativo ao gerenciador de filas. Uma área de dados do usuário é mantida pelo gerenciador de filas para cada conexão que está executando operações de publicação. A saída pode reter informações na área de dados do usuário para cada conexão.

Uma saída de publicação pode usar algumas chamadas MQI. Ela pode usar somente as chamadas MQI que manipulam as propriedades de mensagens. As chamadas são:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Se a saída de publicação mudar o gerenciador de filas de destino ou o nome da fila, nenhuma nova verificação de autoridade será executada.

Compilando uma saída de publicação

A saída de publicação é uma biblioteca dinamicamente carregada; ela pode ser considerada como uma saída de canal. Para obter informações sobre a compilação das saídas, consulte "Composição de saídas e serviços instaláveis em AIX, Linux, and Windows" na página 948.

Saída de publicação de amostra

O programa de saída de amostra é chamado `amqspse0.c`. Ele grava uma mensagem diferente em um arquivo de log dependendo de se a saída foi chamada para inicializar, publicar ou finalizar as operações. Também demonstra o uso do campo da área do usuário de saída para alocar e liberar armazenamento de forma apropriada.

Configurando saídas de publicação

Deve-se configurar certos atributos para configurar uma saída de publicação.

No Windows e no Linux, é possível usar o IBM MQ Explorer para definir os atributos. Os atributos são definidos na página de propriedades do gerenciador de filas, em Publicar/Assinar.

Para configurar a saída de publicação no arquivo `qm.ini` em sistemas AIX and Linux, crie uma sub-rotina chamada `PublishSubscribe`. A sub-rotina `PublishSubscribe` tem os atributos a seguir:

PublishExitPath=[path]|*module_name*
Nome e caminho do módulo que contém o código de saída da publicação. O comprimento máximo desse campo é `MQ_EXIT_NAME_LENGTH`. O padrão é nenhuma saída da publicação.

= PublishExitFunction= *function_name*
Nome do ponto de entrada da função no módulo que contém o código de saída de publicação. O comprimento máximo desse campo é `MQ_EXIT_NAME_LENGTH`.

IBM i No IBM i, se um programa for usado, omita `PublishExitFunction`.

PublishExitData= *string*
Se o gerenciador de filas estiver chamando uma saída de publicação, ele passará uma estrutura `MQPSXP` como entrada. Os dados especificados usando o atributo **PublishExitData** são fornecidos no campo *ExitData* da estrutura. A sequência pode ter até `MQ_EXIT_DATA_LENGTH` caracteres de comprimento. O padrão é de 32 caracteres em branco.

Gravando e Compilando Saídas de Carga de Trabalho do Cluster

Escreva um programa de saída de carga de trabalho do cluster para customizar o gerenciamento de carga de trabalho de clusters. Você pode levar em consideração o custo de usar um canal em diferentes horários do dia ou o conteúdo da mensagem ao rotear as mensagens. Esses são fatores que não são considerados pelo algoritmo de gerenciamento de carga de trabalho padrão.

Na maioria dos casos, o algoritmo de gerenciamento de carga de trabalho é suficiente para suas necessidades. No entanto, para que você possa fornecer seu próprio programa de saída de usuário para customizar o gerenciamento de carga de trabalho, o IBM MQ inclui uma saída de usuário, a saída de carga de trabalho do cluster.

Pode haver alguma informação específica sobre sua rede ou mensagens que poderia usar para influenciar o balanceamento de carga de trabalho. Você pode saber quais são os canais de alta capacidade ou as rotas de rede baratas ou pode desejar rotear as mensagens dependendo de seu conteúdo. Poderia decidir escrever um programa de saída de carga de trabalho de cluster ou usar um fornecido por um terceiro.

A saída de carga de trabalho do cluster é chamada ao acessar uma fila de clusters. Ela é chamada por `MQOPEN`, `MQPUT1` e `MQPUT`.

O gerenciador de filas de destino selecionado no momento de `MQOPEN` será fixado se `MQOO_BIND_ON_OPEN` for especificado. Nesse caso, a saída será executada somente uma vez.

Se o gerenciador de filas de destino não for fixado no momento de `MQOPEN`, o gerenciador de filas de destino será escolhido no momento da chamada `MQPUT`. Se o gerenciador de filas de destino não estiver disponível ou falhar enquanto a mensagem ainda estiver na fila de transmissão, a saída será chamada novamente. Um novo gerenciador de filas de destino será selecionado. Se o canal de mensagens falhar enquanto a mensagem está sendo transferida e a mensagem for restaurada, um novo gerenciador de filas de destino será selecionado.

Multi No Multiplataformas, o gerenciador de filas carregará a nova saída de carga de trabalho do cluster na próxima vez que o gerenciador de filas for iniciado.

Se a definição do gerenciador de filas não contiver um nome de programa de saída de carga de trabalho do cluster, a saída de carga de trabalho do cluster não será chamada.

Vários dados são passados para uma saída de carga de trabalho do cluster na estrutura do parâmetro de saída, `MQWXP`:

- A estrutura de definição de mensagem, `MQMD`.
- O parâmetro de comprimento da mensagem.
- Uma cópia da mensagem ou parte da mensagem.

Em plataformas não z/OS, se você usar `CLWLMode=FAST`, cada processo do sistema operacional carregará sua própria cópia da saída. Diferentes conexões no gerenciador de filas podem fazer com que diferentes cópias da saída sejam chamadas. Se a saída for executada no modo de segurança padrão, `CLWLMode=SAFE`, uma única cópia da saída será executada em seu próprio processo separado.

Escrevendo saídas de carga de trabalho do cluster

z/OS Para obter informações sobre como escrever saídas de carga de trabalho do cluster para o z/OS, consulte “Cluster workload exit programming for IBM MQ for z/OS” na página 1011.

Na IBM MQ 9.1.0, as saídas de carga de trabalho do cluster são executadas no espaço de endereço do inicializador de canais, em vez de no espaço de endereço do gerenciador de filas. Se você tiver uma saída de carga de trabalho do cluster, será necessário remover a instrução CSQXLIB DD do procedimento de tarefa iniciada do gerenciador de filas e incluir o conjunto de dados que contém a saída de carga de trabalho do cluster na concatenação CSQXLIB no procedimento de tarefa iniciada do inicializador de canais.

Multi Para multiplataformas, as saídas de carga de trabalho do cluster não devem usar chamadas MQI. Em outros aspectos, as regras para escrever e compilar programas de saída de carga de trabalho

do cluster são semelhantes às regras que se aplicam aos programas de saída do canal. Siga as etapas em “Composição de saídas e serviços instaláveis em AIX, Linux, and Windows” na página 948 e use o programa de amostra, “Saída de carga de trabalho do cluster de amostra” na página 1010, para ajudar a escrever e compilar sua saída.

Para obter mais informações sobre saídas do canal, consulte “Gravando programas de saída do canal” na página 976.

Configurando saídas de carga de trabalho do cluster

Você denomina as saídas de carga de trabalho do cluster na definição do gerenciador de filas especificando o atributo de saída de carga de trabalho do cluster no comando ALTER QMGR. Por exemplo:

```
ALTER QMGR CLWLEXIT(myexit)
```

Referências relacionadas

Chamada de Saída de Carga de Trabalho do Cluster e Estruturas de Dados

Saída de carga de trabalho do cluster de amostra

O IBM MQ inclui um programa de saída de carga de trabalho do cluster de amostra. É possível copiar a amostra e usá-la como base para seus próprios programas.

z/OS **IBM MQ for z/OS**

O programa de saída de carga de trabalho do cluster de amostra é fornecido em Assembler e em C A versão do Assembler é chamada CSQ4BAF1 e pode ser localizada na biblioteca `thlqual.SCSQASMS`. A versão C é chamada de CSQ4BCF1 e pode ser localizada na biblioteca `thlqual.SCSQC37S`. `thlqual` é o qualificador de alto nível da biblioteca de destino para conjuntos de dados do IBM MQ em sua instalação.

Multi **IBM MQ for Multiplatforms**

O programa de saída de carga de trabalho do cluster de amostra é fornecido em C e é chamado `amqswlm0.c`. Ele pode ser localizado em:

Tabela 144. Local do programa de saída de carga de trabalho do cluster de amostra para multiplataformas

Plataforma	**Caminho de arquivo..**
AIX AIX	*MQ_INSTALLATION_PATH*`/samp`
Windows Windows	*MQ_INSTALLATION_PATH* `\Tools\c\Samples`
IBM i IBM i	A biblioteca `qmqm`

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Esta saída da amostra roteia todas as mensagens para um gerenciador de filas específico, a menos que o gerenciador de filas se torne indisponível. Ela reage contra a falha do gerenciador de filas ao rotear mensagens para outro gerenciador de filas.

Indique qual gerenciador de filas ao qual você deseja que as mensagens sejam enviadas. Forneça o nome do canal do receptor de clusters no atributo CLWLDATA na definição do gerenciador de filas. Por exemplo:

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Para ativar a saída, forneça seu caminho completo e nome no atributo CLWLEXIT:

Linux AIX No AIX and Linux:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows No Windows:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS No z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

em que x é um `'A'` ou `'C'`, dependendo da linguagem de programação da versão que você está usando.

IBM i No IBM i, utilize um dos seguintes comandos:

- Use o comando MQSC:

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  Tanto o nome do programa quanto o nome da biblioteca ocupam 10 caracteres e são preenchidos em branco à direita se necessário.
- Use o comando CL:

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

Agora, em vez de usar o algoritmo de gerenciamento de carga de trabalho fornecido, IBM MQ chama esta saída para rotear todas as mensagens para seu gerenciador de filas escolhido.

z/OS *Cluster workload exit programming for IBM MQ for z/OS*

Cluster workload exits are invoked as if by a z/OS **LINK** command. Exits are subject to a number of stringent programming rules. Avoid using most SVC commands that involve waits, or using a STAE or ESTAE in a workload exit.

Cluster workload exits are invoked as if by a z/OS **LINK** in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31 bit addressing mode
- Storage key 8
- Program Key Mask 8
- TCB key 8

Put the link-edited modules in the data set specified by the `CSQXLIB DD` statement of the started task procedure of the channel initiator. The names of the load modules are specified as the workload exit names in the queue manager definition.

When writing workload exits for IBM MQ for z/OS, the following rules apply:

- You must write exits in assembler or C. If you use C, it must conform to the C systems programming environment for system exits, described in the *z/OS C/C++ Programming Guide*, SC09-4765.
- If using the `MQXCLWLN` call, link edit with `CSQMFCLW`, supplied in *`thlqual`*`.SCSQLOAD`.

- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the queue manager is running, with the new version used in the next MQCONN thread the queue manager starts.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment on return to that at entry.
- Exits must free any storage obtained, or ensure that storage is freed by a subsequent exit invocation.
- No MQI calls are allowed.
- Exits must not use any system services that could cause a wait, because a wait severely degrades the performance of the queue manager. In general, therefore, avoid an SVC, PC, or I/O.
- Exits must not issue an ESTAE or SPIE, apart from within any subtasks they attach.

Note: There are no absolute restrictions on what you can do in an exit. However, most SVCs involve waits, so avoid them, except for the following commands:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Do not use ESTAEs and ESPIEs because their error handling might interfere with the error handling performed by IBM MQ. IBM MQ might not be able to recover from an error, or your exit program might not receive all the error information.

The system parameter EXITLIM limits the amount of time an exit might run for. The default value for EXITLIM is 30 seconds. If you see the return code MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' your exit might be looping. If you think the exit needs more than 30 seconds to complete, increase the value of EXITLIM.

Construindo um aplicativo processual

É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

AIX Construindo seu aplicativo processual no AIX

As publicações AIX descrevem como construir aplicativos executáveis a partir dos programas que você escreve.

Este tópico descreve as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao criar aplicativos do IBM MQ for AIX para executar no AIX. C, C++ e COBOL são suportados. Para obter informações sobre a preparação de seus programas C++, consulte Usando C++.

As tarefas que devem ser executadas para criar um aplicativo executável usando o IBM MQ for AIX variam com a linguagem de programação na qual seu código de origem é escrito. Além de codificar as chamadas do MQI em seu código fonte, deve-se incluir as instruções de linguagem apropriadas para incluir os arquivos de inclusão do IBM MQ for AIX para a linguagem que você está usando. Familiarize-se com o conteúdo desses arquivos. Consulte “Arquivos de definição de dados doIBM MQ” na página 729 para obter uma descrição completa.

Ao executar aplicativos cliente ou servidor encadeados, configure a variável de ambiente AIXTHREAD_SCOPE=S

AIX *Preparando programas C no AIX*

Este tópico contém informações sobre a vinculação de bibliotecas necessárias para preparar programas C no AIX.

Os programas C pré-compilados são fornecidos no diretório *MQ_INSTALLATION_PATH*/samp/bin. Use o compilador ANSI e execute os comandos a seguir. Para obter informações adicionais sobre a programação de aplicativos de 64 bits, veja Padrões de codificação em plataformas de 64 bits.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para aplicativos de 32 bits:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

em que `amqsput0` é um programa de amostra.

Para aplicativos de 64 bits:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

em que `amqsput0` é um programa de amostra.

V 9.4.0 Para aplicativos de 32 bits usando o compilador XLC 17:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

em que `amqsput0` é um programa de amostra.

V 9.4.0 Para aplicativos de 64 bits usando o compilador XLC 17:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

em que `amqsput0` é um programa de amostra.

Se você estiver usando o compilador C/C++ do VisualAge para programas C++, a opção `-q namemangling=v5` deverá ser incluída para obter todos os símbolos do IBM MQ resolvidos ao vincular as bibliotecas.

Se desejar usar os programas em uma máquina que tem apenas o IBM MQ MQI client for AIX instalado, recompile os programas para vinculá-los à biblioteca do cliente (`-lmqic`) em vez disso.

Vinculando bibliotecas

São necessárias as seguintes bibliotecas:

- Vincule seus programas à biblioteca apropriada fornecida pelo IBM MQ.

  Em um ambiente não encadeado, vincule-se a uma das seguintes bibliotecas:

Arquivo de biblioteca	Tipo de programa/saída
libmqm.a	Servidor para C
libmqic.a & libmqm.a	Cliente para C

  Em um ambiente encadeado, vincule-se a uma das seguintes bibliotecas:

Arquivo de biblioteca	Tipo de programa/saída
libmqm_r.a	Servidor para C
libmqic_r.a & libmqm_r.a	Cliente para C

  Por exemplo, para construir um aplicativo IBM MQ encadeado simples a partir de uma única unidade de compilação, execute os comandos a seguir.

  Para aplicativos de 32 bits:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

  em que `amqsput0` é um programa de amostra.

Para aplicativos de 64 bits:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

em que `amqsput0` é um programa de amostra.

V 9.4.0 Para aplicativos de 32 bits usando o compilador XLC 17:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

em que `amqsput0` é um programa de amostra.

V 9.4.0 Para aplicativos de 64 bits usando o compilador XLC 17:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

em que `amqsput0` é um programa de amostra.

Se desejar usar os programas em uma máquina que tem apenas o IBM MQ MQI client for AIX instalado, recompile os programas para vinculá-los à biblioteca do cliente (`-lmqic`) em vez disso.

Nota:

1. Não é possível se vincular a mais de uma biblioteca. Ou seja, não é possível se vincular a uma biblioteca encadeada e a uma não encadeada ao mesmo tempo.
2. Se você estiver gravando um serviço instalável (consulte o Administrando IBM MQ para obter mais informações), será necessário vincular-se à biblioteca `libmqmzf.a` em um aplicativo não encadeado e à biblioteca `libmqmzf_r.a` em um aplicativo encadeado.
3. Se você estiver produzindo um aplicativo para coordenação externa por um gerenciador de transações compatível com XA, como IBM TXSeries, Encina ou BEA Tuxedo, será necessário vincular-se ao `libmqmxa.a` (ou `libmqmxa64.a` se o gerenciador de transações tratar o tipo 'long' como 64 bits) e às bibliotecas `libmqz.a` em um aplicativo não encadeado e ao `libmqmxa_r.a` (ou `libmqmxa64_r.a`) e às bibliotecas `libmqz_r.a` em um aplicativo encadeado.
4. É necessário vincular aplicativos confiáveis para as bibliotecas encadeadas do IBM MQ. No entanto, apenas um encadeamento em um aplicativo confiável em sistemas IBM MQ for AIX or Linux pode ser conectado de cada vez.
5. Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

AIX *Preparando programas COBOL em AIX*

Use estas informações ao preparar programas COBOL no AIX usando o IBM COBOL Set e o Micro Focus COBOL.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

- Copy books COBOL de 32 bits são instalados no seguinte diretório:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy32
  ```

  e links simbólicos são criados em:

  ```
  MQ_INSTALLATION_PATH/inc
  ```

- Copy books COBOL de 64 bits são instalados no diretório a seguir:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

Nos exemplos a seguir configure a variável de ambiente **COBCPY** para:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

para aplicativos de 32 bits e:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

para aplicativos de 64 bits.

É necessário vincular seu programa a um dos arquivos de biblioteca a seguir:

Arquivo de biblioteca	Tipo de programa/saída
libmqmcb.a	Servidor para COBOL (aplicativo não encadeado)
libmqmcb_r.a	Servidor para COBOL (aplicativo encadeado)
libmqicb.a	Cliente para COBOL (aplicativo não encadeado)
libmqicb_r.a	Cliente para COBOL (aplicativo encadeado)

É possível usar o compilador IBM COBOL Set ou o compilador Micro Focus COBOL, dependendo do programa:

- Programas iniciados por `amqm` são adequados para o compilador Micro Focus COBOL e
- Programas iniciados por `amq0` são adequados para ambos os compiladores.

Preparando programas COBOL usando o IBM COBOL Set for AIX

Programas COBOL de amostra são fornecidos com o IBM MQ. Para compilar esse programa, insira o comando apropriado na lista a seguir:

Aplicativo do servidor não encadeado de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

Aplicativo cliente não encadeado de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

Aplicativo do servidor encadeado de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Aplicativo cliente encadeado de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Aplicativo do servidor não encadeado de 64 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

Aplicativo cliente não encadeado de 64 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

Aplicativo do servidor encadeado de 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Aplicativo cliente encadeado de 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Preparando programas COBOL usando o Micro Focus COBOL

Configure as variáveis de ambiente antes de compilar seu programa da seguinte forma:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Para compilar um programa COBOL de 32 bits usando o Micro Focus COBOL, insira:

- Servidor para COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- Cliente para COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- Servidor encadeado para COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Cliente encadeado para COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Para compilar um programa COBOL de 64 bits usando o Micro Focus COBOL, insira:

- Servidor para COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- Cliente para COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- Servidor encadeado para COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Cliente encadeado para COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

em que amqminqx é o programa de amostra

Consulte a documentação do Micro Focus COBOL para obter uma descrição das variáveis de ambiente que precisa configurar.

AIX *Preparando programas de aplicativo CICS no AIX*

Use estas informações ao preparar programas CICS no AIX.

Use módulos do *comutador XA* para vincular CICS com o IBM MQ. Para obter mais informações sobre a estrutura do comutador XA, consulte As estruturas do comutador XA.

O arquivo de código-fonte de amostra é fornecido para permitir que você desenvolva os comutadores XA para outras mensagens de transação. O nome do módulo de carregamento do comutador fornecido está listado emTabela 145 na página 1017.

Tabela 145. Código essencial para programas de aplicativo do CICS no AIX: rotina de inicialização de XA

Descrição	**C (origem)**	**C (exec)-incluir em seu XAD.Stanza**
Rotina de inicialização de XA	amqzscix.c	*amqzsc* - CICS para AIX

Use a versão pré-construída do arquivo de carregamento do comutador do IBM MQ *amqzsc*, que é fornecido com o produto.

Sempre vincule suas transações C à IBM MQ biblioteca *libmqm_r.athread-safe.*, e suas transações COBOL com a biblioteca COBOL *libmqmcb_r.a.*

É possível localizar mais informações sobre o suporte a transações do CICS no Guia de administração do sistema Administrando IBM MQ IBM MQ.

AIX *Suporte ao TXSeries CICS*

IBM MQ no AIX suporta TXSeries CICS usando a interface XA. Certifique-se de que os aplicativos CICS estejam vinculados à versão encadeada das bibliotecas do IBM MQ.

É possível executar programas CICS usando o IBM COBOL Set para o AIX ou Micro Focus COBOL. As seções ações descrevem as diferenças entre a execução de programas do CICS no IBM COBOL Set para o AIX e Micro Focus COBOL.

Grave programas IBM MQ carregados na mesma região do CICS em C ou COBOL. Não é possível fazer uma combinação de chamadas MQI em C e COBOL na mesma região do CICS. A maioria das chamadas MQI na segunda linguagem usada falha com um código de razão de MQRC_HOBJ_ERROR.

Preparando programas CICS COBOL usando o IBM COBOL Set for AIX

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para usar o IBM COBOL, siga estas etapas:

1. Exporte a variável de ambiente a seguir:

   ```
   export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                   -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                   -e _iwz_cobol_main  \
   ```

   em que LIB é uma diretriz do compilador.
2. Converta, compile e vincule o programa digitando:

   ```
   cicstcl -l IBMCOB yourprog.ccp
   ```

Preparando programas CICS COBOL usando Micro Focus COBOL

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para usar Micro Focus COBOL, siga estas etapas:

1. Inclua o módulo de biblioteca de tempo de execução do IBM MQ COBOL na biblioteca de tempo de execução usando o comando a seguir:

   ```
   cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Nota: Com `cicsmkcobol`, o IBM MQ não permite fazer chamadas MQI na linguagem de programação C a partir de seu aplicativo em COBOL.

   Se seus aplicativos existentes tiverem quaisquer chamadas desse tipo, será recomendado que você mova estas funções dos aplicativos COBOL para sua própria biblioteca, por exemplo, `myMQ.so`. Depois de mover as funções, não inclua a biblioteca do IBM MQ `libmqmcbrt.o` ao construir o aplicativo COBOL para CICS.

   Além disso, se seu aplicativo COBOL não fizer nenhuma chamada MQI COBOL, não vincule `libmqmz_r` com `cicsmkcobol`.

   Isso cria o arquivo de método de linguagem Micro Focus COBOL e possibilita que a biblioteca COBOL de tempo de execução do CICS chame os sistemas IBM MQ for AIX or Linux.

   Nota: Execute `cicsmkcobol` somente ao instalar um dos produtos a seguir:

   - Nova versão ou liberação do Micro Focus COBOL
   - Nova versão ou liberação do CICS para AIX
   - Nova versão ou liberação de qualquer produto de banco de dados suportado (apenas para transações COBOL)
   - Nova versão ou liberação do IBM MQ
2. Exporte a variável de ambiente a seguir:

   ```
   COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
   ```
3. Converta, compile e vincule o programa digitando:

   ```
   cicstcl -l COBOL -e yourprog.ccp
   ```

Preparando programas CICS C

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Construir programas CICS C usando os recursos padrão do CICS:

1. Exportar ***uma*** das variáveis de ambiente a seguir:
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" exportar LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" exportar USERLIB
2. Converta, compile e vincule o programa digitando:

   ```
   cicstcl -l C amqscic0.ccs
   ```

Transação de amostra do CICS C

A origem de C de amostra para uma transação AIX IBM MQ é fornecida por AMQSCIC0.CCS. A transação lê mensagens da fila de transmissão SYSTEM.SAMPLE.CICS.WORKQUEUE no gerenciador de filas padrão e coloca-os na fila local com um nome de fila que está contido no cabeçalho de

transmissão da mensagem Quaisquer falhas são enviadas para a fila SYSTEM.SAMPLE.CICS.DLQ. Use o script de MQSC de amostra AMQSCIC0.TST para criar estas filas e filas de entrada de amostra.

IBM i Construindo seu aplicativo processual no IBM i

As publicações do IBM i descrevem como construir aplicativos executáveis a partir dos programas gravados para serem executados com IBM i em sistemas iSeries ou System i.

Este tópico descreve as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao criar aplicativos processuais do IBM MQ for IBM i para executar nos sistemas IBM i. COBOL, C, C++, Java e linguagens de programação RPG são suportadas. Para obter informações sobre a preparação de seus programas C++, consulte Usando C++. Para obter informações sobre a preparação de seus programas Java, consulte Usando IBM MQ classes for Java.

As tarefas que se deve executar para criar um aplicativo executável do IBM MQ for IBM i dependem da linguagem de programação em que o código fonte é escrito. Além de codificar as chamadas MQI em seu código fonte, deve-se incluir as instruções de linguagem apropriadas para incluir os arquivos de definição de dados do IBM MQ for IBM i para a linguagem que você estiver usando. Familiarize-se com o conteúdo desses arquivos. Consulte “Arquivos de definição de dados doIBM MQ” na página 729 para obter uma descrição completa.

IBM i *Preparando programas C no IBM i*

O IBM MQ for IBM i suporta mensagens de até 100 MB de tamanho. Os programas aplicativos escritos em ILE C, suportando mensagens IBM MQ maiores que 16 MB, precisarão usar a opção de compilador `Teraspace` para alocar memória suficiente para essas mensagens.

Para obter mais informações sobre as opções de compilador C, consulte o *WebSphere Guia do programador do Development Studio ILE C/C++*.

Para compilar um módulo C, é possível utilizar o IBM i comando **CRTCMOD** Certifique-se de que a biblioteca que contém os arquivos de inclusão (QMQM) esteja na lista de bibliotecas quando você compilar.

Em seguida, deve-se ligar a saída do compilador ao programa de serviço usando o comando **CRTPGM**.

Tabela 146. Exemplo de CRTPGM em ambientes não encadeados e encadeados

Tipo de ambiente	Comando:	Tipo de programa/saída

Ambiente não encadeado	`CRTPGM PGM( pgmname )`
	`MODULE( pgmname )`
	`BNDSRVPGM(QMQM/LIBMQM)`
		Servidor ou cliente para C

Ambiente encadeado	`CRTPGM PGM( pgmname )`
	`MODULE( pgmname )`
	`BNDSRVPGM(QMQM/LIBMQM_R)`
		Servidor ou cliente para C

em que *pgmname* é o nome do seu programa.

O Tabela 147 na página 1019 lista as bibliotecas que são necessárias ao preparar programas C no IBM i em um ambiente não encadeado e ambiente encadeado.

Tabela 147. Bibliotecas necessárias para ambientes não encadeados e encadeados

Tipo de ambiente	Arquivo de biblioteca	Tipo de programa/saída
Ambiente não encadeado	LIBMQM	Servidor para C
	LIBMQIC & LIBMQM	Cliente para C

Tabela 147. Bibliotecas necessárias para ambientes não encadeados e encadeados (continuação)		
Tipo de ambiente	**Arquivo de biblioteca**	**Tipo de programa/saída**
Ambiente encadeado	LIBMQM_R	Servidor para C
	LIBMQIC_R & LIBMQM_R	Cliente para C

IBM i *Preparando programas COBOL em IBM i*

Aprenda sobre a preparação de programas COBOL no IBM i e sobre o método de acesso ao MQI por meio do programa COBOL.

Sobre esta tarefa

Para acessar o MQI por meio de programas COBOL, o IBM MQ for IBM i fornece uma interface de chamada processual ligada fornecida pelos programas de serviço. Isso fornece acesso a todas as funções MQI no IBM MQ for IBM i e suporte para aplicativos encadeados. Essa interface pode ser usada somente com o compilador ILE COBOL.

A sintaxe COBOL CALL padrão é usada para acessar as funções do MQI.

Os arquivos de cópia de COBOL que contêm as constantes denominadas e as definições de estrutura para uso com a MQI estão contidos no arquivo físico de origem QMQM/QCBLLESRC.

Os arquivos de cópia de COBOL usam o caractere de aspas simples (') como o delimitador de sequência. Os compiladores COBOL IBM i supõem que o delimitador sejam as aspas ("). Para evitar que os compiladores gerem mensagens de aviso, especifique `OPTION(*APOST)` nos comandos **CRTCBLPGM**, **CRTBNDCBL** ou **CRTCBLMOD**.

Para fazer o compilador aceitar o caractere de aspas simples (') como o delimitador de sequência nos arquivos de cópia de COBOL, use a opção do compilador \APOST.

Nota: A interface de chamada dinâmica não é fornecida no IBM MQ 9.0 ou mais recente.

Para usar a interface de chamada de procedimento de ligação, conclua as etapas a seguir.

Procedimento

1. Crie um módulo usando o compilador **CRTCBLMOD** especificando o parâmetro:

   ```
   LINKLIT(*PRC)
   ```

2. Use o comando **CRTPGM** para criar o objeto de programa, especificando o parâmetro apropriado:

   Para aplicativos não encadeados:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
   ```

   Para aplicativos encadeados:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
   ```

   Nota: Exceto para programas criados usando o compilador ILE COBOL V4R4 e que contêm a opção THREAD(SERIALIZE) na instrução PROCESS, os programas COBOL não devem usar as bibliotecas encadeadas do IBM MQ. Mesmo se um programa COBOL tiver se tornado thread-safe dessa maneira, tome cuidado ao projetar o aplicativo, porque THREAD(SERIALIZE) força a serialização de procedimentos COBOL no nível do módulo e isso pode afetar o desempenho geral.

   Consulte o *WebSphere Development Studio: Guia do programador do ILE COBOL* e o *WebSphere Development Studio: Referência do ILE COBOL* para obter informações adicionais.

Para obter mais informações sobre como compilar um aplicativo CICS, consulte o *CICS for IBM i Application Programming Guide*, SC41-5454.

IBM i *Preparando programas CICS no IBM i*

Aprenda sobre as etapas necessárias ao preparar programas CICS no IBM i.

Para criar um programa que inclui instruções EXEC CICS e chamadas MQI, execute estas etapas:

1. Se necessário, prepare mapas usando o comando CRT.CICSMAP.
2. Traduza os comandos EXEC CICS em instruções de idioma nativo. Use o comando CRTCICSC para um programa C. Us o comando CRTCICSCBL para um programa COBOL.

   Inclua `CICSOPT(*NOGEN)` no comando CRTCICSC ou CRTCICSCBL. Isso para o processamento para permitir a inclusão dos programas de serviço CICS e IBM MQ apropriados. Esse comando coloca o código, por padrão, em QTEMP/QACYCICS.
3. Compile o código-fonte usando o comando CRTCMOD (para um programa C) ou o comando CRTCBLMOD (para um programa COBOL).
4. Use CRTPGM para vincular o código compilado aos programas de serviço CICS e IBM MQ apropriados. Isso cria o programa executável.

Um exemplo desse código segue (ele compila o programa de amostra do CICS enviado):

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i *Preparando programas RPG no IBM i*

Se estiver usando o IBM MQ for IBM i, é possível escrever seus aplicativos em RPG.

Para obter mais informações, veja "Codificando programas IBM MQ em RPG (IBM i somente)" na página 1069 e consulte a Referência de programação de aplicativos do IBM i (ILE/RPG).

IBM i *Considerações sobre programação SQL para IBM i*

Aprenda sobre as etapas necessárias ao construir um aplicativo no IBM i usando SQL.

Se seu programa contiver instruções EXEC SQL e chamadas MQI, execute estas etapas:

1. Converta os comandos EXEC SQL em instruções de linguagem nativa. Use o comando CRTSQLCI para um programa C. Use o comando CRTSQLCBLI para um programa COBOL.

   Inclua `OPTION(*NOGEN)` no comando CRTSQLCI ou CRTSQLCBLI. Isso para o processamento para permitir que você inclua os programas de serviço apropriados do IBM MQ. Esse comando coloca o código, por padrão, em QTEMP/QSQLTEMP.
2. Compile o código-fonte usando o comando CRTCMOD (para um programa C) ou o comando CRTCBLMOD (para um programa COBOL).
3. Use CRTPGM para vincular o código compilado aos programas de serviço apropriados do IBM MQ. Isso cria o programa executável.

Um exemplo desse código a seguir (ele compila um programa, SQLTEST, na biblioteca, SQLUSER):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
```

```
CRTPGM   PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux Construindo seu aplicativo processual no Linux

Estas informações descrevem as tarefas adicionais, e as mudanças nas tarefas padrão que devem ser executadas ao construir IBM MQ para os aplicativos Linux a serem executados.

C e C++ são suportados. Para obter informações sobre a preparação de seus programas C++, consulte Usando C++.

Linux *Preparando programas C no Linux*

Programas C pré-compilados são fornecidos no diretório *MQ_INSTALLATION_PATH*/samp/bin . Para construir uma amostra usando o código-fonte, use o compilador gcc.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Trabalhe em seu ambiente normal. Para obter informações adicionais sobre a programação de aplicativos de 64 bits, consulte Padrões de codificação em plataformas de 64 bits.

Vinculando bibliotecas

A tabela a seguir lista as bibliotecas que são necessárias ao preparar programas C no Linux.

- É necessário vincular seus programas à biblioteca apropriada fornecida pelo IBM MQ.

  Em um ambiente não encadeado, vincule-se a somente uma das bibliotecas a seguir:

Arquivo de biblioteca	Tipo de programa/saída
libmqm.so	Servidor para C
libmqic.so & libmqm.so	Cliente para C

  Em um ambiente encadeado, vincule-se a somente uma das bibliotecas a seguir:

Arquivo de biblioteca	Tipo de programa/saída
libmqm_r.so	Servidor para C
libmqic_r.so & libmqm_r.so	Cliente para C

  Nota:

  1. Não é possível se vincular a mais de uma biblioteca. Ou seja, não é possível se vincular a uma biblioteca encadeada e a uma não encadeada ao mesmo tempo.
  2. Se você estiver gravando um serviço instalável (consulte o Administrando IBM MQ para obter mais informações), será necessário vincular-se à biblioteca libmqmzf.so.
  3. Se você estiver produzindo um aplicativo para coordenação externa por um gerenciador de transações compatível com XA, como IBM TXSeries Encina ou BEA Tuxedo, será necessário vincular-se ao libmqmxa.so (ou libmqmxa64.so se o gerenciador de transações tratar o tipo 'long' como 64 bits) e às bibliotecas libmqz.so em um aplicativo não encadeado e ao libmqmxa_r.so (ou libmqmxa64_r.so) e às bibliotecas libmqz_r.so em um aplicativo encadeado.
  4. Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

Linux *Criando aplicativos de 31 bits*

Este tópico contém exemplos dos comandos usados para a construção de programas de 31 bits em vários ambientes.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo cliente C, 31 bits, não encadeado

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Aplicativo cliente C, de 31 bits, encadeado

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Aplicativo do servidor C, de 31 bits, não encadeado

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Aplicativo do servidor C, de 31 bits, encadeado

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Aplicativo cliente C + +, 31 bits, não encadeado

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Aplicativo cliente C + +, de 31 bits, encadeado

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

Aplicativo do servidor C + +, de 31 bits, não encadeado

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Aplicativo do servidor C + +, de 31 bits, encadeado

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

Saída cliente C, de 31 bits, não encadeada

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Saída cliente C, de 31 bits, encadeada

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Saída do servidor C, de 31 bits, não encadeada

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Saída do servidor C, de 31 bits, encadeada

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Criando aplicativos de 32 bits*

Este tópico contém exemplos dos comandos usados para a construção de programas de 32 bits em vários ambientes.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo do cliente C, 32 bits, não encadeado

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Aplicativo do cliente C, 32 bits, encadeado

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Aplicativo do servidor C, 32 bits, não encadeado

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Aplicativo do servidor C, 32 bits, encadeados

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Aplicativo do cliente C++, 32 bits, não encadeado

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Aplicativo do cliente C++, 32 bits, encadeado

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Aplicativo do servidor C++, 32 bits, não encadeado

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Aplicativo do servidor C++, 32 bits, encadeado

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
```

```
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Saída do cliente C, 32 bits, não encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Saída do cliente C, 32 bits, encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Saída do servidor C, 32 bits, não encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Saída do servidor C, 32 bits, encadeado

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Criando aplicativos de 64 bits*

Este tópico contém exemplos dos comandos usados para a construção de programas de 64-bit bits em vários ambientes.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo cliente C, 64 bits, não encadeado

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

Aplicativo cliente C, 64 bits, encadeado

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

Aplicativo do servidor C, de 64 bits, não-encadeados

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

Aplicativo do servidor C, de 64-bit bits, encadeados

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

Aplicativo cliente C ++, 64 bits, não encadeado

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
```

```
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicativo cliente C++, 64 bits, encadeado

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Aplicativo do servidor C ++, de 64 bits, não encadeado

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Aplicativo do servidor C ++, de 64 bits, encadeados

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Saída de cliente C, 64 bits, não encadeado

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

Saída cliente C, de 64-bit bits, encadeados

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

Saída do servidor C, de 64-bit bits, não encadeado

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

Saída do servidor C, de 64-bit bits, encadeados

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux *Preparando programas COBOL em Linux*

Saiba como preparar programas COBOL no Linux e preparar programas COBOL usando o IBM COBOL for Linux no x86 e no Micro Focus COBOL.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

1. Os copybooks COBOL de 32 bits são instalados no seguinte diretório:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   e links simbólicos são criados em:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. Em plataformas de 64 bits, copybooks COBOL de 64 bits são instalados no diretório a seguir:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. Nos seguintes exemplos, configure COBCPY para:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   para aplicações de 32 bits e:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   para aplicativos de 64 bits..

É necessário vincular o seu programa a um dos seguintes:

Arquivo de biblioteca	Tipo de programa/saída
libmqmcb.so	Servidor para COBOL
libmqicb.so	Cliente para COBOL
libmqmcb_r.so	Servidor para COBOL (aplicativo encadeado)
libmqicb_r.so	Cliente para COBOL (aplicativo encadeado)

Preparando programas COBOL usando IBM COBOL para Linux em x86

Programas COBOL de amostra são fornecidos com o IBM MQ. Para compilar esse programa, insira o comando apropriado na lista a seguir:

Aplicativo do servidor não encadeado de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

Aplicativo cliente não encadeado de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

Aplicativo do servidor encadeado de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

Aplicativo cliente encadeado de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

Preparando programas COBOL usando o Micro Focus COBOL

Configure as variáveis de ambiente antes de compilar seu programa da seguinte forma:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Para compilar um programa COBOL de 32 bits, onde suportado, usando Micro Focus COBOL, insira:

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Para compilar um programa COBOL de 64 bits usando o Micro Focus COBOL, insira:

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

em que `amqsput` é o programa de amostra

Consulte a documentação Micro Focus COBOL para uma descrição das variáveis de ambiente que você precisa.

Windows Construindo seu aplicativo processual no Windows

As publicações de sistemas Windows descrevem como construir aplicativos executáveis a partir dos programas que você escreve.

Este tópico descreve as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao criar aplicativos IBM MQ for Windows para executar nos sistemas Windows. As linguagens de programação C, C++, COBOL e Visual Basic são suportadas. Para obter informações sobre a preparação de seus programas C++, consulte Usando C++.

As tarefas que devem ser executadas para criar um aplicativo executável usando o IBM MQ for Windows variam com a linguagem de programação na qual seu código de origem é escrito. Além de codificar as chamadas do MQI em seu código fonte, deve-se incluir as instruções de linguagem apropriadas para incluir os arquivos de inclusão do IBM MQ for Windows para a linguagem que você está usando. Familiarize-se com o conteúdo desses arquivos. Consulte “Arquivos de definição de dados doIBM MQ” na página 729 para obter uma descrição completa.

Windows *Criando aplicativos de 64 bits no Windows*

Aplicativos de 32 bits e de 64 bits são suportadas no IBM MQ for Windows. Os arquivos executáveis e de biblioteca do IBM MQ são fornecidos nos formatos de 32 bits e de 64 bits, use a versão apropriada dependendo do aplicativo com o qual está trabalhando.

Arquivos executáveis e bibliotecas

Ambas as versões de 32 bits e de 64 bits das bibliotecas do IBM MQ são fornecidas nos locais a seguir:

Tabela 148. Localização das bibliotecas do IBM MQ

Versão da biblioteca	**Diretório que contém arquivos de biblioteca**
32 bits	*MQ_INSTALLATION_PATH* \Tools\Lib
64 bits	*MQ_INSTALLATION_PATH* \Tools\Lib64

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativos de 32 bits continuam a funcionar normalmente após a migração. Os arquivos de 32 bits existem no mesmo diretório que em versões anteriores do produto.

Se você deseja criar uma versão de 64 bits, deve-se certificar-se de que o seu ambiente esteja configurado para usar os arquivos da biblioteca em *MQ_INSTALLATION_PATH* \Tools\Lib64. Assegure que a variável de ambiente LIB não esteja configurada para procurar na pasta que contém as bibliotecas de 32 bits.

Windows *Preparando programas C no Windows*

Trabalhe em seu ambiente típico do Windows; o IBM MQ for Windows não requer nada especial.

Para obter informações adicionais sobre a programação de aplicativos de 64 bits, consulte Padrões de codificação em plataformas de 64 bits.

- Vincule seus programas com as bibliotecas apropriadas fornecidas pelo IBM MQ:

Arquivo de biblioteca	Tipo de programa/saída
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	servidor para 32 bits C
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	cliente para 32 bits C
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	cliente para 32 bits C com a coordenação de transação
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	servidor para 64 bits C
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	cliente para 64 bits C
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	cliente para 64 bits C com coordenação de transação

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

O comando a seguir dá um exemplo de compilação do programa de amostra amqsget0 (usando o compilador Microsoft Visual C).

Para aplicativos de 32 bits:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

Para aplicativos de 64 bits:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

Nota:

- Se você estiver gravando um serviço instalável (consulte o Administrando IBM MQ para obter informações adicionais), você precisa se vincular à biblioteca mqmzf.lib.
- Se você estiver produzindo um aplicativo para coordenação externa por um gerenciador de transações compatível com XA como IBM TXSeries Encina ou BEA Tuxedo, será necessário se vincular à biblioteca mqmxa.lib ou mqmxa.lib.
- Se você estiver gravando uma saída CICS, vincule-se à biblioteca mqmcics4.lib.
- Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

- As DLLs devem estar no caminho (PATH) que você especificou.
- Se você usar caracteres minúsculos sempre que possível, é possível mover de sistemas IBM MQ for Windows para IBM MQ for AIX or Linux, onde o uso de caracteres minúsculos é necessário.

Preparando programas CICS e Transaction Server

A origem de C de amostra para uma transação CICS IBM MQ é fornecida por AMQSCIC0.CCS. Você a constrói usando os recursos padrão do CICS. Por exemplo, para o TXSeries para Windows 2000:

1. Configure a variável de ambiente (insira o código a seguir em uma linha):

```
set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. Configure a variável de ambiente USERLIB:

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. Converta, compile e vincule o programa de amostra:

```
cicstcl -l IBMC amqscic0.ccs
```

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Isso é descrito no *Guia de Programação de Aplicativos do Transaction Server for Windows NT (CICS) V4*.

É possível localizar mais informações sobre o suporte a transações do CICS no Administrando IBM MQ.

Windows *Preparando programas COBOL em Windows*

Use estas informações para aprender a preparar programas COBOL no Windows e preparar os programas CICS e Transaction Server.

1. Os copybooks COBOL de 32 bits são instalados no seguinte diretório: *MQ_INSTALLATION_PATH* `\Tools\cobol\CopyBook`.
2. Os copybooks COBOL de 64 bits são instalados no seguinte diretório: *MQ_INSTALLATION_PATH* `\Tools\cobol\CopyBook64`
3. Nos exemplos a seguir, configure `CopyBook` para:

```
CopyBook
```

   para aplicações de 32 bits e:

```
CopyBook64
```

   para aplicativos de 64 bits..

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para preparar programas COBOL em sistemas Windows, vincule seu programa a uma das bibliotecas a seguir fornecidas pelo IBM MQ:

Arquivo de biblioteca	Tipo de programa ou saída
MQ_INSTALLATION_PATH `\Tools\Lib\mqmcb`	Servidor de 32 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib\mqiccb`	Cliente de 32 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib64\mqmcb`	Servidor de 64 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib64\mqiccb`	Cliente de 64 bits para Micro Focus COBOL

Quando você estiver executando um programa no ambiente do cliente MQI, certifique-se de que a biblioteca DOSCALLS apareça antes de qualquer biblioteca COBOL ou IBM MQ.

Preparando programas COBOL usando o Micro Focus COBOL

Vincule novamente quaisquer programas existentes do IBM MQ Micro Focus COBOL de 32 bits usando `mqmcb.lib` ou `mqiccb.lib`, em vez das bibliotecas `mqmcbb` e `mqiccbb`

Para compilar, por exemplo, o programa de amostra `amq0put0`, usando o Micro Focus COBOL:

1. Configure a variável de ambiente COBCPY para apontar para os copy books do IBM MQ COBOL (insira o código a seguir em uma linha):

   ```
   set COBCPY= MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook
   ```

2. Compile o programa para fornecer a você um arquivo de objeto:

   ```
   cobol amq0put0 LITLINK
   ```

3. Vincule o arquivo de objeto para o sistema de tempo de execução.
   - Configure a variável de ambiente LIB para apontar para as bibliotecas COBOL bibliotecas do compilador.
   - Vincule o arquivo de objeto para uso no servidor IBM MQ:

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - Ou vincule o arquivo de objeto para uso no cliente IBM MQ:

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

Preparando programas CICS e Transaction Server

Para compilar e vincular um programa TXSeries for Windows NT V5.1, usando o IBM VisualAge COBOL:

1. Configure a variável de ambiente (insira o código a seguir em uma linha):

   ```
   set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
   Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
   ```

2. Configure a variável de ambiente USERLIB:

   ```
   set USERLIB=MQMCBB.LIB
   ```

3. Converta, compile e vincule o programa:

   ```
   cicstcl -l IBMCOB myprog.ccp
   ```

Isso é descrito no *Transaction Server for Windows NT, V4 Application Programming* Guide.

Para compilar e vincular um programa CICS for Windows V5 usando o Micro Focus COBOL:

- Configure a variável INCLUDE:

  ```
  set
  INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
          drive:\opt\cics\include;%INCLUDE%
  ```

- Configure a variável de ambiente COBCPY:

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- Configure as opções de COBOL:
  - `set`
  - `COBOPTS=/LITLINK /NOTRUNC`

e execute o código a seguir:

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows *Preparando programas Visual Basic no Windows*

Informações a serem consideradas ao usar programas Microsoft Visual Basic no Windows.

Deprecated No IBM MQ 9.0, o suporte para o Microsoft Visual Basic 6.0 foi descontinuado. As classes do IBM MQ para .NET são a tecnologia de substituição recomendada. Para obter mais informações, consulte Desenvolvendo aplicativos .NET.

Nota: As versões de 64 bits dos arquivos do módulo Visual Basic não são fornecidas.

Para preparar programas Visual Basic no Windows:

1. Crie um novo projeto.
2. Inclua o arquivo do módulo fornecido, CMQB.BAS, no projeto.
3. Inclua outros arquivos de módulo fornecidos se você precisar deles:
   - CMQBB.BAS: suporte de MQAI
   - CMQCFB.BAS: suporte de PCF
   - CMQXB.BAS: suporte de saídas de canal
   - CMQPSB.BAS: publicar/assinar

Veja “Codificação no Visual Basic” na página 1065 para obter informações sobre como usar a chamada MQCONNXAny no Visual Basic.

Chame o procedimento MQ_SETDEFAULTS antes de fazer quaisquer chamadas MQI no código do projeto. Esse procedimento configura as estruturas padrão que as chamadas MQI requerem.

Especifique se você estiver criando um servidor ou cliente do IBM MQ, antes de compilar ou executar o projeto, configurando a variável de compilação condicional *MqType*. Configure *MqType* em um projeto Visual Basic como 1 para um servidor ou 2 para um cliente conforme a seguir:

1. Selecione o menu Projeto.
2. Selecione *Name* Propriedades (em que *Name* é o nome do projeto atual).
3. Selecione a guia Fazer na caixa de diálogo.
4. No campo Argumentos de compilação condicional, insira isso para um servidor:

   ```
   MqType=1
   ```

   ou isso para um cliente:

   ```
   MqType=2
   ```

Conceitos relacionados

"Codificação no Visual Basic" na página 1065
Informações a serem consideradas ao codificar programas IBM MQ no Microsoft Visual Basic. O Visual Basic é suportado somente no Windows.

Referências relacionadas

"Vinculando aplicativos Visual Basic ao código do IBM MQ MQI client" na página 932
É possível vincular os aplicativos Microsoft Visual Basic com o código IBM MQ MQI client no Windows.

Windows *Saída de segurança SSPI*

O IBM MQ for Windows fornece uma saída de segurança para o IBM MQ MQI client e o servidor IBM MQ. Esse é um programa de saída do canal que fornece autenticação para canais do IBM MQ usando a Security Services Programming Interface (SSPI). A SSPI fornece os recursos de segurança integrados dos sistemas Windows.

Os pacotes de segurança são carregados a partir de security.dll ou secur32.dll. Esses DLLs são fornecidos com o seu sistema operacional.

Autenticação de via única é fornecida usando os serviços de autenticação NTLM. Autenticação de duas vias é fornecida usando os serviços de autenticação do Kerberos.

O programa de saída de segurança é fornecido no formato de origem e de objeto. É possível usar o código de objeto no estado em que se encontra ou usar o código-fonte como um ponto de início para criar seus próprios programas de saída de usuário.

Consulte também "Usando a saída de segurança SSPI no Windows" na página 1149.

Introdução a saídas de segurança

Uma saída de segurança forma uma conexão segura entre os programas de saída de segurança, onde um programa destina-se a enviar MCA (Agente do Canal de Mensagem), e outro a receber MCA.

O programa que inicia a conexão segura, ou seja, o primeiro programa a obter controle após a sessão do MCA ser estabelecida, é conhecido como o *inicializador de contexto*. O programa parceiro é conhecido como o *aceitador de contexto*.

A tabela a seguir mostra alguns dos tipos de canais que são inicializadores de contexto e seus aceitadores de contexto associados.

Tabela 149. Inicializadores de contexto e seus aceitadores de contexto associados

Inicializador de contexto	**Aceitador de contexto**
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

O programa de saída de segurança tem dois pontos de entrada:

- **SCY_NTLM**

  Usa serviços de autenticação NTLM que fornecem autenticação unilateral. NTLM permite que servidores verifiquem as identidades de seus clientes. Não permite que os clientes verifiquem a identidade de um servidor nem que um servidor verifique a identidade de outro. A autenticação NTLM foi projetada para um ambiente de rede no qual servidores são considerados como genuínos.

- **SCY_KERBEROS**

  Usa serviços de autenticação mútua do Kerberos. O protocolo Kerberos não supõe que os servidores em um ambiente de rede sejam autênticos. As partes em ambas as extremidades de uma conexão de

rede podem verificar a identidade da outra parte. Ou seja, os servidores podem verificar a identidade de clientes e de outros servidores, e os clientes podem verificar a identidade de um servidor.

O que a saída de segurança faz

Este tópico descreve o que os programas de saída do canal SSPI fazem.

Os programas de saída do canal fornecidos fornecem autenticação unidirecional ou de duas vias (mútua) de um sistema parceiro quando uma sessão está sendo estabelecida. Para um canal específico, cada programa de saída tem um *diretor* associado (semelhante a um ID do usuário, consulte "Controle de acesso do IBM MQ e diretores do Windows" na página 1034). Uma conexão entre dois programas de saída é uma associação entre os dois diretores.

Após a sessão subjacente ser estabelecida, uma conexão segura entre os programas de saída de segurança (uma para o MCA de envio e uma para o MCA de recebimento) é estabelecida. A sequência de operações é a seguinte:

1. Cada programa está associado a um diretor específico, por exemplo, como resultado de uma operação de login explícito.
2. O inicializador de contexto solicita uma conexão segura com o parceiro a partir do pacote de segurança (para Kerberos, o parceiro denominado) e recebe um token (chamado token1). O token é enviado, usando a sessão subjacente já estabelecida, ao programa parceiro.
3. O programa parceiro (o aceitador de contexto) passa o token1 para o pacote de segurança, que verifica se o inicializador de contexto é autêntico. Para NTLM, a conexão agora está estabelecida.
4. Para a saída de segurança fornecida pelo Kerberos (ou seja, para autenticação mútua), o pacote de segurança também gera um segundo token (chamado token2), que o aceitador de contexto retorna ao inicializador de contexto usando a sessão subjacente.
5. O inicializador de contexto usa o token2 para verificar se o aceitador de contexto é autêntico.
6. Nesse estágio, se ambos os aplicativos estiverem satisfeitos com a autenticidade do token do parceiro, a conexão segura (autenticada) será estabelecida.

Controle de acesso do IBM MQ e diretores do Windows

O controle de acesso que o IBM MQ fornece é baseado no usuário e no grupo. A autenticação que o Windows fornece é baseada em diretores, como usuário e servicePrincipalName (SPN). No caso de servicePrincipalName, pode haver muitos desses associados a um único usuário.

A saída de segurança SSPI usa os diretores relevantes do Windows para autenticação. Se a autenticação do Windows for bem-sucedida, a saída passa o ID do usuário associado ao diretor do Windows para o IBM MQ para controle de acesso.

Os diretores do Windows relevantes para autenticação variam, dependendo do tipo de autenticação usado.

- Para autenticação NTLM, o diretor do Windows para o Inicializador de contexto é o ID do usuário associado ao processo em execução. Como essa autenticação é unilateral, o diretor associado ao Aceitador de contexto é irrelevante.
- Para autenticação do Kerberos, em canais CLNTCONN, o diretor do Windows é o ID do usuário associado ao processo que está em execução. Caso contrário, o diretor do Windows é o servicePrincipalName formado pela inclusão do prefixo a seguir no QueueManagerName.

```
ibmMQSeries/
```

z/OS Building your procedural application on z/OS

The CICS, IMS, and z/OS publications describe how to build applications that run in these environments.

This collection of topics describes the additional tasks, and the changes to the standard tasks, that you must perform when building IBM MQ for z/OS applications for these environments. COBOL, C, C++, Assembler, and PL/I programming languages are supported. (For information about building C++ applications see Usando C++.)

The tasks that you must perform to create an executable IBM MQ for z/OS application depend on both the programming language that the program is written in, and the environment in which the application will run.

In addition to coding the MQI calls in your program, add the appropriate language statements to include the IBM MQ for z/OS data definition file for the language that you are using. Make yourself familiar with the contents of these files. See “Arquivos de definição de dados doIBM MQ” on page 729 for a full description.

Note

The name **thlqual** is the high-level qualifier of the installation library on z/OS.

z/OS *Preparing your program to run*

After you have written the program for your IBM MQ application to create an executable application, you have to compile or assemble it, then link-edit the resulting object code with the stub program that IBM MQ for z/OS supplies for each environment that it supports.

How you prepare your program depends on both the environment (batch, CICS, IMS(BMP or MPP), Linux or z/OS UNIX System Services) in which the application runs, and the structure of the data sets on your z/OS installation.

“Dynamically calling the IBM MQ stub” on page 1041 describes an alternative method of making MQI calls in your programs so that you do not need to link-edit an IBM MQ stub. This method is not available for all languages and environments.

Do not link-edit a higher level of stub program than that of the version of IBM MQ for z/OS on which your program is running. For example, a program running on MQSeries for OS/390®, V5.2 must not be link-edited with a stub program supplied with IBM MQ for z/OS V7.

z/OS *Building 64 bit C applications*

In z/OS, 64 bit C applications are built using the LP64 compiler and binder options. The IBM MQ for z/OS *cmqc.h* header file recognizes when this option is provided to the compiler, and generates IBM MQ data types and structures appropriate for 64 bit operation.

C code built with this option must be built to use dynamic-link libraries (DLLs) appropriate for the coordination semantic required. To achieve this, you bind the compiled code with the appropriate side-deck defined in the following table:

Table 150. Side-deck name required for each coordination semantic

Coordination	**Side-deck name**
Single phase commit MQI	CSQBMQ2X
Two phase commit with RRS coordination, using RRS verbs	CSQBRR2X
Two phase commit with RRS coordination, using MQI verbs	CSQBRI2X

Note: For 31-bit C applications you also set compiler options for the calling interface (either Language Environment or XPLINK), as described in “Building z/OS batch applications using 31-bit Language Environment or XPLINK” on page 1037. For 64-bit C applications you do not specify the calling interface, because the only supported linkage is XPLINK.

Use the EDCQCB JCL procedure, supplied with *z/OS XL C/C++*, to build a single phase commit IBM MQ program as a batch job, as follows:

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',             < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',             < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//        DD
//        DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

To build an RRS coordinated program in z/OS UNIX System Services, compile and link as follows:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Building z/OS batch applications*

Learn how to build z/OS batch applications and the steps to consider when doing so.

To build an application for IBM MQ for z/OS that runs under z/OS batch, create job control language (JCL) that performs these tasks:

1. Compile (or assemble) the program to produce object code. The JCL for your compilation must include SYSLIB statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - For COBOL, **thlqual**.SCSQCOBC
   - For assembler language, **thlqual**.SCSQMACS
   - For C, **thlqual**.SCSQC370
   - For PL/I, **thlqual**.SCSQPLIC
2. For a C application, prelink the object code created in step “1” on page 1036.
3. For PL/I applications, use the compiler option EXTRN(SHORT).
4. Link-edit the object code created in step “1” on page 1036 (or step “2” on page 1036 for a C application) to produce a load module. When you link-edit the code, you must include one of the IBM MQ for z/OS batch stub programs (CSQBSTUB or one of the RRS stub programs: CSQBRRSI or CSQBRSTB).

   CSQBSTUB
   single-phase commit provided by IBM MQ for z/OS

   CSQBRRSI
   two-phase commit provided by RRS using the MQI

   CSQBRSTB
   two-phase commit provided by RRS directly

   Notes:

   a. If you use CSQBRSTB, you must also link-edit your application with ATRSCSS from SYS1.CSSLIB. Figure 113 on page 1037 and Figure 114 on page 1037 show fragments of JCL to do this. The stubs are language-independent and are supplied in library **thlqual**.SCSQLOAD.

   b. If your application runs under Language Environment, you should ensure you link-edit with the Language Environment DLL instead as described in “Building z/OS batch applications using 31-bit Language Environment or XPLINK” on page 1037.
5. Store the load module in an application load library.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Figure 113. Fragments of JCL to link-edit the object module in the batch environment, using single-phase commit

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

Figure 114. Fragments of JCL to link-edit the object module in the batch environment, using two-phase commit

To run a batch or RRS program, you must include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB or JOBLIB data set concatenation.

To run a TSO program, you must include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB used by the TSO session.

To run a batch program from the z/OS UNIX System Services shell, add the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD to the STEPLIB specification in your $HOME?.profile like this:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Building z/OS batch applications using 31-bit Language Environment or XPLINK*

IBM MQ for z/OS provides a set of dynamic link libraries (DLLs) that must be used when you link-edit your applications.

There are two variants of the libraries that allow the application to use one of the following calling interfaces:

- The 31-bit Language Environment calling interface.
- The 31-bit XPLINK calling interface. z/OS XPLINK is a high performance calling convention available for C applications. See XPLINK | NOXPLINK in the z/OS 2.2 documentation.

To use the DLLs, the application is bound or linked against so called *sidedecks*, instead of the stubs provided with earlier versions. The sidedecks are found in the SCSQDEFS library (instead of the SCSQLOAD library).

Table 151. Variants of dynamic link libraries			
Commit	**31-bit Language Environment DLL**	**31-bit XPLINK DLL**	**Equivalent stub name**
1 phase commit MQI libraries	CSQBMQ1	CSQBMQ1X	CSQBSTUB
2 phase commit with RRS co-ordination using RRS transaction-control verbs	CSQBRR1	CSQBRR1X	CSQBRSTB
2 phase commit with RRS co-ordination using MQI transaction-control verbs	CSQBRI1	CSQBRI1X	CSQBRRSI

Note: All sidedecks contain a definition of the data conversion entry point, MQXCNVC, previously resolved by including CSQASTUB.

Common issues:

- The following message appears on the job log if your application uses asynchronous message consume (MQCB, MQCTL or MQSUB calls) and the previous DLL interface is not used:

  ```
  CSQB001E Language environment programs running in z/OS batch or z/OS UNIX System Services must
  use the DLL interface to IBM MQ
  ```

  Solution: Rebuild your application using sidedecks instead of stubs as detailed previously.

- At program build time, the following message appears

  ```
  IEW2469E The Attributes of a reference to MQAPI-NAME from section your-code do not match the
  attributes of
  the target symbol
  ```

  Reason: This means that you have compiled your XPLINK program with V701 (or later) version of cmqc.h, but are not binding with sidedecks.

  Solution: Change your program's build file to bind against the appropriate sidedeck from SCSQDEFS instead of a stub from SCSQLOAD

The following sample JCL demonstrates how you can compile and link-edit a C program to use the 31 bit Language Environment DLL calling interface:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

Note: The compilation uses the **DLL** option. The link-edit uses **DYNAM=DLL** option and the references the **CSQBMQ1** library.

The following sample JCL demonstrates how you can compile and link-edit a C program to use the 31 bit XPLINK DLL calling interface:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

Note: The compilation uses the **XPLINK** and **DLL** options. The link-edit uses **DYNAM=DLL** option and references the **CSQBMQ1X** library.

Ensure that you add the compilation option DLL to each program in the module. Messages such as IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED are an indication that you need to check that all of the programs have been compiled with the DLL option.

z/OS *Building CICS applications in z/OS*

Use this information when building CICS applications in z/OS.

To build an application for IBM MQ for z/OS that runs under CICS, you must:

- Translate the CICS commands in your program into the language in which the rest of your program is written.
- Compile or assemble the output from the translator to produce object code.
  - For PL/I programs, use the compiler option EXTRN(SHORT).
  - For C applications, if the application is not using XPLINK, use the compiler option DEFINE(MQ_OS_LINKAGE=1).
- Link-edit the object code to create a load module.

CICS provides a procedure to execute these steps in sequence for each of the programming languages it supports.

- For CICS Transaction Server for z/OS, the *CICS Transaction Server for z/OS System Definition Guide* describes how to use these procedures and the *CICS/ESA Application Programming Guide* gives more information on the translation process.

You must include:

- In the SYSLIB statement of the compilation (or assembly) stage, statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
  - For COBOL, **thlqual**.SCSQCOBC
  - For assembler language, **thlqual**.SCSQMACS
  - For C, **thlqual**.SCSQC370
  - For PL/I, **thlqual**.SCSQPLIC
- In your link-edit JCL, the IBM MQ for z/OS CICS stub program (CSQCSTUB). Figure 115 on page 1040 shows fragments of JCL code to do this. The stub is language-independent and is supplied in library **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Figure 115. Fragments of JCL to link-edit the object module in the CICS environment

- For CICS versions later than CICS TS 3.2, or, if you want to use IBM MQ message property APIs, or IBM MQ APIs MQCB, MQCTL, MQSTAT, MQSUB or MQSUBR, you must linkedit your object code with the CICS supplied stub, DFHMQSTB and not the IBM MQ supplied CSQCSTUB. For more information about building IBM MQ programs for CICS, see API stub program to access IBM MQ MQI calls in the CICS product documentation.

When you have completed these steps, store the load module in an application load library and define the program to CICS in the usual way.

Before you run a CICS program, your system administrator must define it to CICS as an IBM MQ program and transaction, You can then run it in the typical way.

z/OS *Building IMS (BMP or MPP) applications*

Use this information when building IMS (BMP or MPP) applications.

If you are building batch DL/I programs, see “Building z/OS batch applications” on page 1036. To build other applications that run under IMS (either as a BMP or an MPP), create JCL that performs these tasks:

1. Compile (or assemble) the program to produce object code. The JCL for your compilation must include SYSLIB statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - For COBOL, **thlqual**.SCSQCOBC
   - For assembler language, **thlqual**.SCSQMACS
   - For C, **thlqual**.SCSQC370
   - For PL/I, **thlqual**.SCSQPLIC
2. For a C application, prelink the object module created in step “1” on page 1040.
3. For PL/I programs, use the compiler option EXTRN(SHORT).
4. For a C application, if the application is not using XPLINK, use the compiler option DEFINE(MQ_OS_LINKAGE=1).
5. Link-edit the object code created in step “1” on page 1040 (or step “2” on page 1040 for a C/370 application) to produce a load module:
   a. Include the IMS language interface module (DFSLI000).
   b. Include the IBM MQ for z/OS IMS stub program (CSQQSTUB). Figure 116 on page 1041 shows fragments of JCL to do this. The stub is language independent and is supplied in library **thlqual**.SCSQLOAD.

      Note: If you are using COBOL, select the NODYNAM compiler option to enable the linkage editor to resolve references to CSQQSTUB unless you intend to use dynamic linking as described in “Dynamically calling the IBM MQ stub” on page 1041.
6. Store the load module in an application load library.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Figure 116. Fragments of JCL to link-edit the object module in the IMS environment

Before you run an IMS program, your system administrator must define it to IMS as an IBM MQ program and transaction: you can then run it in the typical way.

z/OS *Building z/OS UNIX System Services applications*

Use this information when building z/OS UNIX System Services applications.

To build a C application for IBM MQ for z/OS that runs under z/OS UNIX System Services, compile and link your application as follows:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

where **thlqual** is the high-level qualifier used by your installation.

To run the C program, you need to add the following to your `.profile` file; this should be in your root directory:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Note that you need to exit from z/OS UNIX System Services, and enter z/OS UNIX System Services again, for the change to be recognized.

If you want to run multiple shells, add the word export at the beginning of the line, that is:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Once this completes successfully you can link the CSQBSTUB and issue IBM MQ calls.

"Dynamically calling the IBM MQ stub" on page 1041 describes an alternative method of making MQI calls in your programs so that you do not need to link-edit an IBM MQ stub. This method is not available for all languages and environments.

Do not link-edit a higher level of stub program than that of the version of IBM MQ for z/OS on which your program is running. For example, a program running on IBM WebSphere MQ for z/OS 7.1 must not be link-edited with a stub program supplied with IBM MQ for z/OS 8.0.

z/OS *Dynamically calling the IBM MQ stub*

Instead of link-editing the IBM MQ stub program with your object code, you can dynamically call the stub from within your program.

You can do this in the batch, IMS, and CICS environments. This facility is not supported in the RRS environment. If your application program uses RRS to coordinate updates, see "RRS Considerations" on page 1046.

However, this method:

- Increases the complexity of your programs
- Increases the storage required by your programs at execution time
- Reduces the performance of your programs

- Means that you cannot use the same programs in other environments

If you call the stub dynamically, the appropriate stub program and its aliases must be available at execution time. To ensure this, include the IBM MQ for z/OS data set SCSQLOAD:

- For batch and IMS, in the STEPLIB concatenation of the JCL.
- For CICS, in the CICS DFHRPL concatenation.

For IMS, ensure that the library containing the dynamic stub (built as described in the information about installing the IMS adapter in Setting up the IMS adapter) is ahead of the data set SCSQLOAD in the STEPLIB concatenation of the region JCL.

Use the names shown in Table 152 on page 1042 when you call the stub dynamically. In PL/I, only declare the call names used in your program.

Table 152. Call names for dynamic linking

MQI call	Batch (non-RRS) dynamic call names	CICS dynamic call names	IMS dynamic call names
MQBACK	CSQBBACK	not supported	Not supported
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Not supported
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	not supported	Not supported
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Not supported
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGET	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Note: 1. These API calls are available only when using CICS TS 3.2 or later and the CSQCSTUB shipped with CICS must be used. For CICS TS 3.2, APAR PK66866 must be applied. For CICS TS 4.1, APAR PK89844 must be applied.

For examples of how to use this technique, see the following figures:

- Batch and COBOL: see Figure 117 on page 1043
- CICS and COBOL: see Figure 118 on page 1043
- IMS and COBOL: see Figure 119 on page 1044
- Batch and assembler: see Figure 120 on page 1044
- CICS and assembler: see Figure 121 on page 1044
- IMS and assembler: see Figure 122 on page 1044
- Batch and C: Figure 123 on page 1045
- CICS and C: see Figure 124 on page 1045
- IMS and C: see Figure 125 on page 1045
- Batch and PL/I: see Figure 126 on page 1045
- IMS and PL/I: see Figure 127 on page 1046

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 117. Dynamic linking using COBOL in the batch environment

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 118. Dynamic linking using COBOL in the CICS environment

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                    PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 119. Dynamic linking using COBOL in the IMS environment

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

Figure 120. Dynamic linking using assembly language in the batch environment

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Figure 121. Dynamic linking using assembly language in the CICS environment

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

Figure 122. Dynamic linking using assembly language in the IMS environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 123. Dynamic linking using C language in the batch environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 124. Dynamic linking using C language in the CICS environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 125. Dynamic linking using C language in the IMS environment

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Figure 126. Dynamic linking using PL/I in the batch environment

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Figure 127. Dynamic linking using PL/I in the IMS environment

z/OS *RRS Considerations*

Consider using this information if your application program uses RRS to coordinate updates.

IBM MQ provides two different stubs for batch programs which need RRS coordination - see "The RRS batch adapter" on page 903. The difference in behavior of later API calls is determined at MQCONN time by the batch adapter from information passed by the stub routine on the MQCONN or MQCONNX API. This means that dynamic API calls are available for batch programs which need RRS coordination, provided that the initial connection to IBM MQ was done by using the appropriate stub. The following example illustrates this:

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
                               W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS

Debugging your programs

Use this information to learn about debugging TSO and CICS programs, and an insight into CICS trace.

The main aids to debugging IBM MQ for z/OS application programs are the reason codes returned by each API call. For a list of these, including ideas for corrective action, see:

- mensagens, conclusão e códigos de razão do IBM MQ for z/OS for IBM MQ for z/OS
- Mensagens e códigos de razão for all other IBM MQ platforms

This topic also suggests other debugging tools to use in particular environments.

Debugging TSO programs

The following interactive debugging tools are available for TSO programs:

- TEST tool
- VS COBOL II interactive debugging tool
- INSPECT interactive debugging tool for C and PL/I programs

Debugging CICS programs

You can use the CICS Execution Diagnostic Facility (CEDF) to test your CICS programs interactively without having to modify the program or program-preparation procedure.

For more information about EDF, see the *CICS Transaction Server for z/OS CICS Application Programming Guide*.

CICS trace

You will probably also find it helpful to use the CICS Trace Control transaction (CETR) to control CICS trace activity.

For more information about CETR, see *CICS Transaction Server for z/OS CICS-Supplied Transactions* manual.

To determine whether CICS trace is active, display connection status using the CKQC panel. This panel also shows the trace number.

To interpret CICS trace entries, see Table 153 on page 1047.

The CICS trace entry for these values is AP0 *xxx* (where *xxx* is the trace number specified when the CICS adapter was enabled). All trace entries except CSQCTEST are issued by CSQCTRUE. CSQCTEST is issued by CSQCRST and CSQCDSP.

Table 153. CICS adapter trace entries

Name	Description	Trace sequence	Trace data
CSQCABNT	Abnormal termination	Before issuing END_THREAD ABNORMAL to IBM MQ. This is because of the end of the task and an implicit backout could be performed by the application. A ROLLBACK request is included in the END_THREAD call in this case.	Unit of work information. You can use this information when finding out about the status of work. (For example, it can be verified against the output produced by the DISPLAY THREAD command, or the IBM MQ for z/OS log print utility.)
CSQCBACK	Syncpoint backout	Before issuing BACKOUT to IBM MQ for z/OS. This is due to an explicit backout request from the application.	Unit of work information.
CSQCCCRC	Completion code and reason code	After unsuccessful return from API call.	Completion code and reason code.
CSQCCOMM	Syncpoint commit	Before issuing COMMIT to IBM MQ for z/OS. This can be due to a single-phase commit request or the second phase of a two-phase commit request. The request is due to an explicit syncpoint request from the application.	Unit of work information.

Table 153. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCEXER	Execute resolve	Before issuing EXECUTE_RESOLVE to IBM MQ for z/OS.	The unit of work information of the unit of work issuing the EXECUTE_RESOLVE. This is the last indoubt unit of work in the resynchronization process.
CSQCGETW	GET wait	Before issuing CICS wait.	Address of the ECB to be waited on.
CSQCGMGD	GET message data	After successful return from MQGET.	Up to 40 bytes of the message data.
CSQCGMGH	GET message handle	Before issuing MQGET to IBM MQ for z/OS.	Object handle.
CSQCGMGI	Get message ID	After successful return from MQGET.	Message ID and correlation ID of the message.
CSQCINDL	Indoubt list	After successful return from the second INQUIRE_INDOUBT.	The indoubt units of work list.
CSQCINDO	IBM use only		
CSQCINDS	Indoubt list size	After successful return from the first INQUIRE_INDOUBT and the indoubt list is not empty.	Length of the list. Divided by 64 gives the number of indoubt units of work.
CSQCINQH	INQ handle	Before issuing MQINQ to IBM MQ for z/OS.	Object handle.
CSQCLOSH	CLOSE handle	Before issuing MQCLOSE to IBM MQ for z/OS.	Object handle.
CSQCLOST	Disposition lost	During the resynchronization process, CICS informs the adapter that it has been restarted so no disposition information regarding the unit of work being resynchronized is available.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNIND	Disposition not indoubt	During the resynchronization process, CICS informs the adapter that the unit of work being resynchronized should not have been indoubt (that is, perhaps it is still running).	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNORT	Normal termination	Before issuing END_THREAD NORMAL to IBM MQ for z/OS. This is due to the end of the task and therefore the application might perform an implicit syncpoint commit. A COMMIT request is included in the END_THREAD call in this case.	Unit of work information.

Table 153. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCOPNH	OPEN handle	After successful return from MQOPEN.	Object handle.
CSQCOPNO	OPEN object	Before issuing MQOPEN to IBM MQ for z/OS.	Object name.
CSQCPMGD	PUT message data	Before issuing MQPUT to IBM MQ for z/OS.	Up to 40 bytes of the message data.
CSQCPMGH	PUT message handle	Before issuing MQPUT to IBM MQ for z/OS.	Object handle.
CSQCPMGI	PUT message ID	After successful MQPUT from IBM MQ for z/OS.	Message ID and correlation ID of the message.
CSQCPREP	Syncpoint prepare	Before issuing PREPARE to IBM MQ for z/OS in the first phase of two-phase commit processing. This call can also be issued from the distributed queuing component as an API call.	Unit of work information.
CSQCP1MD	PUTONE message data	Before issuing MQPUT1 to IBM MQ for z/OS.	Up to 40 bytes of data of the message.
CSQCP1MI	PUTONE message ID	After successful return from MQPUT1.	Message ID and correlation ID of the message.
CSQCP1ON	PUTONE object name	Before issuing MQPUT1 to IBM MQ for z/OS.	Object name.
CSQCRBAK	Resolved backout	Before issuing RESOLVE_ROLLBACK to IBM MQ for z/OS.	Unit of work information.
CSQCRCMT	Resolved commit	Before issuing RESOLVE_COMMIT to IBM MQ for z/OS.	Unit of work information.
CSQCRMIR	RMI response	Before returning to the CICS RMI (resource manager interface) from a specific invocation.	Architected RMI response value. Its meaning depends of the type of the invocation. These values are documented in the *CICS Transaction Server for z/OS Customization Guide*. To determine the type of invocation, look at previous trace entries produced by the CICS RMI component.
CSQCRSYN	Resynchronization	Before the resynchronization process starts for the task.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCSETH	SET handle	Before issuing MQSET to IBM MQ for z/OS.	Object handle.
CSQCTASE	IBM use only		

Table 153. CICS adapter trace entries (continued)			
Name	**Description**	**Trace sequence**	**Trace data**
CSQCTEST	Trace test	Used in EXEC CICS ENTER TRACE call to verify the trace number supplied by the user or the trace status of the connection.	No data.
CSQCDCFF	IBM use only		

Manipulando erros de programa processual

Estas informações explicarão os erros associados às chamadas MQI de aplicativos, ao realizar uma chamada ou quando sua mensagem for entregue para seu destino final.

Sempre que possível, o gerenciador de filas retornará quaisquer erros assim que uma chamada MQI for realizada. Elas são *erros determinados localmente*.

Ao enviar mensagens para uma fila remota, os erros poderão não estar aparentes quando a chamada MQI for realizada. Nesse caso, o gerenciador de filas que identifica os erros os relatam enviando outra mensagem para o programa de origem. Elas são *erros determinados remotamente*.

Erros determinados localmente

Informações sobre erros localmente determinados que incluem: falha em uma chamada MQI, interrupções do sistema e mensagens contendo dados incorretos.

As três causas mais comuns de erros que o gerenciador de filas pode relatar imediatamente são:

- falha de uma chamada MQI; por exemplo, porque a fila está cheia
- Uma interrupção à execução de alguma parte do sistema do qual seu aplicativo depende; por exemplo, o gerenciador de filas
- Mensagens que contêm dados que não podem ser processados com sucesso

Se você estiver usando o recurso de entrada assíncrona, os erros não são relatados imediatamente. Use a chamada MQSTAT para recuperar informações de status sobre operações de entrada assíncronas anteriores.

Falha de uma chamada MQI

O gerenciador de filas pode relatar imediatamente quaisquer erros na codificação de uma chamada MQI. Ele faz isto usando um conjunto de códigos de retorno predefinidos. Esses são divididos em códigos de conclusão e códigos de razão.

Para mostrar se uma chamada foi bem-sucedida, o gerenciador de filas retorna um *código de conclusão* quando a chamada é concluída. Há três códigos de conclusão, indicando êxito, conclusão parcial e falha da chamada. O gerenciador de filas também retorna um *código de razão* que indica a razão para a conclusão parcial ou a falha da chamada.

Os códigos de conclusão e de razão para cada chamada são listados com a descrição dessa chamada em Códigos de retorno. Para obter informações mais detalhadas, incluindo ideias para ação corretiva, consulte:

- z/OS mensagens, conclusão e códigos de razão do IBM MQ for z/OS for IBM MQ for z/OS
- Mensagens e códigos de razão para todas as outras plataformas IBM MQ

Projete seus programas para manipular todos os códigos de retorno que podem surgir a partir de cada chamada.

Interrupções doSystem i

Seu aplicativo pode não estar ciente de qualquer interrupção se o gerenciador de filas ao qual ele está conectado tiver de se recuperar de uma falha do sistema. No entanto, deve-se projetar seu aplicativo para assegurar que os dados não sejam perdidos se ocorrer uma interrupção.

Os métodos que é possível usar para assegurar que seus dados permaneçam consistentes depende da plataforma na qual o gerenciador de filas está em execução:

z/OS
Em ambientes CICS e IMS, é possível fazer chamadas MQPUT e MQGET dentro de unidades de trabalho que são gerenciadas pelo CICS ou IMS. No ambiente de lote, é possível fazer chamadas MQPUT e MQGET da mesma maneira, mas deve-se declarar pontos de sincronização usando:

- As chamadas IBM MQ for z/OS MQCMIT e MQBACK (consulte "Confirmando e fazendo backup de unidades de trabalho" na página 865) ou
- O z/OS Transaction Management and Recoverable Resource Manager Services (RRS) para fornecer suporte ao ponto de sincronização de duas fases. RRS permite que você atualize ambos os recursos IBM MQ e outros recursos do produto ativado para RRS, como recursos de procedimento armazenado do Db2, dentro de uma unidade de trabalho lógica única. Para obter informações sobre suporte ao ponto de sincronização RRS, consulte "Transaction management and recoverable resource manager services" na página 870.

IBM i
É possível fazer suas chamadas MQPUT e MQGET dentro de unidades globais de trabalho que são gerenciados pelo controle de compromisso do IBM i. É possível declarar pontos de sincronização usando os comandos nativos do IBM i COMMIT e ROLLBACK ou comandos específicos da linguagem. As unidades de trabalho locais são gerenciadas pelo IBM MQ usando as chamadas MQCMIT e MQBACK.

Sistemas AIX, Linux, and Windows
Nestes ambientes, é possível fazer suas chamadas MQPUT e MQGET no modo usual, mas deve-se declarar pontos de sincronização usando as chamadas MQCMIT e MQBACK (consulte "Confirmando e fazendo backup de unidades de trabalho" na página 865). No ambiente CICS, os comandos MQCMIT e MQBACK são desativados, porque é possível fazer suas chamadas MQPUT e MQGET dentro de unidades de trabalho que são gerenciadas pelo CICS.

Use mensagens persistentes para transportar todos os dados que não podem ser perdidos. Mensagens persistentes são recolocadas em filas se o gerenciador de filas tiver de se recuperar de uma falha. ALW Com IBM MQ em AIX, Linux, and Windows, uma chamada MQGET ou MQPUT dentro de seu aplicativo falhará no ponto de preencher todos os arquivos de log, com a mensagem MQRC_RESOURCE_PROBLEM. Para obter mais informações sobre arquivos de log no AIX, Linux, and Windows, consulte Administrando IBM MQ. z/OS Para o z/OS, consulte Planejando em z/OS ..

Se o gerenciador de filas for interrompido por um operador enquanto um aplicativo está em execução, a opção quiesce geralmente é usada. O gerenciador de filas entra em um estado quiesce no qual os aplicativos podem continuar a trabalhar, mas eles devem finalizar assim que conveniente. Aplicativos pequenos e rápidos provavelmente podem ignorar o estado de quiesce e continuar até que eles finalizem normalmente. Aplicativos de execução mais longa ou aqueles que aguardam a chegada de mensagens devem usar a opção *fail if quiescing* quando usarem as chamadas MQOPEN, MQPUT, MQPUT1 e MQGET. Essas opções significam que as chamadas falham quando o gerenciador de filas executa quiesce, mas o aplicativo ainda pode ter tempo para finalizar limpamente, emitindo chamadas que ignoram o estado de quiesce. Tais aplicativos também poderiam confirmar ou restaurar as mudanças que foram feitas e, em seguida, finalizar.

Se o gerenciador de filas for forçado a parar (ou seja, parar sem quiesce), os aplicativos receberão o código de razão MQRC_CONNECTION_BROKEN quando eles fazem chamadas MQI. Saia do aplicativo ou, como alternativa, em sistemas IBM i IBM MQ for IBM i, AIX, Linux, and Windows, emita uma chamada MQDISC.

Mensagens contendo dados incorretos

Ao usar unidades de trabalho em seu aplicativo, se um programa não puder processar com êxito uma mensagem que ele recupera de uma fila, a chamada MQGET é restaurada.

O gerenciador de filas mantém uma contagem (no campo *BackoutCount* do descritor de mensagens) do número de vezes que ocorre. Ele mantém essa contagem no descritor de cada mensagem que é afetada. Essa contagem pode fornecer informações valiosas sobre a eficiência de um aplicativo. Mensagens com contagens de restauração que estão aumentando com o tempo estão sendo rejeitadas repetidamente; projete seu aplicativo para que ele analise as razões para isso e manipula tais mensagens, conforme necessário.

z/OS No IBM MQ for z/OS, para fazer a contagem de restauração sobreviver a reinicializações do gerenciador de filas, configure o atributo **HardenGetBackout** como MQQA_BACKOUT_HARDENED; caso contrário, se o gerenciador de filas precisa ser reiniciado, ele não mantém uma contagem de restauração precisa para cada mensagem. Configurar o atributo desse modo inclui a penalidade de processamento extra.

Em sistemas IBM MQ for IBM i IBM i, AIX, Linux, and Windows, a contagem de restauração sempre sobrevive às reinicializações do gerenciador de filas.

z/OS Além disso, no IBM MQ for z/OS, quando você remover mensagens de uma fila dentro de uma unidade de trabalho, será possível marcar uma mensagem para que não seja disponibilizada novamente se a unidade de trabalho for restaurada pelo aplicativo. A mensagem marcada é tratada como se tivesse sido recuperada sob uma nova unidade de trabalho. É possível marcar a mensagem que está a ignorar a restauração usando a opção MQGMO_MARK_SKIP_BACKOUT(na estrutura MQGMO) quando você usar a chamada MQGET. Consulte “Ignorando restauração” na página 811 para obter mais informações sobre esta técnica.

Usando as mensagens de relatório para determinação de problemas

O gerenciador de filas remotas não pode relatar erros como uma falha ao colocar uma mensagem em uma fila ao fazer sua chamada MQI, mas ele pode enviar uma mensagem de relatório para dizer como ele processou a mensagem.

No seu aplicativo, é possível criar mensagens de relatório (MQPUT), bem como selecionar a opção para recebê-las (neste caso elas serão enviadas por um outro aplicativo ou por um gerenciador de filas).

Criando mensagens de relatório

Mensagens de relatório ativam um aplicativo para informar outro aplicativo que ele não pode lidar com a mensagem que foi enviada.

No entanto, o campo *Report* deve inicialmente ser analisado para determinar se o aplicativo que enviou a mensagem está interessado em ser informado de qualquer problema. Ao determinar que uma mensagem de relatório é necessária, deve-se decidir:

- Se deseja incluir a mensagem original inteira, incluir apenas os primeiros 100 bytes de dados ou não incluir nenhum da mensagem original.
- O que fazer com a mensagem original. É possível descartá-la ou deixá-la ir para a fila de mensagens não entregues.
- Se o conteúdo do *MsgId* e *CorrelId* campos também são necessários.

Use o campo *Feedback* para indicar a razão para a mensagem de relatório estar sendo gerada. Coloque suas mensagens de relatório na fila de resposta de um aplicativo. Consulte Feedback para obter informações adicionais.

Solicitando e recebendo relatório de mensagens (MQGET)

Ao enviar uma mensagem para outro aplicativo, você não é informado sobre quaisquer problemas, a menos que conclua o campo *Report* para indicar o feedback requerido. Consulte Estrutura do campo de relatório para as opções disponíveis.

Gerenciadores de filas sempre colocam mensagens de relatório na fila de resposta de um aplicativo e é recomendado que seus próprios aplicativos façam o mesmo. Ao usar o recurso de mensagem de relatório, especifique o nome de sua fila de resposta no descritor de mensagens da sua mensagem; caso contrário, a chamada MQPUT falhará.

Seu aplicativo deve conter procedimentos que monitoram sua fila de resposta e processar quaisquer mensagens que cheguem a ela. Lembre-se de que uma mensagem de relatório pode conter toda a mensagem original, os primeiros 100 bytes da mensagem original ou nenhum da mensagem original.

O gerenciador de filas configura o campo *Feedback* da mensagem de relatório para indicar a razão do erro; por exemplo, a fila de destino não existe. Os programas devem fazer o mesmo.

Para obter mais informações sobre mensagens de relatório, consulte “Mensagens de relatório” na página 21.

Erros determinados remotamente

Ao enviar mensagens para uma fila remota, mesmo quando o gerenciador de filas locais tiver processado sua chamada MQI sem encontrar um erro, outros fatores podem influenciar o modo como a mensagem é manipulada por um gerenciador de filas remotas.

Por exemplo, a fila que você está direcionando pode estar cheia ou pode nem sequer existir. Se sua mensagem deve ser tratada por outros gerenciadores de filas intermediários na rota para a fila de destino, qualquer um desses poderia localizar um erro.

Problemas na entrega de uma mensagem

Quando uma chamada MQPUT falhar, é possível tentar colocar a mensagem na fila novamente, retorne-a para o emissor ou coloque-a na fila de mensagens não entregues.

Cada opção tem seus méritos, mas você pode não desejar tentar colocar uma mensagem novamente se a razão pela qual MQPUT falhou tiver sido porque a fila de destino estava cheia. Nesta instância, colocá-la na fila de mensagens não entregues permite que você a entregue à fila de destino correta posteriormente.

Tentar novamente entrega da mensagem

Antes da mensagem ser colocada em uma fila de mensagens não entregues, um gerenciador de filas remotas tenta colocar a mensagem na fila novamente se os atributos *MsgRetryCount* e *MsgRetryInterval* tiverem sido configurados para o canal ou se houver um programa de saída de nova tentativa para que ele use (o nome do qual é retido no campo do atributo do canal *MsgRetryExitId*).

Se o campo *MsgRetryExitId* estiver em branco, os valores nos atributos *MsgRetryCount* e *MsgRetryInterval* são usados.

Se o campo *MsgRetryExitId* não estiver em branco, o programa de saída deste nome será executado. Para obter mais informações sobre como usar seus próprios programas de saída, consulte “Programas de Saída de Canal para Canais de Mensagens” na página 973.

Retornar mensagem ao emissor

Você retorna uma mensagem para o emissor solicitando que uma mensagem de relatório seja gerada para incluir tudo da mensagem original.

Consulte “Mensagens de relatório” na página 21 para obter detalhes sobre opções de mensagem de relatório.

Usando a fila de mensagens não entregues

Quando um gerenciador de filas não puder entregar uma mensagem, ele tentará colocar a mensagem em sua fila de mensagens não entregues. Essa fila deve ser definida quando o gerenciador de filas estiver instalado.

Seus programas podem usar a fila de mensagens não entregues da mesma maneira que o gerenciador de filas a usa. É possível localizar o nome da fila de mensagens não entregues ao abrir o objeto do gerenciador de filas (usando a chamada MQOPEN) e consultar a respeito do atributo **DeadLetterQName** (usando a chamada MQINQ).

Quando o gerenciador de filas coloca uma mensagem nessa fila, ele inclui um cabeçalho à mensagem cujo formato é descrito pela estrutura de cabeçalho de mensagens não entregues (MQDLH); consulte MQDLH – Cabeçalho de mensagens não entregues. Este cabeçalho inclui o nome da fila de destino e a razão pela qual a mensagem foi colocada na fila de mensagens não entregues. Ela deve ser removida e o problema deve ser resolvido antes que a mensagem seja colocada na fila pretendida. Além disso, o gerenciador de filas muda o campo *Format* do descritor de mensagens (MQMD) para indicar que a mensagem contém uma estrutura MQDLH.

Estrutura MQDLH

É recomendável incluir uma estrutura MQDLH em todas as mensagens colocadas na fila de mensagens não entregues; no entanto, se você pretende usar o manipulador de mensagens não entregues fornecido por determinados produtos IBM MQ, deve-se incluir uma estrutura MQDLH nas mensagens.

A adição do cabeçalho a uma mensagem pode tornar a mensagem muito longa para a fila de mensagens não entregues, portanto, certifique-se sempre de que suas mensagens sejam mais curtas do que o tamanho máximo permitido para a fila de mensagens não entregues em no mínimo o valor da constante MQ_MSG_HEADER_LENGTH. O tamanho máximo de mensagens permitidas em uma fila é determinado pelo valor do atributo **MaxMsgLength** da fila. Para a fila de mensagens não entregues, certifique-se de que este atributo esteja configurado para o máximo permitido pelo gerenciador de filas. Se o seu aplicativo não puder entregar uma mensagem e a mensagem for muito longa para ser colocada na fila de mensagens não entregues, siga o conselho fornecido na descrição da estrutura MQDLH.

Certifique-se de que a fila de mensagens não entregues seja monitorada e que qualquer mensagem que chega dela seja processada. O manipulador da fila de mensagens não entregues é executado como um utilitário em lote e pode ser usado para executar diversas ações em mensagens selecionadas na fila de mensagens não entregues. Para obter detalhes adicionais, consulte “Processamento de fila de mensagens não entregues” na página 1054.

Se uma conversão de dados for necessária, o gerenciador de filas converterá as informações do cabeçalho ao usar a opção MQGMO_CONVERT na chamada MQGET. Se o processo que coloca a mensagem for um MCA, o cabeçalho será seguido por todo o texto da mensagem original.

As mensagens colocadas na fila de mensagens não entregues podem ser truncadas se forem muito longas para esta fila. Um possível indício dessa situação é se as mensagens na fila de mensagens não entregues tiverem o mesmo comprimento que o valor do atributo **MaxMsgLength** da fila.

Processamento de fila de mensagens não entregues

Essas informações contêm informações da interface de programação de uso geral ao usar processamento de fila de mensagens não entregues.

O processamento da fila de mensagens não entregues depende dos requisitos do sistema local, mas considere o seguinte ao definir a especificação:

- A mensagem pode ser identificada como tendo um cabeçalho de fila de mensagens não entregues porque o valor do campo de formato no MQMD é MQFMT_DEAD_LETTER_HEADER.
- No IBM MQ for z/OS usando o CICS, se um MCA colocar essa mensagem na fila de mensagens não entregues, o campo *PutApplType* será MQAT_CICS e o campo *PutApplName* será o *ApplId* do sistema CICS seguido pelo nome da transação do MCA.
- A razão para que a mensagem seja roteada para a fila de mensagens não entregues está contida no campo *Reason* do cabeçalho da fila de mensagens não entregues.

- O cabeçalho da fila de mensagens não entregues contém detalhes sobre o nome da fila de destino e o nome do gerenciador de filas.
- O cabeçalho da fila de mensagens não entregues contém campos que precisam ser restabelecidos no descritor de mensagens antes que a mensagem seja colocada na fila de destino. São elas:
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- O descritor de mensagens é o mesmo que PUT pelo aplicativo original, exceto para os três campos mostrados (Encoding, CodedCharSetId e Format).

O aplicativo de fila de mensagens não entregues deve executar um ou mais dos procedimentos a seguir:

- Examinar o campo *Reason*. Uma mensagem pode ter sido colocada por um MCA pelas razões a seguir:
  - A mensagem era mais longa do que o tamanho máximo da mensagem para o canal

    A razão é MQRC_MSG_TOO_BIG_FOR_CHANNEL
  - A mensagem não pôde ser colocada em sua fila de destino

    A razão é qualquer código de razão MQRC_* que possa ser retornado por uma operação MQPUT
  - Uma saída de usuário solicitou essa ação

    O código de razão é aquele fornecido pela saída de usuário ou o padrão MQRC_SUPPRESSED_BY_EXIT
- Tentar encaminhar a mensagem a seu destino desejado, quando isso for possível.
- Reter a mensagem por um determinado período de tempo antes de descartar quando a razão para o desvio for determinada, mas não imediatamente corrigível.
- Fornecer instruções para administradores corrigirem problemas quando esses tiverem sido determinados.
- Descartar mensagens corrompidas ou que não podem ser processadas.

Há duas maneiras de lidar com as mensagens recuperadas da fila de mensagens não entregues:

1. Se a mensagem for para uma fila local:
   - Execute quaisquer conversões de código necessárias para extrair os dados do aplicativo
   - Execute conversões de código nesses dados se essa for uma função local
   - Coloque a mensagem resultante na fila local com todos os detalhes do descritor de mensagem restaurados
2. Se a mensagem for para uma fila remota, coloque a mensagem na fila.

Para obter informações sobre como as mensagens não entregues são manipuladas em um ambiente de enfileiramento distribuído, consulte O que acontece quando uma mensagem não pode ser entregue?.

Programação multicast

Use essas informações para aprender sobre as tarefas de programação do Multicast IBM MQ como se conectar a um gerenciador de filas e ao relatório de exceção.

O IBM MQ Multicast foi projetado para ser o mais transparente possível com o usuário e ainda ser compatível com aplicativos existentes. Definição de um objeto COMMINFO e configuração dos parâmetros do objeto TOPIC **MCAST** e **COMMINFO** significa que os aplicativos IBM MQ existentes não requerem regravação substancial para usar o multicast. No entanto, pode haver algumas limitações (consulte “Multicast e o MQI” na página 1056 para obter mais informações) e alguns problemas de segurança a serem considerados (consulte segurança do Multicast para obter mais informações).

Multicast e o MQI

Use estas informações para entender os conceitos principais do Message Queue Interface (MQI) e como eles se relacionam com o IBM MQ Multicast.

As assinaturas Multicast não são duráveis. Uma vez que não há filas físicas envolvidas, não há lugar para armazenar as mensagens off-line que são criadas pelas assinaturas duráveis.

Após um aplicativo ter inscrito um tópico multicast, ele recebe de volta um manipulador de objetos que ele pode consumir ou do qual pode fazer MQGET, como se fosse um manipulador para uma fila. Isso significa que apenas assinaturas multicast gerenciadas (assinaturas criadas com MQSO_MANAGED) são suportadas, ou seja, não é possível fazer uma assinatura e 'apontar' as mensagens em uma fila. Isso significa que as mensagens devem ser consumidas a partir da manipulação de objetos retornada na chamada de assinatura. No cliente, as mensagens são armazenadas em um buffer de mensagem até que sejam consumidas pelo cliente; veja Sub-rotina MessageBuffer do arquivo de configuração do cliente para obter mais informações. Se o cliente não acompanhar a taxa de publicação, as mensagens serão descartadas, conforme necessário, com as mensagens mais antigas primeiro descartados.

Normalmente é uma decisão administrativa se um aplicativo usa multicast ou não, o que é especificado pela configuração do atributo MCAST de um objeto TOPIC. Se um aplicativo de publicação deve se certificar de que multicast não é usado, ele pode usar a opção `MQOO_NO_MULTICAST`. Da mesma forma, um aplicativo de assinatura pode garantir que o multicast não seja usado ao assinar com a opção `MQSO_NO_MULTICAST`.

OIBM MQ Multicast suporta o uso de seletores de mensagens. Um seletor é usado por um aplicativo para registrar seu interesse apenas naquelas mensagens com propriedades que satisfazem a consulta SQL92 que a sequência de seleção representa. Para obter mais informações sobre os seletores de mensagem, veja “Seletores” na página 31.

A tabela a seguir lista todos os principais conceitos de MQI e como eles se relacionam com a Multicast:

Tabela 154. conceitos de MQI e como eles se relacionam com multicast

Conceito de MQI	**Ação ao tentar usar multicast**	**Código de razão**
Colocando uma mensagem de comprimento zero	Rejeitado	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Agrupamento	Rejeitado	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentação	Rejeitado	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
Listas de distribuição	Rejeitado	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	Indicadores para tópicos rejeitados: MQINQ e MQSET de tópicos não são suportado.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	Aceito para manipulador gerenciado. Apenas `Profundidade atual` pode ser consultada.	• Se o valor for de Profundidade Atual, então não há código de razão aplicável. • Se o valor for qualquer outro além da Profundidade Atual, o código de razão é 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR.
MQSET	Rejeitado para todos os identificadores.	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET

Tabela 154. conceitos de MQI e como eles se relacionam com multicast (continuação)

Conceito de MQI	Ação ao tentar usar multicast	Código de razão
Transações (XA ou não)	Rejeitado	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
Mensagem procura	Rejeitado	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
mensagens de bloqueio	Rejeitado	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Procurar com marca	Rejeitado	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Transmitir contexto	Rejeitado	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	Rejeitado. É inválido para tentar e MQPUT1 para um único tópico multicast.	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
assinatura durável	Rejeitado se o tópico está marcado como "multicast apenas", caso contrário, uma assinatura não multicast é feita.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	Rejeitado. Se a sequência do tópico for maior que 255 caracteres, ela é rejeitada no cliente.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
assinatura não gerenciada feita	Rejeitado se o tópico está marcado como "multicast apenas", caso contrário, uma assinatura não multicast é feita.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	Rejeitado	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

Os seguintes itens expandem-se em alguns dos conceitos MQI da tabela anterior e fornecem informações sobre alguns dos conceitos MQI que não estão na tabela:

Persistência de mensagem
Para assinantes não duráveis de multicast, as mensagens persistentes do publicador são entregues em um modo irrecuperável.

Truncamento da Mensagem
truncamento da mensagem é suportado, o que significa que é possível para um aplicativo:

1. Emita uma chamada MQGET.
2. Obter MQRC_TRUNCATED_MSG_FAILED.
3. Aloque um buffer maior.
4. Emita novamente a chamada MQGET para recuperar a mensagem.

expiração de assinatura
A expiração da assinatura não é suportada. Qualquer tentativa de configurar uma expiração é ignorada.

Alta disponibilidade para multicast

Use estas informações para entender a operação contínua de ponto a ponto do IBM MQ Multicast; embora IBM MQ se conecta a um gerenciador de filas do IBM MQ, as mensagens não fluem através desse gerenciador de filas.

Embora uma conexão com um gerenciador de filas deve ser feita para MQOPEN ou MQSUB, o objeto do tópico de multicast e as próprias mensagens não fluem através do gerenciador de filas. Portanto, após o MQOPEN ou MQSUB ser concluído no objeto de tópico de multicast, será possível continuar a transmitir mensagens multicast, mesmo se a conexão com o gerenciador de filas estiver sido perdida. Existem dois modos de operação:

Uma conexão normal é feita para o gerenciador de filas
A comunicação multicast será possível enquanto a conexão com o gerenciador de filas existir. Se a conexão falhar, as regras de MQI normais são aplicadas, por exemplo; uma MQPUT para o identificador de objeto multicast retorna 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

Uma conexão do cliente de reconexão é estabelecida com o gerenciador de filas
A comunicação multicast é possível mesmo durante o ciclo de reconexão. Isso significa que, mesmo quando a conexão com o gerenciador de filas foi interrompida, a colocação e o consumo de mensagens multicast não serão afetados. O cliente tentará se reconectar a um gerenciador de filas e se essa reconexão falhar, a manipulação de conexões será interrompida e todas as chamadas MQI, incluindo os multicast, falharão. Para obter mais informações, consulte: Reconexão automática do cliente

Se algum aplicativo emitir explicitamente um MQDISC, então todas as assinaturas de multicast e identificadores de objetos serão fechados.

Operação contínua ponto a ponto do multicast

Uma das vantagens da comunicação ponto a ponto entre os clientes é que as mensagens não precisam fluir através do gerenciador de filas; portanto, se a conexão com o gerenciador de filas for interrompida, a transferência de mensagem continuará. As restrições a seguir se aplicam aos requisitos de mensagem contínua deste modo:

- A conexão deve ser feita usando uma das opções MQCNO_RECONNECT_* para operação contínua. Esse processo significa que, embora a sessão de comunicações possa ser interrompida, a manipulação de conexões real não será interrompida e, em vez disso, estará no estado de reconexão. Se a reconexão falhar, a manipulação de conexões agora será interrompida com isso evitará todas as chamadas MQI adicionais.
- Apenas MQPUT, MQGET, MQINQ e Consumo Assíncrono são suportados neste modo. Quaisquer verbos MQOPEN, MQCLOSE ou MQDISC requerem a reconexão com o gerenciador de filas para serem concluídos.
- O status flui para a parada do gerenciador de filas; qualquer estado no gerenciador de filas pode, portanto, ser antigo ou ausente. Isso significa que os clientes podem estar enviando e recebendo mensagens e não há status conhecido no gerenciador de filas. Para obter mais informações, consulte: Monitoramento de aplicativo multicast

Conversão de dados no MQI para o sistema de mensagens multicast

Use essas informações para entender como a conversão de dados funciona para o sistema de mensagens multicast do IBM MQ.

IBM MQ Multicast é um protocolo sem conexão, compartilhado e, portanto, não é possível que cada cliente faça solicitações específicas para a conversão de dados. Cada cliente inscrito ao mesmo fluxo multicast receberá os mesmos dados binários; portanto, se a conversão de dados do IBM MQ for necessária, a conversão será executada localmente em cada cliente.

Os dados são convertidos no cliente para o tráfego do IBM MQ Multicast. Se a opção **MQGMO_CONVERT** for especificada, a conversão de dados será feita conforme solicitada. Os formatos definidos pelo usuário precisam da saída de conversão de dados instalada no cliente; consulte "Escrevendo saídas de conversão de dados" na página 995 para obter informações sobre quais bibliotecas estão agora nos pacotes do cliente e do servidor.

Para obter informações sobre a administração de conversão de dados, consulte Ativando a conversão de dados para o sistema de mensagens do Multicast.

Para obter mais informações sobre a conversão de dados, consulte Conversão de dados.

Para obter mais informações sobre saídas de conversão de dados e `ClientExitPath`, consulte ClientExitPath subrotina do arquivo de configuração do cliente.

Relatório de exceção do Multicast

Use essas informações para aprender sobre manipuladores de eventos do IBM MQ Multicast e relatório de exceções do IBM MQ Multicast.

OIBM MQ Multicast ajuda com determinação de problema chamando o manipulador de eventos para relatar eventos multicast que são relatados usando o mecanismo do manipulador de eventos padrão do IBM MQ.

Um evento individual do Multicast pode resultar em mais de um evento do IBM MQ ser chamado porque pode haver várias manipulações de conexões `MQHCONN` usando o mesmo transmissor ou receptor multicast. No entanto, cada exceção multicast faz com que apenas um manipulador de eventos seja chamado por conexão do IBM MQ.

A constante do IBM MQ `MQCBDO_EVENT_CALL` permite que aplicativos registrem um retorno de chamada para receber somente eventos do IBM MQ e o `MQCBDO_MC_EVENT_CALL` permite que os aplicativos registrem um retorno de chamada para receber somente eventos multicast. Se ambas as constantes forem usadas, ambos os tipos de eventos serão recebidos.

Solicitando eventos do Multicast

Os eventos do IBM MQ Multicast usam a constante `MQCBDO_MC_EVENT_CALL` no campo `cbd.Options`. O exemplo a seguir demonstra como solicitar eventos multicast:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

Quando a opção `MQCBDO_MC_EVENT_CALL` é especificada para o campo `cbd.Options`, somente eventos do IBM MQ Multicast são enviados ao manipulador de eventos, em vez de eventos no nível da conexão. Para solicitar que ambos os tipos de eventos sejam enviados ao manipulador de eventos, o aplicativo deve especificar a constante `MQCBDO_EVENT_CALL` no campo `cbd.Options`, bem como a constante `MQCBDO_MC_EVENT_CALL` conforme mostrado no exemplo a seguir:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

Se nenhuma dessas constantes for usada, somente eventos no nível da conexão serão enviadas ao manipulador de eventos.

Para obter mais informações sobre valores para o campo `Options`, consulte Opções (MQLONG).

Formato de evento multicast

As exceções do IBM MQ Multicast incluem algumas informações de suporte que são retornadas no parâmetro **`Buffer`** da função de retorno de chamada. O ponteiro **`Buffer`** aponta para uma matriz de ponteiros e o campo `MQCBC.DataLength` especifica o tamanho, em bytes, da matriz. O primeiro elemento da matriz sempre aponta para uma descrição de texto curta do evento. Mais parâmetros podem ser fornecidos dependendo do tipo de evento. A tabela a seguir lista as exceções:

Tabela 155. Descrições de códigos de eventos multicast

Código de evento	Descrição	Dados adicionais
MQMCEV_PACKET_LOSS	Perda de pacote irrecuperável	Número de pacotes perdidos
MQMCEV_HEARTBEAT_TIMEOUT	Ausência longa do pacote de controle de pulsação	N/D
MQMCEV_VERSION_CONFLICT	Recepção de pacotes mais novos de versão de protocolo	N/D
MQMCEV_RELIABILITY	Diferentes modos de confiabilidade do transmissor e do receptor	N/D
MQMCEV_CLOSED_TRANS	A transmissão do tópico é fechada por uma origem	N/D
MQMCEV_STREAM_ERROR	Erro detectado no fluxo	N/D
MQMCEV_NEW_SOURCE	Uma nova origem inicia a transmissão no tópico	Estrutura da origem
MQMCEV_RECEIVE_QUEUE_TRIMMED	Pacotes removidos de PacketQ devido à expiração de tempo ou espaço	Número de pacotes aparados
MQMCEV_PACKET_LOSS_NACK_EXPIRE	Perda de pacote irrecuperável devido à expiração de NACK	Número de pacotes perdidos
MQMCEV_ACK_RETRIES_EXCEEDED	Pacotes removidos do histórico após **`max_ack_retries`** ter sido excedido	Número de pacotes removidos
MQMCEV_STREAM_SUSPEND_NACK	NACKs foram suspensas em um fluxo aceito por este tópico	Suspender ID do fluxo Tempo em milissegundos que o fluxo está suspenso
MQMCEV_STREAM_RESUME_NACK	NACKs foram continuadas após terem sido suspensas em um fluxo	ID do fluxo
MQMCEV_STREAM_EXPELLED	Um fluxo aceito por este tópico foi rejeitado devido a uma solicitação de expulsão	ID do fluxo
MQMCEV_FIRST_MESSAGE	Primeira mensagem de uma origem	Número da mensagem

Tabela 155. Descrições de códigos de eventos multicast (continuação)

Código de evento	Descrição	Dados adicionais
MQMCEV_LATE_JOIN_FAILURE	Falha ao iniciar sessão se associação postergada	N/D
MQMCEV_MESSAGE_LOSS	Perda de mensagem irrecuperável	Número de mensagens perdidas
MQMCEV_SEND_PACKET_FAILURE	O transmissor multicast falhou ao enviar um pacote multicast	N/D
MQMCEV_REPAIR_DELAY	O receptor multicast não recebeu um pacote de reparo para um NAK pendente	N/D
MQMCEV_MEMORY_ALERT_ON	Buffers de recepção do receptor estão ficando cheios	Porcentagem de utilização do buffer pool
MQMCEV_MEMORY_ALERT_OFF	Buffers de recepção do receptor voltaram ao normal	Porcentagem de utilização do buffer pool
MQMCEV_NACK_ALERT_ON	A taxa de solicitações de pacote de reparo do receptor atingiu a marca d'água alta	A taxa de solicitações de reparo atual em pacotes por segundo
MQMCEV_NACK_ALERT_OFF	A taxa de solicitações de pacote de reparo do receptor voltou ao normal	A taxa de solicitações de reparo atual em pacotes por segundo
MQMCEV_REPAIR_ALERT_ON	A taxa de envio de pacote de reparo do transmissor atingiu a marca d'água alta	N/D
MQMCEV_REPAIR_ALERT_OFF	A taxa de envio de pacote de reparo do transmissor voltou ao normal	N/D
MQMCEV_SHM_DEST_UNUSABLE	A região de Memória compartilhada usada por um destino de tópico transmissor foi detectado como inutilizado	N/D
MQMCEV_SHM_PORT_UNUSABLE	A porta de Memória compartilhada usada por uma instância do receptor foi detectada como inutilizada	N/D
MQMCEV_CCT_GETTIME_FAILED	O tempo de get do Tempo do cluster coordenado falhou	N/D
MQMCEV_DEST_INTERFACE_FAILURE	A interface de rede usada por um destino de tópico do transmissor falhou e uma interface de rede de backup está indisponível	
MQMCEV_DEST_INTERFACE_FAILOVER	A interface de rede usada por um destino de tópico do transmissor falhou e um failover bem-sucedido para outra interface foi concluído	

Tabela 155. Descrições de códigos de eventos multicast (continuação)		
Código de evento	**Descrição**	**Dados adicionais**
MQMCEV_PORT_INTERFACE-FAILURE	A interface de rede usada por rmmPort de um receptor falhou e uma interface de rede de backup está indisponível (ou também falhou)	configuração de RMM
MQMCEV_PORT_INTERFACE_FAILOVER	A interface de rede usada por rmmPort de um receptor falhou e um failover bem-sucedido para outra interface foi concluído	configuração de RMM

Codificação em C

Observe as informações nas seções a seguir ao codificar programas IBM MQ em C.

Parâmetros das chamadas MQI

Os parâmetros que são *somente entrada* e do tipo MQHCONN, MQHOBJ, MQHMSG ou MQLONG são passados por valor; para todos os outros parâmetros, o *endereço* do parâmetro é passado por valor.

Nem todos os parâmetros que são passados por endereço precisam ser especificados toda vez que uma função for chamada. Quando um parâmetro específico não for necessário, um ponteiro nulo pode ser especificado como o parâmetro na chamada de função, no lugar do endereço dos dados do parâmetro. Parâmetros para os quais isso é possível estão identificados nas descrições de chamada.

Nenhum parâmetro é retornado como o valor da função; na terminologia de C, isso significa que todas as funções retornam nulo.

Os atributos da função são definidos pela variável de macro MQENTRY; o valor dessa variável de macro depende do ambiente.

Parâmetros com o tipo de dados indefinido

As funções MQGET, MQPUT e MQPUT1 têm, cada uma delas, um parâmetro **`Buffer`** que tem um tipo de dados indefinido. Esse parâmetro é usado para enviar e receber os dados da mensagem do aplicativo.

Parâmetros desse tipo são mostrados nos exemplos de C como matrizes de MQBYTE. É possível declarar os parâmetros dessa maneira, mas geralmente é mais conveniente declará-los como a estrutura que descreve o layout dos dados na mensagem. O parâmetro da função é declarado como um ponteiro para nulo e, portanto, o endereço de quaisquer dados pode ser especificado como o parâmetro na chamada de função.

Tipos de dados

Todos os tipos de dados são definidos com a instrução `typedef`.

Para cada tipo de dados, o tipo de dados do ponteiro correspondente também é definido. O nome do tipo de dados do ponteiro é o nome do tipo de dados elementar ou de estrutura com o prefixo P para denotar um ponteiro. Os atributos do ponteiro são definidos pela variável de macro MQPOINTER; o valor dessa variável de macro depende do ambiente. O código a seguir ilustra como declarar tipos de dados do ponteiro:

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

Manipulando sequências binárias

Sequências de dados binários são declaradas como um dos tipos de dados MQBYTEn.

Sempre que copiar, comparar ou configurar campos desse tipo, use as funções C `memcpy`, `memcmp`ou `memset`:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

Não use as funções de sequência strcpy, strcmp, strncpy ou strncmp, pois elas não funcionam corretamente com os dados declarados como MQBYTE24.

Manipulação de sequências de caracteres

Quando o gerenciador de filas retornar dados de caracteres para o aplicativo, o gerenciador de filas sempre preencherá os dados de caracteres com espaços em branco para o comprimento definido do campo. O gerenciador de filas não retorna sequências terminadas em nulo, mas é possível usá-las em sua entrada. Portanto, ao copiar, comparar concatenar essas sequências, use as funções de sequência strncpy, strncmp ou strncat.

Não use as funções de sequência que requerem a sequência a ser finalizada por um nulo (strcpy, strcmp e strxfrm). Além disso, não use a função strlen para determinar o comprimento da sequência; use em vez disso a função sizeof para determinar o comprimento do campo.

Valores iniciais para estruturas

O arquivo include <cmqc.h> define várias variáveis de macro que podem ser usadas para fornecer valores iniciais para as estruturas ao declarar instâncias dessas estruturas. Essas variáveis de macro têm nomes no formato MQxxx_DEFAULT, em que MQxxx representa o nome da estrutura. Use-as desta forma:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

Para alguns campos de caracteres, a MQI define valores específicos válidos (por exemplo, para os campos *StrucId* ou para o campo *Format* no MQMD). Para cada um dos valores válidos, duas variáveis de macro são fornecidas:

- Uma variável de macro define o valor como uma sequência com um comprimento, excluindo o nulo implícito, que corresponda exatamente ao comprimento definido do campo. Nos exemplos a seguir, o símbolo ¬ representa um único caractere em branco:

```
#define MQMD_STRUC_ID "MD¬¬"
#define MQFMT_STRING "MQSTR¬¬¬"
```

  Use esse formato com as funções memcpy e memcmp.
- A outra variável de macro define o valor como uma matriz de char; o nome dessa variável de macro é o nome da forma de sequência com o sufixo _ARRAY. Por exemplo:

```
#define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
#define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
```

  Use esse formato para inicializar o campo quando uma instância da estrutura for declarada com valores diferentes dos fornecidos pela variável de macro MQMD_DEFAULT.

Valores iniciais para as estruturas dinâmicas

Quando um número variável de instâncias de uma estrutura é necessário, as instâncias são geralmente criadas no armazenamento principal obtido dinamicamente usando as funções calloc ou malloc.

Para inicializar os campos nessas estruturas, a técnica a seguir é recomendada:

1. Declare uma instância da estrutura usando a variável de macro MQxxx_DEFAULT apropriada para inicializar a estrutura. Esta instância se torna o *modelo* para outras instâncias:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                     /* declare model instance */
   ```

   Codifique as palavras-chave static e auto na declaração para fornecer à instância modelo tempo de vida estático ou dinâmico, conforme necessário.
2. Use as funções calloc ou malloc para obter armazenamento para uma instância dinâmica da estrutura:

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                     /* get storage for dynamic instance */
   ```
3. Use a função memcpy para copiar a instância modelo para a instância dinâmica:

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                     /* initialize dynamic instance */
   ```

Uso de C++

Para a linguagem de programação C++, os arquivos de cabeçalho contêm as instruções adicionais a seguir que são incluídas somente quando um compilador C++ é usado:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Codificação no Visual Basic

Informações a serem consideradas ao codificar programas IBM MQ no Microsoft Visual Basic. O Visual Basic é suportado somente no Windows.

Nota:

Stabilized No IBM WebSphere MQ 7.0, fora do ambiente do .NET, o suporte para o Visual Basic (VB) foi estabilizado no nível IBM WebSphere MQ 6.0. A maior parte das novas funções incluídas no IBM WebSphere MQ 7.0 ou mais recente não está disponível para aplicativos VB. Se você estiver programando no VB.NET, use as classes do IBM MQ para o .NET. Para obter mais informações, consulte Desenvolvendo aplicativos .NET.

Deprecated No IBM MQ 9.0, o suporte para o Microsoft Visual Basic 6.0 foi descontinuado. As classes do IBM MQ para .NET são a tecnologia de substituição recomendada.

Para evitar a conversão indesejada de dados binários passando entre o Visual Basic e o IBM MQ, use uma definição MQBYTE em vez de MQSTRING. CMQB.BAS define vários tipos MQBYTE novos que são equivalentes a uma definição de byte C e usa essas estruturas em IBM MQ. Por exemplo, para a estrutura MQMD (descritor de mensagens), MsgId (identificador de mensagem) é definido como MQBYTE24.

O Visual Basic não possui um tipo de dados de ponteiro, portanto as referências a outras estruturas de dados do IBM MQ são por deslocamento em vez de ponteiro. Declare uma estrutura composta que consiste das duas estruturas de componente e especifique a estrutura composta na chamada. O suporte do IBM MQ para o Visual Basic fornece uma chamada MQCONNXAny para tornar isso possível e permitir que os aplicativos clientes especifiquem as propriedades de canal em uma conexão do cliente. Ele aceita uma estrutura sem tipo (MQCNOCD) no lugar da estrutura MQCNO típica.

A estrutura MQCNOCD é uma estrutura composta que consiste em um MQCNO seguido por um MQCD. Esta estrutura é declarada no arquivo de cabeçalho saídas CMQXB. Use a rotina MQCNOCD_DEFAULTS para inicializar uma estrutura MQCNOCD. Uma amostra realizando chamadas MQCONNX fornecida (amqscnxb.vbp).

MQCONNXAny possui os mesmos parâmetros que MQCONNX, exceto o fato de que o parâmetro **ConnectOpts** é declarado como sendo do tipo qualquer tipo de dados em vez de dados de tipo MQCNO. Isso permite que a função aceite tanto a estrutura MQCNO como a MQCNOCD. Essa função é declarada no arquivo de cabeçalho principal CMQB.

Conceitos relacionados

“Preparando programas Visual Basic no Windows” na página 1032
Informações a serem consideradas ao usar programas Microsoft Visual Basic no Windows.

Referências relacionadas

“Vinculando aplicativos Visual Basic ao código do IBM MQ MQI client” na página 932
É possível vincular os aplicativos Microsoft Visual Basic com o código IBM MQ MQI client no Windows.

Codificação em COBOL

Observe as informações na seção a seguir ao codificar programas IBM MQ em COBOL.

Constantes nomeadas

Os nomes de constantes são mostrados contendo o caractere sublinhado (_) como parte do nome. Em COBOL, deve-se usar o caractere de hífen (-) no lugar do sublinhado. As constantes que possuem valores de sequência de caracteres usam o caractere de aspas simples (') como o delimitador de sequência. Para fazer o compilador aceitar esse caractere, use a opção do compilador APOST.

O arquivo de cópia CMQV contém declarações de constantes nomeadas como itens de nível 10. Para usar as constantes, declare o item nível 01 explicitamente, em seguida, use a instrução COPY para copiar nas declarações das constantes:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

No entanto, este método faz com que as constantes ocupem o armazenamento no programa, mesmo se elas não forem referidas. Se as constantes forem incluídas em muitos programas separados dentro da mesma unidade de execução, diversas cópias de constantes existirão; isso pode resultar em uma quantidade significativa de armazenamento principal que está sendo usado. É possível evitar isso incluindo a cláusula GLOBAL na declaração de nível 01:

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

Isso aloca armazenamento para apenas *um* conjunto de constantes dentro da unidade de execução; as constantes, no entanto, podem ser referenciadas por *qualquer* programa na unidade de execução, não apenas o programa que contém a declaração nível 01.

Assegurando o alinhamento da estrutura

Deve-se tomar cuidado para assegurar que as estruturas do IBM MQ que são transmitidas para iniciar nas chamadas do MQ devem ser alinhadas nos limites de palavra. Um limite de palavra é 4 bytes para processos de 32 bits, 8 bytes para processos de 64 bits e 16 bytes para processos de 128 bits (IBM i).

Quando possível, coloque todas as estruturas do IBM MQ juntas para que sejam todas alinhadas por limite.

Coding in System/390 assembler language (Message queue interface)

Note the information in the following sections when coding IBM MQ for z/OS programs in assembler language.

Names

The names of parameters in the descriptions of calls, and the names of fields in the descriptions of structures are shown in mixed case. In the assembler-language macros supplied with IBM MQ, all names are in uppercase.

Using the MQI calls

The MQI is a call interface, so assembler-language programs must observe the OS linkage convention.

In particular, before they issue an MQI call, assembler-language programs must point register R13 at a save area of at least 18 full words. This save area provides storage for the called program. It stores the registers of the caller before their contents are destroyed, and restores the contents of the caller's registers on return.

Note: This is important for CICS assembler-language programs that use the DFHEIENT macro to set up their dynamic storage, but that choose to override the default DATAREG from R13 to other registers. When the CICS Resource Manager Interface receives control from the stub, it saves the current contents of the registers at the address to which R13 is pointing. Failing to reserve a save area for this purpose gives unpredictable results, and will probably cause an abend in CICS.

Declaring constants

Most constants are declared as equates in macro CMQA.

However, the following constants cannot be defined as equates, and these are not included when you call the macro using default options:

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

To include them, add the keyword EQUONLY=NO when you call the macro.

CMQA is protected against multiple declaration, so you can include it many times. However, the keyword EQUONLY takes effect only the first time that the macro is included.

Specifying the name of a structure

To allow more than one instance of a structure to be declared, the macro that generates the structure prefixes the name of each field with a user-specifiable string and an underscore character (_).

Specify the string when you invoke the macro. If you do not specify a string, the macro uses the name of the structure to construct the prefix:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

The structure declarations in Call descriptions show the default prefix.

Specifying the form of a structure

The macros can generate structure declarations in one of two forms, controlled by the DSECT parameter:

DSECT=YES
An assembler-language DSECT instruction is used to start a new data section; the structure definition immediately follows the DSECT statement. No storage is allocated, so no initialization is possible. The label on the macro invocation is used as the name of the data section; if no label is specified, the name of the structure is used.

DSECT=NO
Assembler-language DC instructions are used to define the structure at the current position in the routine. The fields are initialized with values, which you can specify by coding the relevant parameters on the macro invocation. Fields for which no values are specified on the macro invocation are initialized with default values.

DSECT=NO is assumed if the DSECT parameter is not specified.

Controlling the listing

You can control the appearance of the structure declaration in the assembler-language listing with the LIST parameter:

LIST=YES
The structure declaration appears in the assembler-language listing.

LIST=NO
The structure declaration does not appear in the assembler-language listing. This is assumed if the LIST parameter is not specified.

Specifying initial values for fields

You can specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message descriptor structure with the *MsgType* field initialized with MQMT_REQUEST, and the *ReplyToQ* field initialized with the string MY_REPLY_TO_QUEUE, use the following code:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

If you specify a named constant (or equate) as a value on the macro invocation, use the CMQA macro to define the named constant. You must not enclose in single quotation marks (' ') values that are character strings.

Writing reenterable programs

IBM MQ uses its structures for both input and output. If you want your program to remain reenterable:

1. Define working storage versions of the structures as DSECTs, or define the structures inline within an already-defined DSECT. Then copy the DSECT to storage that is obtained using:
   - For batch and TSO programs, the STORAGE or GETMAIN z/OS assembler macros
   - For CICS, the working storage DSECT (DFHEISTG) or the EXEC CICS GETMAIN command

   To correctly initialize these working storage structures, copy a constant version of the corresponding structure to the working storage version.

   Note: The MQMD and MQXQH structures are each more than 256 bytes long. To copy these structures to storage, use the MVCL assembler instruction.
2. Reserve space in storage by using the LIST form (MF=L) of the CALL macro. When you use the CALL macro to make an MQI call, use the EXECUTE form (MF=E) of the macro, using the storage reserved earlier, as shown in the example under “Using CEDF” on page 1069. For more examples of how to do this, see the assembler language sample programs as shipped with IBM MQ.

Use the assembler language RENT option to help you to determine if your program is reenterable.

For information on writing reenterable programs, see *z/OS MVS Application Development Guide: Assembler Language Programs*.

Using CEDF

If you want to use the CICS-supplied transaction, CEDF (CICS Execution Diagnostic Facility) to help you to debug your program, add the ,VL keyword to each CALL statement, for example:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

The previous example is reenterable assembler-language code where PARMAREA is an area in the working storage that you specified.

Using the MQI calls

The MQI is a call interface, so assembler-language programs must observe the OS linkage convention. In particular, before they issue an MQI call, assembler-language programs must point register R13 at a save area of at least 18 full words. This save area provides storage for the called program. It stores the registers of the caller before their contents are destroyed, and restores the contents of the caller's registers on return.

Note: This is important for CICS assembler-language programs that use the DFHEIENT macro to set up their dynamic storage, but that choose to override the default DATAREG from R13 to other registers. When the CICS Resource Manager Interface receives control from the stub, it saves the current contents of the registers at the address to which R13 is pointing. Failing to reserve a proper save area for this purpose gives unpredictable results, and will probably cause an abend in CICS.

IBM i Codificando programas IBM MQ em RPG (IBM i somente)

Na documentação do IBM MQ, os parâmetros de chamadas, os nomes de tipos de dados, os campos de estruturas e os nomes de constantes são todos descritos usando seus nomes longos. Em RPG, esses nomes são abreviados para seis ou menos caracteres maiúsculos.

Por exemplo, o campo *MsgType* se torna *MDMT* em RPG. Para obter mais informações, consulte a IBM i Referência de Programação do Aplicativo (ILE/RPG).

Coding in PL/I (z/OS only)

Useful information when coding for IBM MQ in PL/I.

Structures

Structures are declared with the BASED attribute, and so do not occupy any storage unless the program declares one or more instances of a structure.

An instance of a structure can be declared using the `like` attribute, for example:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

The structure fields are declared with the INITIAL attribute; when the `like` attribute is used to declare an instance of a structure, that instance inherits the initial values defined for that structure. You need to set only those fields where the value required is different from the initial value.

PL/I is not sensitive to case, and so the names of calls, structure fields, and constants can be coded in lowercase, uppercase, or mixed case.

Named constants

The named constants are declared as macro variables; as a result, named constants that are not referred to by the program do not occupy any storage in the compiled procedure.

However, the compiler option that causes the source to be processed by the macro preprocessor must be specified when the program is compiled.

All the macro variables are character variables, even the ones that represent numeric values. Although this might seem counter intuitive, it does not result in any data-type conflict after the macro variables have been substituted by the macro processor, for example:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

Usando os programas processuais de amostra do IBM MQ

Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

Sobre esta tarefa

Há dois conjuntos de amostras:

- Multi Programas de amostra para Multiplataformas
- z/OS Programas de amostra para z/OS.

Procedimento

- Use os seguintes links para descobrir mais sobre os programas de amostra:
  - Multi "Usando os programas de amostra em multiplataformas" na página 1071
  - z/OS "Using the sample programs for z/OS" na página 1176

Conceitos relacionados

"Conceitos de desenvolvimento de aplicativos" na página 7
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Antes de começar a projetar e escrever seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ.

"Desenvolvendo aplicativos para o IBM MQ" na página 5
É possível desenvolver aplicativos para enviar e receber mensagens e para gerenciar gerenciadores de fila e recursos relacionados. O IBM MQ suporta aplicativos escritos em muitas linguagens e estruturas diferentes.

"Considerações de design para aplicativos IBM MQ" na página 50
Quando tiver decidido como seus aplicativos podem tirar proveito das plataformas e ambientes disponíveis para você, será necessário decidir como usar os recursos oferecidos pelo IBM MQ.

"Escrevendo um aplicativo processual para enfileiramento" na página 732
Use essas informações para aprender como escrever aplicativos de enfileiramento, conectar e desconectar de um gerenciador de filas, publicar/assinar e abrir e fechar objetos.

"Escrevendo aplicativos clientes processuais" na página 924
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

"Escrevendo aplicativos de publicar/assinar" na página 821

Inicie a escrever aplicativos de publicar/assinar IBM MQ.

“Construindo um aplicativo processual” na página 1012
É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

“Manipulando erros de programa processual” na página 1050
Estas informações explicarão os erros associados às chamadas MQI de aplicativos, ao realizar uma chamada ou quando sua mensagem for entregue para seu destino final.

Multi Usando os programas de amostra em multiplataformas

Estes programas processuais de amostra são entregues com o produto. As amostras são escritas em C e em COBOL, e demonstram os usos típicos do Message Queue Interface (MQI).

Sobre esta tarefa

As amostras não são destinados a demonstrar técnicas de programação geral, portanto, alguma verificação de erro que você queira incluir em um programa de produção será omitida.

O código de origem para todas as amostras é fornecido com o produto; esta origem inclui comentários que explicam as técnicas de enfileiramento de mensagens demonstradas nos programas.

IBM i Para programação de RPG, consulte IBM i Application Programming Reference (ILE/RPG).

Os nomes das amostras de iniciam com o prefixo `amq`. O quarto caractere indica a linguagem de programação e o compilador, onde necessário:

- s: idioma C
- 0: idioma COBOL em compiladores IBM e Micro Focus
- i: idioma COBOL somente em compiladores IBM
- m: idioma COBOL somente em compiladores Micro Focus

O oitavo caractere do executável indica se a amostra é executada no modo cliente ou no modo de ligação local. Se não houver um oitavo caractere, então, a amostra será executada no modo de ligações locais. Se o oitavo caractere for 'c', então, a amostra será executada no modo cliente.

Para poder executar os aplicativos de amostra, deve-se primeiro criar e configurar um gerenciador de filas. Para configurar o gerenciador de filas para aceitar conexões do cliente, veja “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081.

Procedimento

- Use os seguintes links para descobrir mais sobre os programas de amostra:
  - “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072
  - “Preparando e executando os programas de amostra” na página 1081
  - “O programa de amostra de saída de API” na página 1089
  - “O programa de amostra Asynchronous Consumption” na página 1089
  - “O programa de amostra Asynchronous Put” na página 1091
  - “Os programas de amostra Browse” na página 1091
  - “O programa de amostra Browser” na página 1093
  - “A transação de amostra do CICS” na página 1094
  - “O programa de amostra Connect” na página 1094
  - “O programa de amostra Data-Conversion” na página 1096
  - “Amostras de coordenação de banco de dados” na página 1096
  - “Amostra do manipulador de filas de mensagens não entregues” na página 1103

- “O programa de amostra Distribution List” na página 1103
- “Os programas de amostra Echo” na página 1104
- “Os programas de amostra Get” na página 1105
- “Programas de amostra de alta disponibilidade” na página 1107
- “Os programas de amostra Inquire” na página 1111
- “O programa de amostra Inquire Properties of a Message Handle” na página 1112
- “Os programas de amostra Publish/Subscribe” na página 1112
- “O programa de amostra Publish Exit” na página 1116
- “Os programas de amostra Put” na página 1117
- “Os programas de amostra Reference Message” na página 1119
- “Os programas de amostra Request” na página 1127
- “Os programas de amostra Set” na página 1133
- “O programa de amostra TLS” na página 1134
- “Os programas de amostra Triggering” na página 1137
- “Usando as amostras TUXEDO no AIX, Linux, and Windows” na página 1139
- “Usando a saída de segurança SSPI no Windows” na página 1149
- “Executando as amostras usando filas remotas” na página 1150
- “O programa de amostra de Cluster Queue Monitoring (AMQSCLM)” na página 1150
- “Programa de amostra para Connection Endpoint Lookup (CEPL)” na página 1160

Conceitos relacionados

“Programas de amostra C++” na página 540
Quatro programas de amostra são fornecidos, para demonstrar a obtenção e a colocação de mensagens.

Tarefas relacionadas

“Using the sample programs for z/OS” na página 1176
The sample procedural applications that are delivered with IBM MQ for z/OS demonstrate typical uses of the Message Queue Interface (MQI).

Multi *Recursos demonstrados nos programas de amostra em multiplataformas*

Uma coleção de tabelas que mostram as técnicas demonstradas pelos programas de amostra do IBM MQ.

Todas as amostras abrem e fecham filas usando as chamadas MQOPEN e MQCLOSE, portanto, essas técnicas não estão listados separadamente nas tabelas. Veja o título que inclui a plataforma na qual você está interessado.

z/OS Para a plataforma z/OS, consulte “Using the sample programs for z/OS” na página 1176.

Linux AIX *Amostras para sistemas AIX and Linux*

As técnicas demonstradas pelos programas de amostra para o IBM MQ for AIX or Linux.

Veja “Preparando e executando programas de amostra em AIX and Linux” na página 1085 para saber onde os programas de amostra do IBM MQ for AIX or Linux são armazenados.

Tabela 156 na página 1073 A tabela lista quais arquivos de origem C e COBOL são fornecidos e se um executável do servidor ou cliente é incluído.

Tabela 156. Programas de amostra que demonstram o uso do MQI (C e COBOL) no AIX and Linux.

Uma tabela com quatro colunas. A primeira coluna lista as técnicas demonstradas pelas amostras. A segunda coluna lista as amostras C e a terceira coluna lista as amostras COBOL que demonstram cada uma das técnicas listadas na primeira coluna. A quarta coluna mostra se um executável C do servidor está ou não incluído e a quinta coluna mostra se um executável C do cliente está ou não incluído.

Técnica	**C (origem) ("1" na página 1075)**	**COBOL (origem) ("2" na página 1075)**	**Servidor (executável C)**	**Cliente (executável em C)**
Usando a interface de publicação/assinatura	amqssbxa amqssuba amqspuba	sem amostra	amqssbx amqssub amqspub	sem amostra
Colocando mensagens usando a chamada MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Colocando uma única mensagem usando a chamada MQPUT1	amqsecha amqsinqa	amqiechx amqiinqx amqmechx amqminqx	amqsech amqsinq	amqsechc
Colocando mensagens em uma lista de distribuição ("3" na página 1075)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Respondendo a uma mensagem de solicitação	amqsinqa	amqiinqx amqminqx	amqsinq	sem amostra
Obtendo mensagens utilizando navegar (nenhuma espera)	amqsgbr0	amq0gbr0	amqsgbr	sem amostra
Obtendo mensagens (aguardar com um limite de tempo)	amqsget0	amq0get0	amqsget	amqsgetc
Obtendo mensagens (espera ilimitada)	amqstrg0	sem amostra	amqstrg	amqstrgc
Obtendo mensagens (com conversão de dados)	amqsecha	sem amostra	amqsech	sem amostra
Colocando mensagens de referência em uma fila ("3" na página 1075)	amqsprma	sem amostra	amqsprm	amqsprmc
Obtendo mensagens de referência de uma fila ("3" na página 1075)	amqsgrma	sem amostra	amqsgrm	amqsgrmc
Saída do canal de mensagem de referência ("3" na página 1075)	amqsxrma amqsqrma	sem amostra	amqsxrm	sem amostra
Procurando 20 primeiros caracteres de uma mensagem	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Procurando mensagens completas	amqsbcg0	sem amostra	amqsbcg	amqsbcgc
Usando uma fila de entrada compartilhada	amqsinqa	amqiinqx amqminqx	amqsinq	amqsinqc
Usando uma fila de entrada exclusiva	amqstrg0	amq0req0	amqstrg	amqstrgc
Usando a chamada MQINQ	amqsinqa	amqiinqx amqminqx	amqsinq	sem amostra
Usando a chamada MQSET	amqsseta	amqmsetx amqisetx	amqsset	amqssetc

Tabela 156. Programas de amostra que demonstram o uso do MQI (C e COBOL) no AIX and Linux.

Uma tabela com quatro colunas. A primeira coluna lista as técnicas demonstradas pelas amostras. A segunda coluna lista as amostras C e a terceira coluna lista as amostras COBOL que demonstram cada uma das técnicas listadas na primeira coluna. A quarta coluna mostra se um executável C do servidor está ou não incluído e a quinta coluna mostra se um executável C do cliente está ou não incluído.

(continuação)

Técnica	**C (origem) ("1" na página 1075)**	**COBOL (origem) ("2" na página 1075)**	**Servidor (executável C)**	**Cliente (executável em C)**
Usando uma fila de resposta	amqsreq0	amq0req0	amqsreq	amqsreqc
Solicitando exceções de mensagens	amqsreq0	amq0req0	amqsreq	sem amostra
Aceitando uma mensagem truncada	amqsgbr0	amq0gbr0	amqsgbr	sem amostra
Usando um nome de fila resolvido	amqsgbr0	amq0gbr0	amqsgbr	sem amostra
Acionando um processo	amqstrg0	sem amostra	amqstrg	amqstrgc
Usando a conversão de dados	("4" na página 1075)	sem amostra	sem amostra	sem amostra
IBM MQ (coordenando gerenciadores de banco de dados compatível com XA) acessando um banco de dados único usando SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	sem amostra	sem amostra
IBM MQ (coordenando gerenciadores de banco de dados compatíveis com XA) acessando dois bancos de dados usando SQL	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	sem amostra	sem amostra
Transação CICS ("5" na página 1075)	amqscic0.ccs	sem amostra	amqscic0	sem amostra
Transação Encina ("3" na página 1075)	amqsxae0	sem amostra	amqsxae0	sem amostra
Transação TUXEDO para colocar mensagens "6" na página 1075)	amqstxpx	sem amostra	sem amostra	sem amostra
Transação TUXEDO para obter mensagens ("6" na página 1075)	amqstxgx	sem amostra	sem amostra	sem amostra
Servidor para TUXEDO ("6" na página 1075)	amqstxsx	sem amostra	sem amostra	sem amostra
Manipulador da fila de devoluções	Diretório `./tools/c/Samples/dlq` ("7" na página 1075)	sem amostra	amqsdlq	sem amostra
A partir de um cliente MQI, a colocação de uma mensagem	sem amostra	sem amostra	sem amostra	amqsputc
A partir de um cliente MQI, obter uma mensagem	sem amostra	sem amostra	sem amostra	amqsgetc
Conectando-se ao gerenciador de filas usando MQCONNX	amqscnxc	sem amostra	sem amostra	amqscnxc
Usando saídas de API	amqsaxe0	sem amostra	amqsaxe	sem amostra

Tabela 156. Programas de amostra que demonstram o uso do MQI (C e COBOL) no AIX and Linux.

Uma tabela com quatro colunas. A primeira coluna lista as técnicas demonstradas pelas amostras. A segunda coluna lista as amostras C e a terceira coluna lista as amostras COBOL que demonstram cada uma das técnicas listadas na primeira coluna. A quarta coluna mostra se um executável C do servidor está ou não incluído e a quinta coluna mostra se um executável C do cliente está ou não incluído.

(continuação)

Técnica	**C (origem) ("1" na página 1075)**	**COBOL (origem) ("2" na página 1075)**	**Servidor (executável C)**	**Cliente (executável em C)**
Saída de balanceamento de carga de trabalho do cluster	amqswlm0	sem amostra	amqswlm	sem amostra
Colocando mensagens de maneira assíncrona e obtendo o status usando a chamada MQSTAT	amqsapt0	sem amostra	amqsapt	amqsaptc
Clientes reconectáveis	amqsphac amqsghac amqsmhac	sem amostra	não aplicável	amqsphac amqsghac amqsmhac
Utilizando os consumidores de mensagens para consumir mensagens de várias filas de maneira assíncrona	amqscbf0	sem amostra	amqscbf	amqscbfc
Especificando informações de conexão TLS no MQCONNX	amqssslc	sem amostra	não aplicável	amqssslc

Notas:

1. A versão executável das amostras do IBM MQ MQI client compartilha a mesma origem que as amostras que são executadas em um ambiente do servidor.
2. Programas de compilação que começam com 'amqm' com o compilador Micro Focus COBOL, aqueles que começam com 'amqi' com o compilador COBOL IBM e aqueles que começam com 'amq0' qualquer um deles.
3. AIX Suportado em IBM MQ for AIX apenas.
4. AIX No IBM MQ for AIX, este programa é chamado `amqsvfc0.c`
5. AIX O CICS é suportado apenas pelo IBM MQ for AIX.
6. Linux TUXEDO não é suportado por IBM MQ for Linux em System p.
7. A origem para o manipulador da fila de mensagens não entregues consiste em vários arquivos e é fornecida em um diretório separado.

Para obter informações detalhadas sobre o suporte para sistemas AIX and Linux, veja Requisitos do sistema para IBM MQ.

Windows *Amostras para IBM MQ for Windows*

As técnicas demonstradas pelos programas de amostra para o IBM MQ for Windows.

Tabela 157 na página 1076 lista quais arquivos de origem C e COBOL são fornecidos e se um executável do servidor ou cliente é incluído.

Tabela 157. Programas de amostra do IBM MQ for Windows demonstrando o uso do MQI (C e COBOL)

Técnica	**C (origem)**	**COBOL (origem)**	**Servidor (executável C)**	**Cliente (executável C)**
Usando a interface de publicação/assinatura	amqssbxa amqssuba amqspuba	sem amostra	amqssbx amqssub amqspub	sem amostra
Colocando mensagens usando a chamada MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Colocando uma única mensagem usando a chamada MQPUT1	amqsecha amqsinqa	amqminq2 amqmech2 amqiinq2 amqiech2	amqsech amqsinq	amqsechc amqsinqc
Colocando mensagens em uma lista de distribuição	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Respondendo a uma mensagem de solicitação	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Obtendo mensagens (nenhuma espera)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Obtendo mensagens (aguardar com um limite de tempo)	amqsget0	amq0get0	amqsget	amqsgetc
Obtendo mensagens (espera ilimitada)	amqstrg0	sem amostra	amqstrg	amqstrgc
Obtendo mensagens (com conversão de dados)	amqsecha	sem amostra	amqsech	amqsechc
Colocando mensagens de referência em uma fila	amqsprma	sem amostra	amqsprm	amqsprmc
Obtendo Mensagens de Referência de uma fila	amqsgrma	sem amostra	amqsgrm	amqsgrmc
Saída do canal de mensagem de referência	amqsxrma amqsqrma	sem amostra	amqsxrm	sem amostra
Procurando 20 primeiros caracteres de uma mensagem	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Procurando mensagens completas	amqsbcg0	sem amostra	amqsbcg	amqsbcgc
Usando uma fila de entrada compartilhada	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Usando uma fila de entrada exclusiva	amqstrg0	amq0req0	amqstrg	amqstrgc
Usando a chamada MQINQ	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Usando a chamada MQSET	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
Usando a chamada MQINQMP	amqsiqma	sem amostra	sem amostra	sem amostra
Usando uma fila de resposta	amqsreq0	amq0req0	amqsreq	amqsreqc
Solicitando exceções de mensagens	amqsreq0	amq0req0	amqsreq	amqsreqc
Aceitando uma mensagem truncada	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Usando um nome de fila resolvido	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc

Tabela 157. Programas de amostra do IBM MQ for Windows demonstrando o uso do MQI (C e COBOL) (continuação)

Técnica	**C (origem)**	**COBOL (origem)**	**Servidor (executável C)**	**Cliente (executável C)**
Acionando um processo	amqstrg0	sem amostra	amqstrg	amqstrgc
Usando a conversão de dados	amqsvfc0	sem amostra	sem amostra	sem amostra
IBM MQ (coordenando gerenciadores de banco de dados compatível com XA) acessando um banco de dados único usando SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	sem amostra	sem amostra
IBM MQ (coordenando gerenciadores de banco de dados compatíveis com XA) acessando dois bancos de dados usando SQL	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	sem amostra	sem amostra
TUXEDO transação para colocar mensagens	amqstxpx	sem amostra	sem amostra	sem amostra
TUXEDO para obter mensagens da transação	amqstxgx	sem amostra	sem amostra	sem amostra
o Servidor para TUXEDO	amqstxsx	sem amostra	sem amostra	sem amostra
Manipulador da fila de devoluções	Diretório `./tools/c/Samples/dlq` ("1" na página 1078)	sem amostra	amqsdlq	sem amostra
De um IBM MQ MQI client, colocando uma mensagem	sem amostra	sem amostra	sem amostra	amqsputc
De um IBM MQ MQI client, obtendo uma mensagem	sem amostra	sem amostra	sem amostra	amqsgetc
Conectando-se ao gerenciador de filas usando MQCONNX	amqscnxc	sem amostra	sem amostra	amqscnxc
Usando saídas de API	amqsaxe0	sem amostra	amqsaxe	sem amostra
Balanceamento de carga de trabalho do cluster	amqswlm0	sem amostra	amqswlm	sem amostra
rotinas de segurança SSPI	amqsspin	sem amostra	amqrspin.dll	amqrspin.dll
Colocando mensagens de maneira assíncrona e obtendo o status usando a chamada MQSTAT	amqsapt0	sem amostra	amqsapt	amqsaptc
Clientes reconectáveis	amqsphac amqsghac amqsmhac	sem amostra	Não-aplicável	amqsphac amqsghac amqsmhac

Tabela 157. Programas de amostra do IBM MQ for Windows demonstrando o uso do MQI (C e COBOL) (continuação)				
Técnica	**C (origem)**	**COBOL (origem)**	**Servidor (executável C)**	**Cliente (executável C)**
Utilizando os consumidores de mensagens para consumir mensagens de várias filas de maneira assíncrona	amqscbf0	sem amostra	amqscbf	amqscbfc
Especificando informações de conexão TLS no MQCONNX	amqssslc	sem amostra	não aplicável	amqssslc

Notas:

1. A origem para o manipulador da fila de mensagens não entregues consiste em vários arquivos e é fornecida em um diretório separado.

Windows *Amostras em Visual Basic para o IBM MQ for Windows*

As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas Windows.

O Tabela 158 na página 1078 mostra as técnicas demonstradas pelos programas de amostra do IBM MQ for Windows.

Um projeto pode conter vários arquivos. Quando você abre um projeto no Visual Basic, os outros arquivos serão carregados automaticamente. Nenhum programa executável é fornecido.

Todos os projetos de amostra, exceto mqtrivc.vbp, são configurados para funcionarem com o servidor IBM MQ. Para descobrir como mudar os projetos de amostra para funcionarem com os clientes IBM MQ, consulte “Preparando programas Visual Basic no Windows” na página 1032.

Tabela 158. Programas de amostra do IBM MQ for Windows que demonstram o uso da MQI (Visual Basic)	
Técnica	**Nome do arquivo do projeto**
Colocando mensagens usando a chamada MQPUT	amqsputb.vbp
Obtendo mensagens usando a chamada MQGET	amqsgetb.vbp
Procurando em uma fila usando a chamada MQGET	amqsbcgb.vbp
Amostras de MQGET e MQPUT simples (cliente)	mqtrivc.vbp
Amostras de MQGET e MQPUT simples (servidor)	mqtrivs.vbp
Colocando e obtendo sequências e estruturas definidas pelo usuário usando MQPUT e MQGET	strings.vbp
Usando estruturas PCF para iniciar e parar um canal	pcfsamp.vbp
Criando uma fila usando MQAI	amqsaicq.vbp
Listando as filas de um gerenciador de filas usando MQAI	amqsailq.vbp
Monitorando eventos usando MQAI	amqsaiem.vbp

IBM i *Amostras para IBM i*

As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas IBM i.

O Tabela 159 na página 1079 mostra as técnicas demonstradas pelos programas de amostra do IBM MQ for IBM i. Algumas técnicas ocorrem em mais de um programa de amostra, mas apenas um programa é listado na tabela.

Tabela 159. Programas de amostra demonstrando o uso do MQI (C e COBOL) no IBM i

Técnica	**C (origem) ("1" na página 1080)**	**COBOL (origem) ("2" na página 1080)**	**RPG (origem) ("3" na página 1080)**	**Cliente (executável C)(4)**
Colocando mensagens usando a chamada MQPUT	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
Colocando mensagens de um arquivo de dados usando a chamada MQPUT	AMQSPUT4	sem amostra	sem amostra	sem amostra
Colocando uma única mensagem usando a chamada MQPUT1	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
Colocando mensagens em uma lista de distribuição	AMQSPTL4	sem amostra	sem amostra	AMQSPTLC
Respondendo a uma mensagem de solicitação	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Obtendo mensagens (nenhuma espera)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Obtendo mensagens (aguardar com um limite de tempo)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
Obtendo mensagens (espera ilimitada)	AMQSTRG4	sem amostra	AMQ3TRG4	AMQSTRGC
Obtendo mensagens (com conversão de dados)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
Colocando mensagens de referência em uma fila	AMQSPRM4	sem amostra	sem amostra	AMQSPRMC
Obtendo Mensagens de Referência de uma fila	AMQSGRM4	sem amostra	sem amostra	AMQSGRMC
Saída do canal de mensagem de referência	AMQSQRM4, AMQSXRM4	sem amostra	sem amostra	sem Amostra
Saída de mensagen	AMQSCMX4	sem amostra	sem amostra	sem Amostra
Procurando os primeiros 49 caracteres de uma mensagem	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Procurando mensagens completas	AMQSBCG4	sem amostra	sem amostra	AMQSBCGC
Usando uma fila de entrada compartilhada	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Usando uma fila de entrada exclusiva	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Usando a chamada MQINQ	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Usando a chamada MQSET	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
Usando uma fila de resposta	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Solicitando exceções de mensagens	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Aceitando uma mensagem truncada	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Usando um nome de fila resolvido	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Acionando um processo	AMQSTRG4	sem amostra	AMQ3TRG4	AMQSTRGC
Servidor do acionador	AMQSERV4	sem amostra	AMQ3SRV4	sem amostra
Usando um servidor do acionador (incluindo transações do CICS)	AMQSERV4	sem amostra	AMQ3SRV4	sem amostra

Tabela 159. Programas de amostra demonstrando o uso do MQI (C e COBOL) no IBM i (continuação)				
Técnica	**C (origem) ("1" na página 1080)**	**COBOL (origem) ("2" na página 1080)**	**RPG (origem) ("3" na página 1080)**	**Cliente (executável C)(4)**
Usando a conversão de dados	AMQSVFC4	sem amostra	sem amostra	sem amostra
Usando saídas de API	AMQSAXE0	sem amostra	sem amostra	sem amostra
Balanceamento de carga de trabalho do cluster	AMQSWLM0	sem amostra	sem amostra	sem amostra
Colocando mensagens de maneira assíncrona e obtendo o status usando a chamada MQSTAT	AMQSAPT0	sem amostra	sem amostra	AMQSAPTC
Usando a interface de publicação/assinatura	AMQSPUBA, AMQSSUBA, AMQSSBXA	sem amostra	sem amostra	AMQSPUBC, AMQSSUBC, AMQSSBXC
Clientes reconectáveis (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	sem amostra	sem amostra	sem amostra
Usando os consumidores de mensagens para consumir assincronamente mensagens de várias filas (5)	AMQSCBFO	sem amostra	sem amostra	sem amostra
Especificando informações de conexão TLS no MQCONNX	AMQSSSLC	sem amostra	sem amostra	AMQSSSLC
Conectando-se ao gerenciador de filas usando MQCONNX	AMQSCNXC	sem amostra	sem amostra	AMQSCNXC
Consultar propriedades de uma manipulação de mensagens, usando MQINQMP, a partir de uma fila de mensagens	AMQISQMA	sem amostra	sem amostra	AMQISQMC
Configure propriedades de uma manipulação de mensagens usando MQSETMP e coloque-a em uma fila de mensagens	AMQSSQMA	sem amostra	sem amostra	AMQSSQMC

Notas:

1. A origem para as amostras C está no arquivo QMQMSAMP/QCSRC. Os arquivos de inclusão existem como membros no arquivo QMQM/H.
2. A origem para as amostras COBOL estão nos arquivos QMQMSAMP/QCBLLESRC. Os membros são denominados AMQ0 xxx 4, em que xxx indica a função de amostra.
3. A origem para as amostras RPG estão em QMQMSAMP/QRPGLESRC. Os membros são denominados AMQ3 xxx 4, em que xxx indica a função de amostra. Os membros de cópia existem em QMQM/QRPGLESRC. O nome de cada membro tem o sufixo G.
4. A versão executável das amostras do IBM MQ MQI client compartilha a mesma origem que as amostras que são executadas em um ambiente do servidor. A origem para as amostras no ambiente do cliente é a mesma que a do servidor. As amostras do IBM MQ MQI client são vinculadas com a biblioteca do cliente LIBMQIC e as amostras do servidor IBM MQ são vinculadas com a biblioteca do servidor LIBMQM.
5. Se o executável do cliente para o aplicativo de amostra do cliente Reconectável e para o aplicativo do consumidor de forma assíncrona tiver que ser executado, ele deverá ser compilado e vinculado à

biblioteca encadeada LIBMQIC_R. Assim, ele deve ser executado em ambiente encadeado. Configure a variável de ambiente QIBM_MULTI_THREADED como 'Y' e execute o aplicativo de qsh.

Consulte Configurando IBM MQ com Java e JMS para obter mais informações..

Consulte “Preparando e executando programas de amostra em IBM i” na página 1083 para obter mais informações.

Além disso, a opção de amostra IBM MQ for IBM i inclui um arquivo de dados de amostra, que você usa como entrada para os programas de amostra, AMQSDATA e os programas CL de amostra que demonstram tarefas de administração. As amostras de CL são descritas em Administrando o IBM i . O programa CL de amostra `amqsamp4` poderia ser usado para criar filas a serem usadas com os programas de amostra descritos neste tópico.

Multi ***Preparando e executando os programas de amostra***

Depois de concluir alguma preparação inicial, será possível então executar os programas de amostra.

Sobre esta tarefa

Antes de executar os programas de amostra, deve-se primeiro criar um gerenciador de filas e também criar as filas que você precisa. Também pode ser necessário fazer alguma preparação adicional, por exemplo, se você desejar executar amostras COBOL. Depois de concluir a preparação necessária, será possível então executar os programas de amostra.

Procedimento

Para obter informações sobre como preparar e executar os programas de amostra, veja os tópicos a seguir:

- “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081
- “Preparando e executando programas de amostra em IBM i” na página 1083
- “Preparando e executando programas de amostra em AIX and Linux” na página 1085
- “Preparando e executando programas de amostra em Windows” na página 1087

Multi *Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms*

Para poder executar os aplicativos de amostra, deve-se primeiro criar um gerenciador de filas. É possível então configurar o gerenciador de filas para aceitar com segurança as solicitações de conexão recebidas de aplicativos em execução no modo cliente.

Antes de começar

Assegure que o gerenciador de filas já exista e tenha sido iniciado. Determine se os registros de autenticação de canal já estão ativados emitindo o comando do MQSC:

```
DISPLAY QMGR CHLAUTH
```

Importante: Esta tarefa espera que os registros de autenticação de canal estejam ativados. Se esse for um gerenciador de filas usado por outros usuários e aplicativos, a mudança dessa configuração afetará todos os outros usuários e aplicativos. Se o seu gerenciador de filas não fizer uso de registros de autenticação de canal, a etapa 4 poderá ser substituída por um método de autenticação alternativo (por exemplo, uma saída de segurança) que configura o MCAUSER para o *non-privileged-user-id* que você obterá na etapa “1” na página 1082.

Deve-se saber qual nome de canal seu aplicativo espera usar para que o aplicativo possa ter permissão para usar o canal. Deve-se também saber quais objetos, por exemplo, filas ou tópicos, seu aplicativo espera usar para que seu aplicativo possa ter permissão para usá-los.

Sobre esta tarefa

Esta tarefa cria um ID do usuário não privilegiado para ser usado para um aplicativo cliente que se conecta ao gerenciador de filas. O acesso é concedido ao aplicativo cliente somente para ser capaz de usar o canal que necessita e a fila que precisa por uso desse ID do usuário.

Procedimento

1. Obtenha um ID do usuário no sistema no qual seu gerenciador de filas está em execução. Para essa tarefa, esse ID do usuário não deve ser um usuário administrativo privilegiado. Esse ID do usuário será a autoridade sob a qual a conexão do cliente será executada no gerenciador de filas.
2. Inicie um programa de listener com os comandos a seguir, em que:

   qmgr-name é o nome do seu gerenciador de filas
   nnnn é o número da porta escolhido

   a) 
   Para sistemas AIX, Linux, and Windows:

   ```
   runmqlsr -t tcp -m qmgr-name -p nnnn
   ```

   b) 
   Para IBM i:

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. Se seu aplicativo usar SYSTEM.DEF.SVRCONN, então, esse canal já está definido. Se o seu aplicativo utiliza outro canal, crie-o ao emitir o comando MQSC a seguir:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   onde *channel-name* é o nome do seu canal.
4. Crie uma regra de autenticação de canal permitindo que apenas o endereço IP do seu sistema cliente use o canal ao emitir o comando MQSC a seguir:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   em que

   channel-name é o nome de seu canal.
   client-machine-IP-address é o endereço IP de seu sistema cliente. Se o aplicativo cliente de amostra estiver em execução na mesma máquina que o gerenciador de filas, então, use o endereço IP '127.0.0.1' se seu aplicativo for se conectar usando 'localhost'. Se várias máquinas clientes diferentes forem se conectar, é possível usar um padrão ou um intervalo em vez de um único endereço IP. Consulte Endereços IP genéricos para obter detalhes.
   non-privileged-user-id é o ID do usuário obtido na etapa “1” na página 1082
5. Se seu aplicativo usar SYSTEM.DEFAULT.LOCAL.QUEUE, então, essa fila já está definida. Se o seu aplicativo usa outra fila, crie-a emitindo o seguinte comando MQSC:

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   em que *queue-name* é o nome de sua fila.
6. Conceda acesso para conectar-se a e consultar o gerenciador de filas ao emitir o comando MQSC a seguir:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

onde *non-privileged-user-id* é a identificação de usuário obtida na etapa "1" na página 1082.

7. Se o seu aplicativo for um aplicativo ponto a ponto, ou seja, ele usa filas, conceda acesso para permitir a consulta, a colocação e a obtenção de mensagens usando sua fila pela identificação de usuário a ser usada ao emitir os comandos MQSC a seguir:

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

em que

queue-name é o nome da sua fila

non-privileged-user-id é o ID do usuário obtido na etapa "1" na página 1082

8. Se seu aplicativo for de publicar/assinar, ou seja, usa tópicos, conceda acesso para permitir publicação e assinatura usando seu tópico pelo ID do usuário a ser usado, emitindo os comandos do MQSC:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

em que

non-privileged-user-id é o ID do usuário obtido na etapa "1" na página 1082

Isso fornecerá acesso *non-privileged-user-id* a qualquer tópico na árvore de tópicos; como alternativa, é possível definir um objeto do tópico usando **DEFINE TOPIC** e conceder acesso somente à parte da árvore de tópicos referida por esse objeto do tópico. Consulte Controlando o acesso do usuário aos tópicos para obter detalhes.

Como proceder a seguir

Agora, seu aplicativo cliente pode se conectar ao gerenciador de filas e colocar ou obter mensagens usando a fila.

Conceitos relacionados

ALW Concedendo acesso a um objeto do IBM MQ no AIX, Linux, and Windows

Referências relacionadas

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

IBM i Autoridades do IBM MQ no IBM i

IBM i *Preparando e executando programas de amostra em IBM i*

Antes de executar os programas de amostra no IBM i, deve-se primeiro criar um gerenciador de filas e também criar as filas necessárias. Se você deseja executar amostras COBOL, pode ser necessário fazer alguma preparação adicional.

Sobre esta tarefa

A origem para os programas de amostra do IBM MQ for IBM i é fornecida na biblioteca QMQMSAMP como membros de QCSRC, QCLSRC, QCBLLESRC e QRPGLESRC.

É possível usar suas próprias filas ao executar as amostras ou executar o programa de amostra AMQSAMP4 para criar algumas filas de amostra. A origem para o programa AMQSAMP4 é incluída no arquivo QCLSRC na biblioteca QMQMSAMP. É possível compilá-la usando o comando CRTCLPGM.

Para executar as amostras, use as versões executáveis C que são fornecidas na biblioteca QMQM ou compile-as de uma maneira semelhante a qualquer outro aplicativo IBM MQ.

V 9.4.0 V 9.4.0 Os programas de amostra a seguir têm recursos de autenticação:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsssl c.c`
- `amqssuba.c`

As versões executáveis dessas amostras têm a autenticação ativada No entanto, compilar as versões de origem com autenticação ativada requer que a sinalização de compilação **SAMPLE_AUTH_ENABLED** seja definida e o arquivo de origem `amqsauth.c` seja compilado com a amostra desejada. Por exemplo:

- Criando o programa `amqsssl c` sem autenticação ativada:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC) MODULE(MYLIB/AMQSSSLC) BNDSRVPGM(QMQM/LIBMQIC)
```

- Criando o `amqsssl c` com a autenticação ativada:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) DEFINE('SAMPLE_AUTH_ENABLED') SRCFILE(QMQMSAMP/QCSRC)
CRTCMOD MODULE(MYLIB/AMQSAUTH) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC_AUTH) MODULE(MYLIB/AMQSSSLC MYLIB/AMQSAUTH) BNDSRVPGM(QMQM/LIBMQIC)
```

Procedimento

1. Crie um gerenciador de filas e configure as definições padrão.

   Deve-se fazer isso antes que seja possível executar algum dos programas de amostra. Para obter mais informações sobre criar um gerenciador de filas, consulte Administrando IBM MQ. Para obter informações sobre como configurar um gerenciador de filas para aceitar com segurança as solicitações de conexão recebidas dos aplicativos que estão em execução no modo cliente, consulte “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081.
2. Para chamar um dos programas de mostra usando dados do membro PUT no arquivo AMQSDATA da biblioteca QMQMSAMP, use um comado como:

   ```
   CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
   ```

   Nota: Para um módulo compilado usar o sistema de arquivos IFS, especifique a opção SYSIFCOPT(*IFSIO) em CRTCMOD, em seguida, o nome do arquivo, passado como um parâmetro, deve ser especificado no formato a seguir:

   ```
   home/me/myfile
   ```
3. Se você deseja usar as versões COBOL dos exemplos de Inquire, Set e Echo, mude as definições de processo antes de executar essas amostras.

   Para os exemplos de Inquire, Set e Echo, as definições de amostra acionam as versões de C dessas amostras. Se você desejar as versões COBOL, deverá mudar as definições de processo:

   - SYSTEM.SAMPLE.INQPROCESS

- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

No IBM i, é possível usar o comando **CHGMQMPRC** (para obter detalhes, consulte Mudar processo do MQ (CHGMQMPRC)) ou editar e executar o comando **AMQSAMP4** com a definição alternativa.

4. Execute os programas de amostra.

   Para obter mais informações sobre os parâmetros que cada uma das amostras espera, consulte as descrições das amostras individuais.

   Nota: Para os programas de amostra em COBOL, ao passar nomes de filas como parâmetros, deve-se fornecer 48 caracteres, preenchendo com caracteres em branco, se necessário. Qualquer coisa diferente de 48 caracteres faz com que o programa falhe com o código de razão 2085.

Referências relacionadas

"Amostras para IBM i" na página 1078
As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas IBM i.

Linux AIX *Preparando e executando programas de amostra em AIX and Linux*

Antes de executar os programas de amostra no AIX and Linux, deve-se primeiro criar um gerenciador de filas e também criar as filas necessárias. Se você deseja executar amostras COBOL, pode ser necessário fazer alguma preparação adicional.

Sobre esta tarefa

O IBM MQ em arquivos de sistema de amostra do AIX and Linux estão nos diretórios listados em Tabela 160 na página 1085 se os padrões tiverem sido usados no momento da instalação.

Tabela 160. Onde localizar as amostras para IBM MQ em sistemas AIX and Linux

Conteúdo	**Diretório**
arquivos de origem	*MQ_INSTALLATION_PATH*/samp
arquivos de origem do manipulador da fila de mensagens não entregues	*MQ_INSTALLATION_PATH*/samp/dlq
arquivos executáveis	*MQ_INSTALLATION_PATH*/samp/bin

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

As amostras precisam de um conjunto de filas com o qual trabalhar. É possível usar suas próprias filas ou executar o arquivo de amostra do MQSC amqscos0.tst para criar um conjunto. Para executar as amostras, use as versões executáveis fornecidas ou compile as versões de origem como você faria com qualquer outro aplicativo usando um compilador ANSI.

V 9.4.0 V 9.4.0 Os programas de amostra a seguir têm recursos de autenticação:

- amqsbcg0.c
- amqsfhac.c
- amqsget0.c
- amqsghac.c
- amqsmhac.c
- amqsphac.c
- amqspuba.c
- amqsput0.c
- amqsssl c.c
- amqssuba.c

As versões executáveis dessas amostras têm a autenticação ativada No entanto, compilar as versões de origem com autenticação ativada requer que a sinalização de compilação **SAMPLE_AUTH_ENABLED** seja definida e o arquivo de origem `amqsauth.c` seja compilado com a amostra desejada. Por exemplo:

- Compilando `amqsput0.c` sem autenticação ativada:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -o /bin/amqsput
amqsput0.c
```

- Compilando `amqsput0.c` com a autenticação ativada:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -D SAMPLE_AUTH_ENABLED
-o /bin/amqsputc_auth amqsauth.c amqsput0.c
```

Procedimento

1. Crie um gerenciador de filas e configure as definições padrão.

   Deve-se fazer isso antes que seja possível executar algum dos programas de amostra. Para obter mais informações sobre criar um gerenciador de filas, consulte Administrando IBM MQ. Para obter informações sobre como configurar um gerenciador de filas para aceitar com segurança as solicitações de conexão recebidas dos aplicativos que estão em execução no modo cliente, consulte “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081.
2. Se você não estiver usando suas próprias filas, execute o arquivo de amostra do MQSC amqscos0.tst para criar um conjunto de filas.

   Para fazer isso em sistemas AIX and Linux, insira:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   Verifique o arquivo `sampobj.out` para certificar-se de que não haja erros.
3. Se você deseja usar as versões COBOL dos exemplos de Inquire, Set e Echo, mude as definições de processo antes de executar essas amostras.

   Para os exemplos de Inquire, Set e Echo, as definições de amostra acionam as versões de C dessas amostras. Se você desejar as versões COBOL, deverá mudar as definições de processo:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   No AIX and Linux, faça isso editando o arquivo `amqscos0.tst` e mudando os nomes dos arquivos executáveis C para os nomes dos arquivos executáveis COBOL antes de usar o comando **runmqsc** para executar essas amostras.
4. Execute os programas de amostra.

   Para executar uma amostra, insira seu nome seguido por quaisquer parâmetros, por exemplo:

   ```
   amqsput myqueue qmanagername
   ```

   em que *myqueue* é o nome da fila na qual as mensagens serão colocadas e *qmanagername* é o gerenciador de filas que possui *myqueue*.

   Para obter mais informações sobre os parâmetros que cada uma das amostras espera, consulte as descrições das amostras individuais.

   Nota: Para os programas de amostra em COBOL, ao passar nomes de filas como parâmetros, deve-se fornecer 48 caracteres, preenchendo com caracteres em branco, se necessário. Qualquer coisa diferente de 48 caracteres faz com que o programa falhe com o código de razão 2085.

Referências relacionadas

“Amostras para sistemas AIX and Linux” na página 1072
As técnicas demonstradas pelos programas de amostra para o IBM MQ for AIX or Linux.

Windows *Preparando e executando programas de amostra em Windows*

Antes de executar os programas de amostra no Windows, deve-se primeiro criar um gerenciador de filas e também criar as filas necessárias. Se você deseja executar amostras COBOL, pode ser necessário fazer alguma preparação adicional.

Sobre esta tarefa

Os arquivos de amostra do IBM MQ for Windows estão nos diretórios listados em Tabela 161 na página 1087, se os padrões foram usados no momento da instalação. A unidade de instalação é padronizada como <c:>.

Tabela 161. Onde localizar amostras para IBM MQ for Windows

Conteúdo	**Diretório**
Código de origem C	*MQ_INSTALLATION_PATH*\Tools\C\Samples
Código fonte para a amostra do manipulador de mensagens não entregues	\tools\c\samples\dlq do*MQ_INSTALLATION_PATH*
Código fonte COBOL	*MQ_INSTALLATION_PATH*\Tools\Cobol \ Amostras
Arquivos C executáveis [1]	*MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin (versões 32 bits) do *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (versões 64 bits)
Arquivos de amostra do MQSC	*MQ_INSTALLATION_PATH*\Tools\MQSC\Samples
Código fonte Visual Basic	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
Amostras do .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Amostras

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Nota: Versões de 64 bits estão disponíveis de algumas amostras de arquivos C executáveis.

As amostras precisam de um conjunto de filas com o qual trabalhar. É possível usar suas próprias filas ou executar o arquivo de amostra do MQSC amqscos0.tst para criar um conjunto de filas. Para executar as amostras, use as versões executáveis fornecidas ou compile as versões de origem como você faria com qualquer outro aplicativo IBM MQ for Windows.

V 9.4.0 V 9.4.0 Os programas de amostra a seguir têm recursos de autenticação:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsssl c.c`
- `amqssuba.c`

As versões executáveis dessas amostras têm a autenticação ativada No entanto, compilar as versões de origem com autenticação ativada requer que a sinalização de compilação **SAMPLE_AUTH_ENABLED** seja definida e o arquivo de origem `amqsauth.c` seja compilado com a amostra desejada. Por exemplo:

- Compilando `amqsput0.c` sem autenticação ativada:

```
CL amqsput0.c /link mqic.lib /OUT:Bin\amqsputc.exe
```

- Compilando `amqsput0.c` com a autenticação ativada:

```
CL /D SAMPLE_AUTH_ENABLED amqsauth.c amqsput0.c /link mqic.lib /OUT:Bin\amqsputc_auth.exe
```

Procedimento

1. Crie um gerenciador de filas e configure as definições padrão.

   Deve-se fazer isso antes que seja possível executar algum dos programas de amostra. Para obter mais informações sobre criar um gerenciador de filas, consulte Administrando IBM MQ. Para obter informações sobre como configurar um gerenciador de filas para aceitar com segurança as solicitações de conexão recebidas dos aplicativos que estão em execução no modo cliente, consulte "Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms" na página 1081.
2. Se você não estiver usando suas próprias filas, execute o arquivo de amostra do MQSC amqscos0.tst para criar um conjunto de filas.

   Para fazer isso em sistemas Windows, insira:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   Verifique o arquivo `sampobj.out` para certificar-se de que não haja erros. Esse arquivo encontra-se em seu diretório atual.
3. Se você deseja usar as versões COBOL dos exemplos de Inquire, Set e Echo, mude as definições de processo antes de executar essas amostras.

   Para os exemplos de Inquire, Set e Echo, as definições de amostra acionam as versões de C dessas amostras. Se você desejar as versões COBOL, deverá mudar as definições de processo:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   No Windows, faça isso editando o arquivo `amqscos0.tst` e mudando os nomes dos arquivos executáveis C para os nomes dos arquivos executáveis COBOL antes de usar o comando **runmqsc** para executar essas amostras.
4. Execute os programas de amostra.

   Para executar uma amostra, insira seu nome seguido por quaisquer parâmetros, por exemplo:

   ```
   amqsput myqueue qmanagername
   ```

   em que *myqueue* é o nome da fila na qual as mensagens serão colocadas e *qmanagername* é o gerenciador de filas que possui *myqueue*.

   Para obter mais informações sobre os parâmetros que cada uma das amostras espera, consulte as descrições das amostras individuais.

   Nota: Para os programas de amostra em COBOL, ao passar nomes de filas como parâmetros, deve-se fornecer 48 caracteres, preenchendo com caracteres em branco, se necessário. Qualquer coisa diferente de 48 caracteres faz com que o programa falhe com o código de razão 2085.

Referências relacionadas

"Amostras para IBM MQ for Windows" na página 1075
As técnicas demonstradas pelos programas de amostra para o IBM MQ for Windows.

"Amostras em Visual Basic para o IBM MQ for Windows" na página 1078
As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas Windows.

O programa de amostra de saída de API

A saída API de amostra gera um rastreio MQI para um arquivo especificado pelo usuário com um prefixo definido na variável de ambiente **MQAPI_TRACE_LOGFILE** .

Para obter informações adicionais sobre as saídas de API, consulte “Gravando e compilando saídas de API em Multiplataformas” na página 965.

Origem

amqsaxe0.c

Binário

amqsaxe

Configurando para a saída de amostra

1. Inclua as seguintes informações no ApiExitSub-rotina local do arquivo `qm.ini` .

   Plataformas diferentes de Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.

2. Configure a variável de ambiente **MQAPI_TRACE_LOGFILE**

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. Execute seu aplicativo.

   Os arquivos de saída são criados no diretório `/tmp` com nomes como: `MqiTrace.pid.tid.log`.

O programa de amostra Asynchronous Consumption

O programa de amostra amqscbf demonstra o uso de MQCB e MQCTL para consumir mensagens de diversas filas de forma assíncrona.

amqscbf é fornecido como código-fonte C e um cliente binário e executável de servidor em plataformas AIX, Linux, and Windows.

O programa é iniciado a partir da linha de comandos e aceita os seguintes parâmetros opcionais:

```
Usage: [Options] Queue Name {queue_name}
  where Options are:
    -m Queue Manager Name
    -o Open options
    -r Reconnect Type
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

Forneça mais de um nome de fila para ler mensagens de diversas filas (no máximo dez filas são suportadas pela amostra).

Nota: `Reconnect type` é válido somente para programas clientes.

exemplo

O exemplo mostra amqscbf executado como um programa do servidor lendo uma mensagem de `QL1` e, em seguida, sendo interrompido.

Use o IBM MQ Explorer para colocar uma mensagem de teste em `QL1`. Pare o programa pressionando Enter.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

O que amqscbf demonstra

A amostra mostra como ler mensagens de diversas filas na ordem de chegada. Isso exigiria muito mais código usando MQGET síncrono. No caso de consumo assíncrono, nenhuma pesquisa é necessário e gerenciamento de encadeamento e armazenamento é executado pelo IBM MQ. Um exemplo do "mundo real" precisaria lidar com erros; na amostra, os erros são gravados no console.

O código de amostra tem as etapas a seguir,

1. Definir a função de retorno de chamada de consumo de mensagem única,

   ```
   void MessageConsumer(MQHCONN      hConn,
                        MQMD    * pMsgDesc,
                        MQGMO   * pGetMsgOpts,
                        MQBYTE  * Buffer,
                        MQCBC   * pContext)
   { ... }
   ```

2. Conectar-se ao gerenciador de filas,

   ```
   MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
   ```

3. Abrir as filas de entrada e associar cada uma à função de retorno de chamada `MessageConsumer`,

   ```
   MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
   cbd.CallbackFunction = MessageConsumer;
   MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
   ```

   `cbd.CallbackFunction` não precisa ser configurado para cada fila; é um campo somente de entrada. Mas você poderia associar uma função de retorno de chamada diferente a cada fila.

4. Iniciar o consumo das mensagens,

   ```
   MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
   ```

5. Esperar até que o usuário tenha pressionado Enter e, em seguida, parar o consumo de mensagens,

   ```
   MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
   ```

6. Por fim, desconectar-se do gerenciador de filas,

   ```
   MQDISC(&Hcon,&CompCode,&Reason);
   ```

O programa de amostra Asynchronous Put

Aprenda sobre execução da amostra amqsapt e o design do programa de amostra Asynchronous Put.

O programa de amostra Asynchronous Put coloca as mensagens em uma fila, usando a chamada MQPUT assíncrona e, em seguida, recupera as informações de status usando a chamada MQSTAT. Consulte "Recursos demonstrados nos programas de amostra em multiplataformas" na página 1072 para obter o nome deste programa em diferentes plataformas.

Executando a Amostra amqsapt

Este programa usa até 6 parâmetros:

1. O nome da fila de destino (obrigatório)
2. O nome do gerenciador de filas (opcional)
3. Opções de abertura (opcional)
4. Opções de fechamento (opcional)
5. O nome do gerenciador de filas de destino (opcional)
6. O nome da fila dinâmica (opcional)

Se um gerenciador de filas não for especificado, amqsapt se conecta ao gerenciador de filas padrão.

Design do programa de amostra Asynchronous Put

O programa usa a chamada MQOPEN com as opções de saída fornecidas ou com as opções MQOO_OUTPUT e MQOO_FAIL_IF_QUIESCING para abrir a fila de destino para colocar mensagens.

Se ele não puder abrir a fila, o programa envia uma mensagem de erro contendo o código de razão retornado pela chamada MQOPEN. Para manter o programa simples, nesta e em chamadas MQI subsequentes, o programa usa valores padrão para muitas das opções.

Para cada linha de entrada, o programa lê o texto em um buffer e usa a chamada MQPUT com MQPMO_ASYNC_RESPONSE para criar uma mensagem de datagrama que contém o texto dessa linha e o coloca de forma assíncrona na fila de destino. O programa continua até chegar ao fim da entrada ou a chamada MQPUT falha. Se o programa atingir o final da entrada, ele fechará a fila usando a chamada MQCLOSE.

O programa, em seguida, emite a chamada MQSTAT, retornando uma estrutura MQSTS, e exibe mensagens que contêm o número de mensagens colocadas com êxito, o número de mensagens colocadas com um aviso e o número de falhas.

Os programas de amostra Browse

As mensagens de Procurar por programas de amostra em uma fila usando a chamada MQGET.

Consulte "Recursos demonstrados nos programas de amostra em multiplataformas" na página 1072 para obter os nomes desses programas.

Design do programa de amostra Browse

O programa abre a fila de destino usando a chamada MQOPEN com a opção MQOO_BROWSE. Se ele não puder abrir a fila, o programa envia uma mensagem de erro contendo o código de razão retornado pela chamada MQOPEN.

Para cada mensagem na fila, o programa usará a chamada MQGET para copiar a mensagem da fila, em seguida, exibirá os dados contidos na mensagem. A chamada MQGET usa estas opções:

MQGMO_BROWSE_NEXT
Depois da chamada MQOPEN, o cursor de pesquisa é posicionado logicamente antes da primeira mensagem na fila, portanto, essa opção faz com que a ***primeira*** mensagem seja retornada quando a chamada é feita pela primeira vez.

MQGMO_NO_WAIT
O programa não espera se não houver mensagens na fila.

MQGMO_ACCEPT_TRUNCATED_MSG
A chamada MQGET especifica um buffer de tamanho fixo. Se uma mensagem for maior do que esse buffer, o programa exibe a mensagem truncada, juntamente com um aviso de que a mensagem foi truncada.

O programa demonstra como os campos *MsgId* e *CorrelId* devem ser desmarcados da estrutura MQMD após cada chamada MQGET, porque a chamada define esses campos para os valores contidos na mensagem que ela recupera. Desmarcar esses campos significa que sucessivas chamadas MQGET recuperam as mensagens na ordem em que elas são retidas na fila.

O programa continua até o final da fila; a chamada MQGET retorna o código de razão MQRC_NO_MSG_AVAILABLE e o programa exibe uma mensagem de aviso. Se a chamada MQGET falhar, o programa exibirá uma mensagem de erro que contém o código de razão.

O programa, então, fecha a fila usando a chamada MQCLOSE.

Os programas de amostra Procurar para AIX, Linux, and Windows
Considere usar este tópico quando estiver aprendendo sobre programas de amostra Procurar no AIX, Linux, and Windows.

A versão de C do programa aceita dois parâmetros

1. O nome da fila de origem (necessário)
2. O nome do gerenciador de filas (opcional)

Se um gerenciador de filas não for especificado, ele se conectará ao padrão. Por exemplo, insira um dos seguintes:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

em que `myqueue` é o nome da fila a partir da qual as mensagens serão visualizadas e `qmanagername` é o gerenciador de filas que possui `myqueue`.

Se você omitir o `qmanagername`, quando estiver executando a amostra em C, supõe-se que o gerenciador de filas padrão possui a fila.

A versão em COBOL não tem nenhum parâmetro. Ela se conecta ao gerenciador de filas padrão e quando executá-la, será solicitado o seguinte:

```
Please enter the name of the target queue
```

Somente os primeiros 50 caracteres de cada mensagem serão exibidos, seguidos por `- - - truncated` quando este for o caso.

Os programas de amostra Procurar no IBM i
Cada programa recupera cópias de todas as mensagens na fila que você especifica ao chamar o programa; as mensagens permanecem na fila.

É possível usar a fila fornecida SYSTEM.SAMPLE.LOCAL; execute o programa de amostra Put primeiro para colocar algumas mensagens na fila. É possível usar a fila SYSTEM.SAMPLE.ALIAS, que é um nome de alias para a mesma fila local. O programa continua até atingir o final da fila ou uma chamada MQI falhar.

As amostras C permitem que você especifique o nome do gerenciador de filas, geralmente como o segundo parâmetro, de forma semelhante às amostras dos sistemas Windows. Por exemplo:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Se um gerenciador de filas não for especificado, ele se conectará ao padrão. Isso também é relevante para as amostras RPG. No entanto, com as amostras RPG, deve-se fornecer um nome de gerenciador de filas em vez de permitir o uso do padrão.

ALW *O programa de amostra Browser*

O programa de amostra Browser lê e grava o descritor de mensagens e os campos de conteúdo de mensagens de todas as mensagens em uma fila.

O programa de amostra é gravado como um utilitário, não apenas para demonstrar uma técnica. Consulte "Recursos demonstrados nos programas de amostra em multiplataformas" na página 1072 para obter os nomes desses programas.

Este programa usa esses parâmetros posicionais:

1. O nome da fila de origem (obrigatório)
2. O nome do gerenciador de filas (obrigatório)
3. Um parâmetro opcional para propriedades (opcional)

Use as variáveis de ambiente a seguir para fornecer credenciais que são usadas para autenticar com o gerenciador de filas:

MQSAMP_USER_ID
Configure como o ID do usuário a ser usado para autenticação de conexão, se você desejar usar um ID do usuário e uma senha para autenticar com o gerenciador de filas. O programa solicita a senha para acompanhar o ID do usuário.

Linux V 9.4.0 AIX **MQSAMP_TOKEN**
Configure como um valor não em branco se desejar fornecer um token de autenticação para autenticar com o gerenciador de filas. O programa solicita o token de autenticação. Os tokens de autenticação podem ser usados somente pela amostra **amqsbcgc** que usa ligações do cliente

Para executar esses programas, insira um dos seguintes comandos:

- `amqsbcg myqueue qmanagername`
- `amqsbcgc myqueue qmanagername`

em que *myqueue* é o nome da fila na qual as mensagens serão procuradas e *qmanagername* é o gerenciador de filas que possui *myqueue*.

Ele lê cada mensagem da fila e grava o seguinte para stdout:

- Campos do descritor de mensagens formatados
- Dados da mensagem (com dump em hex e, quando possível, formato de caractere)

Tabela 162. Valores permitidos para o parâmetro da propriedade

Value	Comportamento
0	Comportamento padrão.. As propriedades que são entregues ao aplicativo dependem do atributo de fila **PropertyControl** do qual a mensagem é recuperada.

Tabela 162. Valores permitidos para o parâmetro da propriedade (continuação)	
Value	**Comportamento**

1	Um identificador de mensagens é criado e usado com o MQGET. As propriedades da mensagem, exceto aquelas contidas no descritor de mensagens (ou extensão), são exibidas de forma semelhante para o descritor de mensagens. Por exemplo:
	`****Message properties****`
	`   property name: property value`
	Ou se nenhuma propriedade estiver disponível:
	`****Message properties****`
	`   None`
	Valores numéricos são exibidos usando o `printf`, os valores de sequência são colocados entre aspas simples e as sequências de bytes são marcadas com X e colocadas entre aspas simples, como para o descritor de mensagens.

2	MQGMO_NO_PROPERTIES é especificado, de forma que apenas as propriedades do descritor de mensagens sejam retornadas.
3	MQGMO_PROPERTIES_FORCE_MQRFH2 é especificado, de forma que todas as propriedades sejam retornadas nos dados da mensagem.
4	MQGMO_PROPERTIES_COMPATIBILITY é especificado para que todas as propriedades possam ser retornadas, dependendo de uma propriedade IBM MQ estar incluída, caso contrário, as propriedades serão descartadas.

O programa é restrito à impressão dos primeiros 65535 caracteres da mensagem e falha com a razão `truncated msg` se uma mensagem mais longa for lida.

Para obter um exemplo da saída desse utilitário, consulte Procurando filas.

A transação de amostra do CICS

Uma transação do programa de amostra do CICS é fornecida denominada amqscic0.ccs para código-fonte e amqscic0 para a versão executável. É possível construir transações usando os recursos padrão do CICS.

Consulte “Construindo um aplicativo processual” na página 1012 para obter detalhes sobre os comandos necessários para sua plataforma.

A transação lê mensagens da fila de transmissão SYSTEM.SAMPLE.CICS.WORKQUEUE no gerenciador de filas padrão e coloca-os na fila local, cujo nome está contido no cabeçalho de transmissão da mensagem Quaisquer falhas são enviadas para a fila SYSTEM.SAMPLE.CICS.DLQ.

Nota: É possível usar um script de amostra MQSC amqscic0.tst para criar estas filas e filas de entrada de amostra.

O programa de amostra Connect

O programa de amostra Connect permite explorar a chamada MQCONNX e suas opções a partir de um cliente. A amostra se conecta ao gerenciador de filas usando a chamada MQCONNX, consulta sobre o nome do gerenciador de filas usando a chamada MQINQ e exibe o mesmo. Além disso, aprenda sobre como executar a amostra amqscnxc.

Nota: O programa de amostra Connect é uma amostra do cliente. É possível compilar e executar a mesma em um servidor, mas a função é significativa somente em um cliente e somente arquivos executáveis de cliente são fornecidos.

Executando a amostra amqscnxc

A sintaxe da linha de comandos do programa de amostra Connect é:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

Os parâmetros são opcionais e sua ordem não é importante, exceto para QMgrName, que, se especificado, deve vir por último. Os parâmetros são:

ConnName
O nome da conexão TCP/IP do gerenciador de filas do servidor

Se você não especificar o nome da conexão TCP/IP, MQCONNX será emitido com *ClientConnPtr* configurado para NULL.

SvrconnChannelName
O nome do canal de conexão do servidor

Se você especificar o nome da conexão TCP/IP, mas não o canal de conexão do servidor (o inverso não é permitido), a amostra usará o nome SYSTEM.DEF.SVRCONN.

User
O nome do usuário a ser usado para autenticação da conexão

Se especificado, o programa solicitará uma senha para acompanhar esse ID do usuário.

QMgrName
O nome do gerenciador de filas de destino

Se não especificar o gerenciador de filas de destino, a amostra se conecta a qualquer gerenciador de filas que esteja atendendo no nome da conexão TCP/IP especificada.

Nota: Se inserir um ponto de interrogação como o único parâmetro ou se inserir parâmetros incorretos, receberá uma mensagem explicando como usar o programa.

Se executar a amostra sem opções da linha de comandos, os conteúdos da variável de ambiente MQSERVER serão usados para determinar as informações de conexão. (Neste exemplo, MQSERVER é configurado para `SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com`.) Você verá uma saída semelhante a esta:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

Se executar a amostra e fornecer um nome de conexão TCP/IP e um nome de canal de conexão de servidor, mas nenhum nome do gerenciador de filas de destino, desta forma:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

o nome do gerenciador de filas padrão será usado e você verá uma saída semelhante a esta:

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Se executar a amostra e fornecer um nome de conexão TCP/IP e um nome do gerenciador de filas de destino, desta forma:

```
amqscnxc -x machine.site.company.com MACHINE
```

você verá uma saída semelhante a esta:

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

O programa de amostra Data-Conversion

O programa de amostra de conversão de dados é uma estrutura básica de uma rotina de saída de conversão de dados. Saiba mais sobre o design da amostra de conversão de dados.

Consulte “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072 para obter os nomes desses programas.

Design da amostra de conversão de dados

Cada rotina de saída de conversão de dados converte um único formato da mensagem nomeada. Essa estrutura básica é destinada como um wrapper para fragmentos de código gerados pelo programa utilitário da geração de saída de conversão de dados.

O utilitário produz um fragmento de código para cada estrutura de dados. Diversas estruturas compõem um formato, portanto, vários fragmentos de código são incluídos nesta estrutura básica para produzir uma rotina para fazer a conversão de dados do formato inteiro.

O programa, então, verifica se a conversão foi bem-sucedida ou se houve falha e retorna os valores necessários para o responsável pela chamada.

Amostras de coordenação de banco de dados

Duas amostras são fornecidas que demonstram como o IBM MQ pode coordenar atualizações do IBM MQ e atualizações do banco de dados na mesma unidade de trabalho.

Essas amostras são:

1. AMQSXAS0 (em C) ou AMQ0XAS0 (em COBOL), que atualiza um banco de dados único em uma unidade de trabalho do IBM MQ.
2. AMQSXAG0 (em C) ou AMQ0XAG0 (em COBOL), AMQSXAB0 (em C) ou AMQ0XAB0 (em COBOL) e AMQSXAF0 (em C) ou AMQ0XAF0 (em COBOL), que juntas atualizam dois bancos de dados em uma unidade de trabalho do IBM MQ, mostrando como vários bancos de dados podem ser acessados. Essas amostras são fornecidos para mostrar o uso da chamada MQBEGIN, chamadas SQL e IBM MQ combinadas e onde e quando se conectar a um banco de dados.

Figura 128 na página 1097 mostra como as amostras fornecidas são usadas para atualizar os bancos de dados:

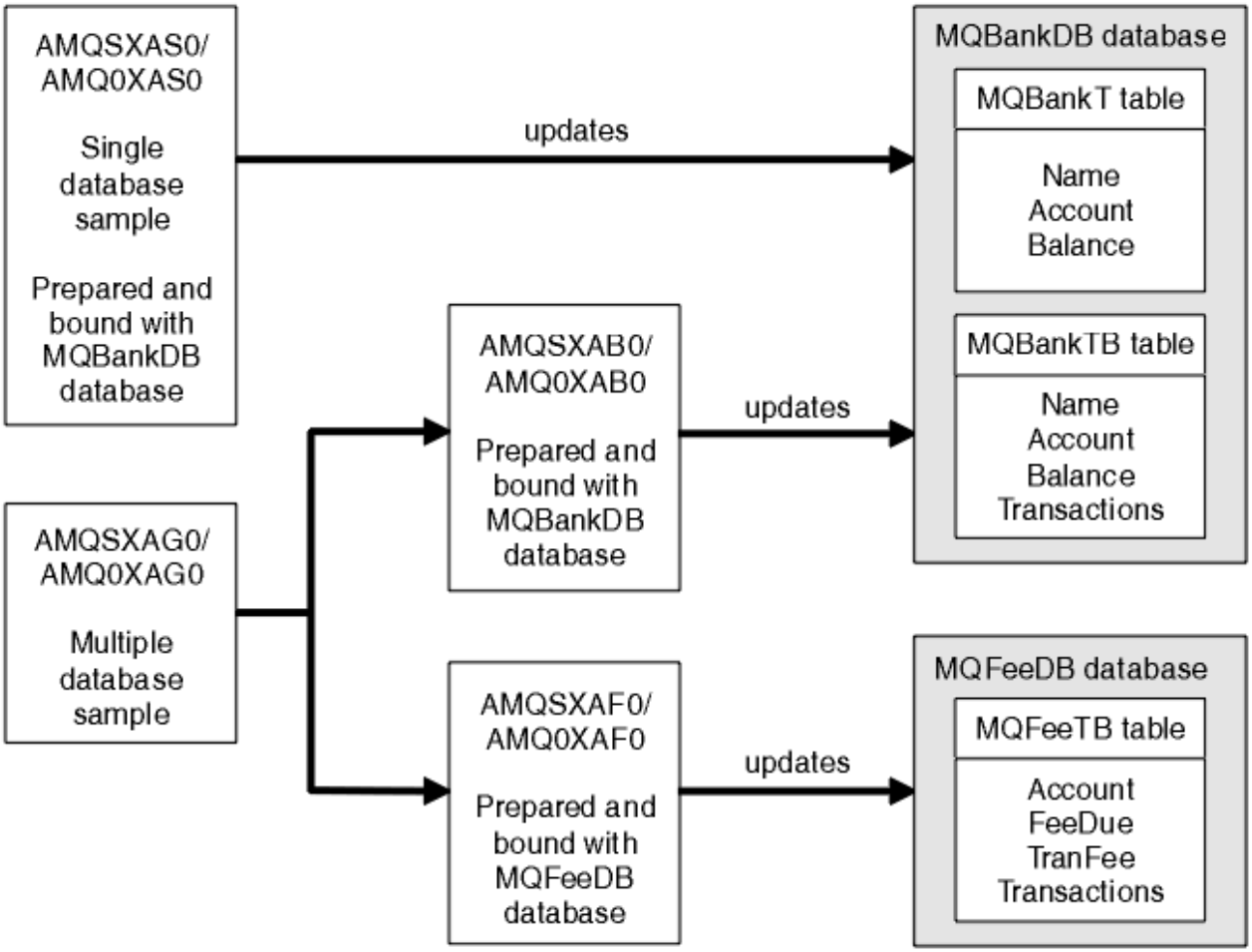


Figura 128. As amostras de coordenação de banco de dados

Os programas leem uma mensagem de uma fila (sob o ponto de sincronização), em seguida, usando as informações na mensagem, obtêm as informações relevantes do banco de dados e atualizam o mesmo. O novo status do banco de dados será, então, impresso.

A lógica do programa é a seguinte:

1. Use o nome da fila de entrada a partir do argumento de programa
2. Conecte-se ao gerenciador de filas padrão (ou, como opção, ao nome fornecido em C) usando MQCONN
3. Abra uma fila (usando MQOPEN) para a entrada enquanto não houver falhas
4. Inicie uma unidade de trabalho usando MQBEGIN
5. Obtenha a próxima mensagem (usando MQGET) da fila sob o ponto de sincronização
6. Obtenha informações de bancos de dados
7. Atualize as informações de bancos de dados
8. Consolide mudanças usando MQCMIT
9. Imprima informações atualizadas (nenhuma mensagem disponível conta como uma falha e o loop é finalizado)
10. Feche a fila usando MQCLOSE
11. Desconecte-se da fila usando MQDISC

Os cursores de SQL são usados nas amostras, de modo que leituras dos bancos de dados (ou seja, diversas instâncias) são bloqueadas enquanto uma mensagem estiver sendo processada, permitindo que

várias instâncias desses programas sejam executadas simultaneamente. Os cursores são explicitamente abertos, mas implicitamente fechados pela chamada MQCMIT.

A amostra de banco de dados único (AMQSXAS0 ou AMQ0XAS0) não tem instruções SQL CONNECT e a conexão com o banco de dados é feita implicitamente pelo IBM MQ com a chamada MQBEGIN. A amostra de diversos banco de dados (AMQSXAG0 ou AMQ0XAG0, AMQSXAB0 ou AMQ0XAB0 e AMQSXAF0 ou AMQ0XAF0) tem instruções SQL CONNECT, pois alguns produtos de banco de dados permitem somente uma conexão ativa. Se esse não for o caso para seu produto de banco de dados ou se você estiver acessando um único banco de dados em produtos de vários bancos de dados, as instruções SQL CONNECT podem ser removidas.

As amostras são preparadas com o produto de banco de dados IBM Db2, portanto, pode ser necessário modificá-las para trabalhar com outros produtos de banco de dados.

A verificação de erros de SQL usa rotinas em UTIL.C e CHECKERR.CBL fornecidas pelo Db2. Elas devem ser compiladas ou substituídas antes da compilação e ligação.

Nota: Se estiver usando o código-fonte CHECKERR.MFC do Micro Focus COBOL para verificação de erro de SQL, deve-se mudar o ID do programa para maiúsculas, ou seja, CHECKERR, para que AMQ0XAS0 faça a ligação corretamente.

Criando os bancos de dados e as tabelas
Crie os bancos de dados e as tabelas antes de compilar as amostras.

Para criar os bancos de dados, use o método usual para seu produto de banco de dados, por exemplo:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Crie as tabelas usando instruções SQL da seguinte forma:

Em C:

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

Em COBOL:

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
```

```
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

Insira os dados nas tabelas usando instruções SQL, da seguinte forma:

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
.
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
.
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
.
```

Nota: Para COBOL, use as mesmas instruções SQL, mas inclua END_EXEC no final de cada linha.

Pré-compilação, compilação e vinculação de amostras
Aprenda sobre a pré-compilação, compilação e vinculação de amostras em C e COBOL.

Pré-compile os arquivos `.SQC` (em C) e os arquivos `.SQB` (em COBOL) e ligue-os com relação ao banco de dados apropriado para produzir os arquivos `.C` ou `.CBL`. Para fazer isto, use o método típico para seu produto de banco de dados.

Pré-compilando em C

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

Pré-compilando em COBOL

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

Compilando e vinculando-se

Os comandos de amostra a seguir usam os símbolos *DB2TOP* e *MQ_INSTALLATION_PATH*. *DB2TOP* representa o diretório de instalação para o produto Db2. O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

- AIX No AIX, o caminho do diretório é:

```
/usr/lpp/db2_05_00
```

- Windows Em sistemas Windows, o caminho do diretório depende do caminho escolhido ao instalar o produto. Se você tiver escolhido as configurações padrão, o caminho será este:

```
c:\sqllib
```

Nota: Antes de emitir o comando de link em sistemas Windows, certifique-se de que a variável de ambiente LIB contenha caminhos para as bibliotecas do Db2 e do IBM MQ.

Copie os arquivos a seguir em um diretório temporário:

- O arquivo `amqsxag0.c` da sua instalação do IBM MQ

  Nota: Esse arquivo pode ser encontrado nos diretórios a seguir:

  – Linux AIX Nos sistemas AIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows Nos sistemas Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Os arquivos `.c` obtidos ao pré-compilar os arquivos de origem `.sqc`, `amqsxas0.sqc`, `amqsxaf0.sqc` e `amqsxab0.sqc`.
- Os arquivos `util.c` e `util.h` da sua instalação do Db2.

  Nota: Esses arquivos podem ser localizados no diretório:

  ```
  DB2TOP/samples/c
  ```

Construa os arquivos de objeto para cada arquivo `.c` usando o comando do compilador a seguir para a plataforma que você está usando:

- AIX AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Windows Sistemas Windows

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

Construa o arquivo executável `amqsxag0` usando o comando de link a seguir para a plataforma que você está usando:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows Sistemas Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

Construa o arquivo executável `amqsxas0` usando os comandos de compilação e de vinculação a seguir para a plataforma que você está usando:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Sistemas Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Informações adicionais

AIX Se você estiver trabalhando no AIX e desejar acessar o Oracle, use o compilador xlc_r e link para libmqm_r.a.

Executando as amostras
Use estas informações para aprender como configurar o gerenciador de filas antes de executar amostras de coordenação de banco de dados em C e COBOL.

Antes de executar as amostras, configure o gerenciador de filas com o produto de banco de dados que está usando. Para obter informações sobre como fazer isso, consulte Cenário 1: o gerenciador de filas executa a coordenação.

O títulos a seguir fornecem informações sobre como executar amostras em C e COBOL:

- “Amostras em C” na página 1101
- “Amostras em COBOL” na página 1102

Amostras em C

As mensagens devem estar no formato a seguir para serem lidas a partir de uma fila:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT pode ser usado para colocar a mensagem na fila.

As amostras de coordenação de banco de dados aceitam dois parâmetros:

1. Nome da fila (obrigatório)
2. Nome do gerenciador de filas (opcional)

Supondo que tenha criado e configurado um gerenciador de filas para a amostra de banco de dados único chamada singDBQM, com uma fila chamada singDBQ, você incrementa a conta do Sr. Fred Bloggs em 50 da seguinte forma:

```
AMQSPUT singDBQ singDBQM
```

Em seguida, digite a mensagem a seguir:

```
UPDATE Balance change=50 WHERE Account=1
```

É possível colocar várias mensagens na fila.

```
AMQSXAS0 singDBQ singDBQM
```

O status atualizado da conta do Sr. Fred Bloggs será, então, impresso.

Supondo que tenha criado e configurado um gerenciador de filas para a amostra de diversos bancos de dados chamada multDBQM, com uma fila chamada multDBQ, você decrementa a conta da Sra. Mary Brown em 75 da seguinte forma:

```
AMQSPUT multDBQ multDBQM
```

Em seguida, digite a mensagem a seguir:

```
UPDATE Balance change=-75 WHERE Account=3
```

É possível colocar várias mensagens na fila.

```
AMQSXAG0 multDBQ multDBQM
```

O status atualizado da conta da Sra. Mary Brown será, então, impresso.

Amostras em COBOL

As mensagens devem estar no formato a seguir para serem lidas a partir de uma fila:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

Para simplificar, `Balance change` deve ser um número com oito caracteres com sinal e `Account` deve ser um número de oito caracteres.

A amostra AMQSPUT pode ser usada para colocar as mensagens na fila.

As amostras não aceitam parâmetros e usam o gerenciador de filas padrão. É possível configurar para que somente uma das amostras seja executada por vez. Supondo que tenha configurado o gerenciador de filas padrão para a amostra de banco de dados único, com uma fila chamada singDBQ, você incrementa a conta do Sr. Fred Bloggs em 50 da seguinte forma:

```
AMQSPUT singDBQ
```

Em seguida, digite a mensagem a seguir:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

É possível colocar várias mensagens na fila:

```
AMQ0XAS0
```

Digite o nome da fila:

```
singDBQ
```

O status atualizado da conta do Sr. Fred Bloggs será, então, impresso.

Supondo que tenha configurado o gerenciador de filas padrão para a amostra de diversos bancos de dados, com uma fila chamada multDBQ, você decrementa a conta da Sra. Mary Brown em 75 da seguinte forma:

```
AMQSPUT multDBQ
```

Em seguida, digite a mensagem a seguir:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

É possível colocar várias mensagens na fila:

```
AMQ0XAG0
```

Digite o nome da fila:

```
multDBQ
```

O status atualizado da conta da Sra. Mary Brown será, então, impresso.

Amostra do manipulador de filas de mensagens não entregues

Um manipulador de fila de mensagens não entregues de amostra é fornecido, o nome da versão executável é `amqsdlq` Se você desejar um manipulador de fila de devoluções diferente de **RUNMQDLQ**, a origem da amostra estará disponível para você usar como sua base.

A amostra é semelhante ao manipulador de mensagens não entregues fornecido no produto, mas o rastreio e relatório de erros são diferentes. Há duas variáveis de ambiente disponíveis para você:

ODQ_TRACE
: Configure como YES ou `yes` para ativar o rastreamento.

ODQ_MSG
: Configure para o nome do arquivo que contém mensagens de erro e de informações. O arquivo fornecido é chamado de `amqsdlq.msg`

É necessário fazer essas variáveis conhecidas para seu ambiente usando os comandos **`export`** ou **`set`** os comandos, dependendo de sua plataforma; o rastreio é desativado usando o comando **`unset`**.

É possível modificar o arquivo de mensagens de erro, `amqsdlq.msg`, para atender aos seus próprios requisitos. A amostra coloca mensagens no stdout, **não** no arquivo de log de erro do IBM MQ.

Para obter mais informações sobre como o manipulador de devoluções funciona e como executá-lo, consulte Processando mensagens em uma IBM MQ fila de devoluções ou o *Guia de Gerenciamento do Sistema* para a sua plataforma

O programa de amostra Distribution List

A amostra Distribution List amqsptl0 fornece um exemplo de como colocar uma mensagem em várias filas de mensagens. Ela é baseada na amostra MQPUT, amqsput0.

Executando a amostra Distribution List, amqsptl0

A amostra Distribution List é executada de maneira semelhante às amostras Put.

Ela aceita os parâmetros a seguir:

- Os nomes das filas
- Os nomes dos gerenciadores de filas

Esses valores são inseridos como pares. Por exemplo:

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

As filas são abertas usando MQOPEN e as mensagens são colocadas nas filas usando MQPUT. Códigos de razão são retornados se qualquer um dos nomes de filas ou de gerenciadores de filas não forem reconhecidos.

Lembre-se de definir canais entre os gerenciadores de filas para que as mensagens possam fluir entre eles. O programa de amostra não faz isso para você.

Design da amostra Distribution List

Put Message Records (MQPMRs) especificam atributos de mensagens para cada destino. A amostra fornece valores para *MsgId* e *CorrelId* e eles substituem os valores especificados na estrutura MQMD.

O campo *PutMsgRecFields* na estrutura MQPMO indica quais campos estão presentes em MQPMRs:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

Em seguida, a amostra aloca os registros de resposta e os registros de objeto. Os registros de objeto (MQORs) requerem pelo menos um par de nomes e um número par de nomes, ou seja, *ObjectName* e *ObjectQMgrName*.

A próxima etapa envolve a conexão com os gerenciadores de filas usando MQCONN. A amostra tenta se conectar ao gerenciador de filas associado à primeira fila no MQOR; se isso falhar, ele passa pelos registros de objeto da vez. Você será informado se não for possível se conectar a nenhum gerenciador de filas e o programa sairá.

As filas de destino são abertas usando MQOPEN e a mensagem é colocada nessas filas usando MQPUT. Quaisquer problemas e falhas são relatados nos registros de resposta (MQRRs).

Por fim, as filas de destino são fechadas usando MQCLOSE e o programa se desconecta do gerenciador de filas usando MQDISC. Os mesmos registros de resposta são usados para cada chamada informando *CompCode* e *Reason*.

Os programas de amostra Echo

Os programas de amostra Echo repetem uma mensagem de uma fila de mensagens para a fila de resposta.

Consulte “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072 para obter os nomes desses programas.

Os programas são destinados a execução como programas acionados.

Nos sistemas IBM i, AIX, Linux, and Windows, sua única entrada é uma estrutura MQTMC2 (mensagem do acionador) que contém o nome de uma fila de destino e o gerenciador de filas. A versão COBOL usa o gerenciador de filas padrão.

IBM i No IBM i, para o processo de acionamento funcionar, assegure-se de que o programa de amostra Echo que você deseja usar seja acionado por mensagens que chegam na fila SYSTEM.SAMPLE.ECHO. Para fazer isso, especifique o nome do programa de amostra Echo que você deseja usar no campo *ApplId* da definição de processo SYSTEM.SAMPLE.ECHOPROCESS. (Para isso, é possível usar o comando CHGMQMPRC; para obter detalhes, consulte Mudar o processo MQ (CHGMQMPRC).) A fila de amostra tem um tipo de acionador de FIRST, portanto, se já houver mensagens na fila antes que a amostra Request seja executada, a amostra Echo não será acionada pelas mensagens enviadas.

Quando você tiver configurado a definição corretamente, primeiro inicie AMQSERV4 em uma tarefa e, em seguida, inicie AMQSREQ4 em outra. Seria possível usar AMQSTRG4 em vez de AMQSERV4, mas os potenciais atrasos de envio de tarefa poderiam tornar mais difícil seguir o que está acontecendo.

Use os programas de amostra Request para enviar mensagens para a fila SYSTEM.SAMPLE.ECHO. Os programas de amostra Echo enviam uma mensagem de resposta contendo os dados na mensagem de solicitação para a fila de resposta especificada na mensagem de solicitação.

Design dos programas de amostra Echo

O programa abre a fila denominada na estrutura da mensagem do acionador que foi passada quando ele foi iniciado. (Para clareza, isso é referido como fila de solicitações.) O programa usa a chamada MQOPEN para abrir essa fila para entrada compartilhada.

O programa usa a chamada MQGET para remover as mensagens dessa fila. Essa chamada usa as opções MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT e MQGMO_WAIT, com um intervalo de espera de 5 segundos. O programa testa o descritor de cada mensagem para ver se é uma mensagem de solicitação; se não for, o programa descartará a mensagem e exibirá uma mensagem de aviso.

Para cada linha de entrada, o programa lê o texto em um buffer e usa a chamada MQPUT1 para colocar uma mensagem de solicitação, contendo o texto dessa linha, na fila de resposta.

Se a chamada MQGET falhar, o programa colocará uma mensagem de relatório na fila de resposta, configurando o campo *Feedback* do descritor de mensagens como o código de razão retornado pelo MQGET.

Quando não houver nenhuma mensagem restante na fila de solicitações, o programa fecha essa fila e desconecta do gerenciador de filas.

IBM i No IBM i, o programa também pode responder às mensagens enviadas para a fila a partir de plataformas diferentes de IBM MQ for IBM i, embora nenhuma amostra seja fornecida para esta situação. Para fazer o programa ECHO funcionar:

- Grave um programa, especificando corretamente os parâmetros **Format**, **Encoding** e **CCSID**, para enviar mensagens de solicitação de texto.

  O programa ECHO solicita que o gerenciador de filas execute a conversão de dados da mensagem, se isto for necessário.
- Especifique CONVERT(*YES) no canal de envio do IBM MQ for IBM i, se o programa que você gravou não fornecer conversão similar para a resposta.

Os programas de amostra Get

Os programas de amostra Get recebem mensagens de uma fila usando a chamada MQGET.

Consulte “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072 para obter os nomes desses programas.

Design do programa de amostra Get

O programa abre a fila de destino usando a chamada MQOPEN com a opção MQOO_INPUT_AS_Q_DEF. Se não puder abrir a fila, o programa exibirá uma mensagem de erro contendo o código de razão retornado pela chamada MQOPEN.

Para cada mensagem na fila, o programa usa a chamada MQGET para remover a mensagem da fila e, em seguida, exibe os dados contidos na mensagem. A chamada MQGET usa a opção MQGMO_WAIT, especificando um *WaitInterval* de 15 segundos, para que o programa aguarde esse período se não houver nenhuma mensagem na fila. Se nenhuma mensagem chegar antes desse intervalo expirar, a chamada falhará e retornará o código de razão MQRC_NO_MSG_AVAILABLE.

O programa demonstra como os campos *MsgId* e *CorrelId* devem ser desmarcados da estrutura MQMD após cada chamada MQGET porque a chamada configura esses campos como os valores contidos na mensagem que ela recupera. Desmarcar esses campos significa que sucessivas chamadas MQGET recuperam as mensagens na ordem em que elas são retidas na fila.

A chamada MQGET especifica um buffer de tamanho fixo. Se uma mensagem for maior do que esse buffer, a chamada falhará e o programa irá parar.

O programa continua até que a chamada MQGET retorne o código de razão MQRC_NO_MSG_AVAILABLE ou a chamada MQGET falhe. Se a chamada falhar, o programa exibirá uma mensagem de erro que contém o código de razão.

O programa, então, fecha a fila usando a chamada MQCLOSE.

Executando as amostras amqsget e amqsgetc

Cada um desses programas aceita os parâmetros posicionais a seguir:

1. O nome da fila de origem (obrigatório)
2. O nome do gerenciador de filas (opcional)

   Se um gerenciador de filas não for especificado, o **amqsget** se conectará ao gerenciador de filas padrão e o **amqsgetc** se conectará ao gerenciador de filas identificado pela variável de ambiente *MQSERVER* ou pelo arquivo de definição de canal do cliente...
3. As opções de abertura (opcional)

   Se as opções de abertura não forem especificadas, a amostra usa um valor de 8193, que é a combinação destas duas opções:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. As opções de fechamento (opcional)

   Se as opções de fechamento não forem especificadas, a amostra usará um valor de 0, que é MQCO_NONE.

Use as variáveis de ambiente a seguir para fornecer credenciais que são usadas para autenticar com o gerenciador de filas:

MQSAMP_USER_ID
Configure como o ID do usuário a ser usado para autenticação de conexão, se você desejar usar um ID do usuário e uma senha para autenticar com o gerenciador de filas. O programa solicita a senha para acompanhar o ID do usuário.

Linux V 9.4.0 AIX **MQSAMP_TOKEN**
Configure como um valor não em branco se desejar fornecer um token de autenticação para autenticar com o gerenciador de filas. O programa solicita o token de autenticação. Os tokens de autenticação podem ser usados somente pela amostra **amqsgetc** que usa ligações do cliente

Para executar esses programas, insira uma das opções a seguir:

- `amqsget` *`myqueue qmanagername`*
- `amqsgetc` *`myqueue qmanagername`*

em que *myqueue* é o nome da fila a partir da qual o programa receberá mensagens e *qmanagername* é o gerenciador de filas que tem *myqueue*.

Usando amqsget e amqsgetc

Observe que **amqsget** executa uma conexão local com o gerenciador de filas usando a memória compartilhada para se conectar ao gerenciador de filas e, como tal, só pode ser executada no sistema em que o gerenciador de filas reside, enquanto **amqsgetc** executa uma conexão no estilo do cliente (mesmo se está se conectando a um gerenciador de filas no mesmo sistema).

Ao usar **amqsgetc**, é necessário fornecer os detalhes do aplicativo de como realmente chegar ao gerenciador de filas, em termos do host ou do endereço IP do gerenciador de filas e da porta do listener do gerenciador de filas.

Normalmente, isso é feito usando a variável de ambiente **MQSERVER** ou definindo detalhes da conexão usando uma tabela de definição de canal do cliente, que também pode ser fornecida para **amqsgetc** usando variáveis de ambiente; por exemplo, consulte MQCCDTURL.

Um exemplo usando **MQSERVER**, conectando-se a um gerenciador de filas localmente, que possui um listener em execução na porta 1414 e usando o canal de conexão do servidor padrão é:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

Programas de amostra de alta disponibilidade

Os programas de amostra de alta disponibilidade **amqsghac**, **amqsphac** e **amqsmhac** usam reconexão do cliente automatizada para demonstrar recuperação após a falha de um gerenciador de filas. **amqsfhac** verifica se um gerenciador de filas usando armazenamento em rede mantém a integridade de dados após uma falha.

Os programas **amqsghac**, **amqsphac** e **amqsmhac** são iniciados a partir da linha de comandos e podem ser usados em combinação para demonstrar a reconexão após a falha de uma instância de um gerenciador de filas de várias instâncias.

Como alternativa, também é possível usar as amostras **amqsghac**, **amqsphac** e **amqsmhac** para demonstrar reconexão do cliente a gerenciadores de filas de instância única, geralmente configurados em um grupo de gerenciadores de filas.

Para manter o exemplo simples, para facilitar a configuração, serão mostrados os programas de amostra reconectando a um gerenciador de filas de instância única iniciado, interrompido e, em seguida, reiniciado novamente; consulte "Configurar e controlar o gerenciador de filas" na página 1109.

Use **amqsfhac** em paralelo com **amqmfsck** para verificar a integridade do sistema de arquivos. Consulte **amqmfsck** (verificação do sistema de arquivos) e Verificando o comportamento do sistema de arquivo compartilhado comportamento para obter mais informações.

amqsphac *queueName* [*qMgrName*]

- **amqsphac** é um aplicativo IBM MQ MQI client. Ele coloca uma sequência de mensagens em uma fila com um atraso de dois segundos entre cada mensagem e exibe eventos enviados ao seu manipulador de eventos.
- Nenhum ponto de sincronização é usado para colocar mensagens na fila.
- A reconexão pode ser feita para qualquer gerenciador de filas no mesmo grupo de gerenciadores de filas.

amqsghac *queueName* [*qMgrName*]

- **amqsghac** é um aplicativo IBM MQ MQI client. Ele obtém mensagens de uma fila e exibe eventos enviados ao seu manipulador de eventos.
- Nenhum ponto de sincronização é usado para obter mensagens da fila.
- A reconexão pode ser feita para qualquer gerenciador de filas no mesmo grupo de gerenciadores de filas.

amqsmhac -s *sourceQueueName* -t *targetQueueName* [-m *qMgrName*] [-w *waitInterval*]

- **amqsmhac** é um aplicativo IBM MQ MQI client. Ele copia mensagens de uma fila para outra com um intervalo de espera padrão de 15 minutos após a última mensagem recebida antes da conclusão do programa.
- As mensagens são copiadas dentro do ponto de sincronização.
- A reconexão pode ser feita somente no mesmo gerenciador de filas.

amqsfhac *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* (0 | 1 | 2)

- **amqsfhac** é um aplicativo IBM MQ MQI client. Ele verifica se um gerenciador de filas de várias instâncias do IBM MQ que está usando armazenamento em rede, como um NAS ou um sistema de arquivos de cluster, mantém a integridade dos dados. Siga as etapas para executar **amqsfhac** em Verificando o comportamento do sistema de arquivo compartilhado.
- Ele usa a opção `MQCNO_RECONNECT_Q_MGR` ao se conectar ao *QueueManagerName*. Reconecta automaticamente quando o gerenciador de filas efetua failover.
- Coloca mensagens persistentes *InTransactionCount*RepeatCount* em *QueueName* enquanto isso você faz o gerenciador de filas efetuar failover um número qualquer de vezes. **amqsfhac** reconecta-se ao gerenciador de filas toda vez e continua. O teste é para assegurar que nenhuma mensagem seja perdida.

- Mensagens *InTransactionCount* são colocadas dentro de cada transação. A transação é repetida *RepeatCount* número de vezes. Se ocorrer uma falha em uma transação, **amqsfhac** recupera e reenvia a transação quando **amqsfhac** reconectar ao gerenciador de filas.
- Ele também coloca mensagens em *SideQueueName*. Usa *SideQueueName* para verificar se o todas as mensagens estão confirmadas ou retrocedidas do *QueueName* com sucesso. Se detectar uma inconsistência, grava uma mensagem de erro.
- Varie a quantia de rastreio de saída de **amqsfhac** configurando o último parâmetro para `(0|1|2)`.

  0
  Menos saída.

  1
  Saída moderada.

  2
  Mais saída.

Configurando uma conexão do cliente

Você precisa configurar um canal de conexão do cliente e do servidor para executar as amostras. O procedimento de verificação do cliente explica como configurar um ambiente de teste do cliente.

Como alternativa, use a configuração fornecida no exemplo a seguir.

Exemplo de uso de amqsghac, amqsphac e amqsmhac

O exemplo demonstra clientes reconectáveis usando um gerenciador de filas de instância única.

Mensagens são colocadas na fila SOURCE por **amqsphac**, transferidas para TARGET por **amqsmhac** e recuperadas de TARGET por **amqsghac**; consulte Figura 129 na página 1108.

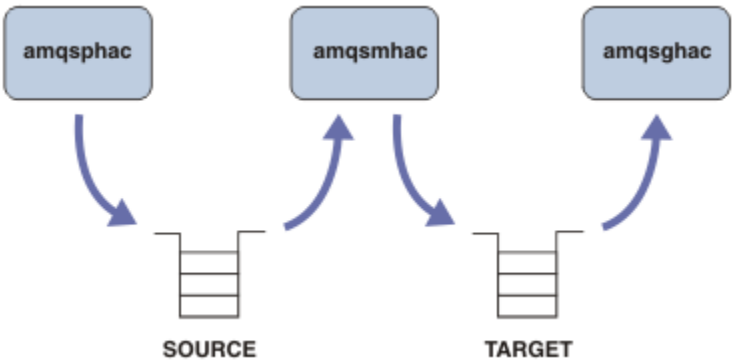


Figura 129. Amostras de clientes reconectáveis

Siga estas etapas para executar as amostras.

1. Crie um arquivo `hasamples.tst` contendo os comandos:

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. Digite os comandos a seguir em um prompt de comandos:

   a. `crtmqm QM1`

   b. `strmqm QM1`

c. `runmqsc QM1 < hasamples.tst`

3. Configure a variável de ambiente **MQCHLLIB** para o caminho para o arquivo de definição de canal do cliente AMQCLCHL.TAB; por exemplo, `SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Abra três novas janelas com **MQCHLLIB** configurado; por exemplo, no Windows, digite **start** três vezes no prompt de comandos anterior iniciando cada programa em uma das janelas. Consulte a etapa “5” na página 1110 em “Configurar e controlar o gerenciador de filas” na página 1109.)
5. Digite o comando `endmqm -r -p QM1` para parar o gerenciador de filas e, em seguida, permita que os clientes se reconectem.
6. Digite o comando `strmqm QM1` para reiniciar o gerenciador de filas.

Os resultados da execução das amostras **amqsghac**, **amqsphac** e **amqsmhac** no Windows são mostradas nos exemplos a seguir.

Configurar e controlar o gerenciador de filas

1. Crie o gerenciador de filas.

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Lembre-se do diretório de dados para configurar a variável **MQCHLLIB** posteriormente.

2. Inicie o gerenciador de filas.

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. Crie as filas e os canais, modifique a porta do listener e inicie o listener e o canal.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. Torne a tabela do canal do cliente conhecida para os clientes.

Use o diretório de dados retornado do comando **crtmqm** na etapa “1” na página 1109 e inclua o diretório `@ipcc` nele para configurar a variável **MQCHLLIB**.

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. Iniciar os programas de amostra nas outras janelas

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. Finalize o gerenciador de filas e reinicie-o novamente.

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

Tarefas relacionadas

Verificando o comportamento do sistema de arquivo compartilhado

Referências relacionadas

amqmfsck (verificação de sistema de arquivos)

Os programas de amostra Inquire

Programas de amostra Inquire consultam sobre alguns dos atributos de uma fila usando a chamada MQINQ.

Consulte “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072 para obter os nomes desses programas.

Esses programas são destinados a serem executados como programas acionados, portanto, sua única entrada é uma estrutura MQTMC2 (mensagem do acionador) para IBM MQ for Multiplatforms. Essa estrutura contém o nome de uma fila de destino com atributos que devem ser consultados. A versão de C também usa o nome do gerenciador de filas. A versão COBOL usa o gerenciador de filas padrão.

Para o processo de acionamento trabalhar, assegure-se de que o programa de consulta de amostra que você deseja usar seja acionado pelas mensagens que chegam na fila de SYSTEM.SAMPLE.INQ. Para fazer isso, especifique o nome do programa de amostra Inquire que desejar usar no campo *ApplicId* da definição de processo SYSTEM.SAMPLE.INQPROCESS. IBM i Para o IBM i, é possível usar o comando CHGMQMPRC para isso; para obter detalhes, consulte Mudar processo MQ (CHGMQMPRC). A fila de amostra tem um tipo de acionador FIRST; se já houver mensagens na fila antes de executar a amostra de solicitação, a amostra de consulta não será acionada pelas mensagens enviadas por você.

Quando a definição tiver sido configurada corretamente:

- ALW Para AIX, Linux, and Windows, inicie o programa **runmqtrm** em uma sessão; em seguida, inicie o programa `amqsreq` em outra.
- IBM i Para o IBM i, inicie o programa AMQSERV4 em uma sessão, em seguida, inicie o programa AMQSREQ4 em outra. Seria possível usar AMQSTRG4 em vez de AMQSERV4, mas os potenciais atrasos de envio de tarefa poderiam tornar mais difícil seguir o que está acontecendo.

Use os programas de amostra Request para enviar mensagens de pedido, cada uma contendo apenas um nome de fila para a fila SYSTEM.SAMPLE.INQ. Para cada mensagem de solicitação, os programas de amostra de consulta enviarão uma mensagem de resposta contendo informações sobre a fila especificada na mensagem de pedido. As respostas são enviadas à fila de resposta especificada na mensagem de solicitação.

IBM i No IBM i, se o arquivo de entrada de amostra membro QMQMSAMP.AMQSDATA(INQ) for usado, a última fila denominada não existe, então a amostra retorna uma mensagem de relatório com um código de razão para a falha.

Design do programa de amostra de consulta

O programa abre a fila denominada na estrutura da mensagem do acionador que foi passada quando ele foi iniciado. (Por questão de clareza, chamaremos isso de *fila de solicitações*.) O programa usa a chamada MQOPEN para abrir essa fila para entrada compartilhada.

O programa usa a chamada MQGET para remover as mensagens dessa fila. Essa chamada usa as opções MQGMO_ACCEPT_TRUNCATED_MSG e MQGMO_WAIT com um intervalo de espera de 5 segundos. O programa testa o descritor de cada mensagem para ver se é uma mensagem de solicitação; se não for, o programa descartará a mensagem e exibirá uma mensagem de aviso.

Para cada mensagem de solicitação removida da fila de pedidos, o programa lê o nome da fila (que chamaremos de *fila de destino*) contido nos dados e abre essa fila usando a chamada MQOPEN com a opção MQOO_INQ. O programa, então, usa a chamada MQINQ para consultar sobre os valores dos atributos *InhibitGet*, **CurrentQDepth** e **OpenInputCount** da fila de destino.

Se a chamada MQINQ for bem-sucedida, o programa usará a chamada MQPUT1 para colocar uma mensagem de resposta na fila de resposta. Essa mensagem contém os valores dos três atributos.

Se a chamada MQOPEN ou MQINQ for mal sucedida, o programa usará a chamada MQPUT1 para colocar uma mensagem de relatório na fila de resposta. No campo *Feedback* do descritor de mensagens dessa mensagem de relatório está o código de razão retornado pela chamada MQOPEN ou MQINQ, dependendo de qual delas falhou.

Após a chamada MQINQ, o programa fecha a fila de destino usando a chamada MQCLOSE.

Quando não houver nenhuma mensagem restante na fila de solicitações, o programa fecha essa fila e desconecta do gerenciador de filas.

O programa de amostra Inquire Properties of a Message Handle

AMQSIQMA é um programa C de amostra para consultar propriedades de um identificador de mensagens a partir de uma fila de mensagens e é um exemplo de uso da chamada de API MQINQMP.

Essa amostra cria uma manipulação de mensagem e a coloca no campo MsgHandle da estrutura MQGMO. A amostra, então, obtém uma mensagem e consulta e imprime todas as propriedades com as quais a manipulação de mensagem foi preenchida.

```
C:\Program Files\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

Os programas de amostra Publish/Subscribe

Os programas de amostra Publish/Subscribe demonstram o uso dos recursos de publicação e assinatura no IBM MQ.

Há três programas de amostra na linguagem C que ilustram como programar para a interface de publicar/assinar do IBM MQ. Há algumas amostras em C que usam as interfaces mais antigas e há amostras em Java. As amostras em Java usam a interface de publicar/assinar do IBM MQ em com.ibm.mq.jar e a interface de publicar/assinar do JMS em com.ibm.mqjms. As amostras do JMS não são cobertas neste tópico.

C

Localize a amostra do publicador `amqspub` na pasta de amostras C. Execute-a com qualquer nome de tópico que quiser como o primeiro parâmetro, seguido por um nome do gerenciador de filas opcional. Por exemplo, `amqspub mytopic QM3`. Há também uma versão do cliente chamada `amqspubc`. Se optar por executar a versão de cliente, primeiro consulte “Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms” na página 1081 para obter detalhes.

O publicador se conecta ao gerenciador de filas padrão e responde com a saída, `target topic is mytopic`. Cada linha que inserir nessa janela de agora em diante será publicada em `mytopic`.

Abra outra janela de comando no mesmo diretório e execute o programa do assinante, `amqssub`, fornecendo-o com o mesmo nome de tópico e um nome de gerenciador de filas opcional. Por exemplo, `amqssub mytopic QM3`.

O assinante responde com a saída, `Calling MQGET : 30 seconds wait time`. De agora em diante, as linhas que digitar no publicador aparecem na saída do assinante.

Inicie outro assinante em outra janela de comandos e observe ambos os assinantes receberem publicações.

Para a documentação completa dos parâmetros, incluindo opções de configuração, consulte o código-fonte de amostra. Os valores para o campo de opções do assinante estão descritos no tópico a seguir: Opções (MQLONG).

Há outra amostra de assinante `amqssbx`, que oferece opções de assinatura adicionais como comutadores de linha de comandos.

Digite `amqssbx -d mysub -t mytopic -k` para chamar o assinante que está usando assinaturas duráveis que são retidas após o assinante ter finalizado.

Teste a assinatura publicando outro item usando o publicador. Espere 30 segundos até o assinante finalizar. Publique mais alguns itens sob o mesmo tópico. Reinicie o assinante. O último item publicado enquanto o assinante não estava em execução será exibido pelo assinante imediatamente ao ser reiniciado.

Legado de C

Há um conjunto adicional de amostras C que demonstram os comandos enfileirados. Algumas dessas amostras foram enviadas originalmente como parte do MQ0C Supportpac. O recursos que as amostras demonstram são totalmente suportados por razões de compatibilidade.

Desaconselhamos o uso da interface de comando enfileirada. É muito mais complexa do que a API de publicar/assinar e não há razão funcional convincente para programar comandos enfileirados complexos. No entanto, você pode achar a abordagem enfileirada mais adequada, talvez porque já esteja usando a interface ou porque seu ambiente de programação facilita a construção de uma mensagem complexa e chamar uma chamada MQPUT genérica, em vez de construir diferentes chamadas para MQSUB.

As amostras adicionais estão localizadas no subdiretório `pubsub` na pasta `samples`.

Há seis tipos de amostras listados em Tabela 163 na página 1113.

Tabela 163. Categorias de programas C de amostra Publish/Subscribe de legado

Categoria	Programas	Comments
RFH1	`amqssr1a.c` `amqspr1a.c`	Exemplo simples de publicar/assinar construído usando mensagens de formato RFH1.
RFH2	`amqssr2a.c` `amqspr2a.c`	Exemplo simples de publicar/assinar construído usando mensagens de formato RFH2.
Amostras MQAI	`amqsppca.c` `amqsspca.c`	Exemplo simples de publicar/assinar construído usando os comandos PCF e a interface de comandos MQAI.
Serviço MA0C Results usando RFH1	`amqsgama.c` `amqsresa.c`	Serviço Results construído usando cabeçalhos RFH1 1. Requer as filas definidas no `amqsgama.tst` e no `amqsresa.tst` 2. `amqsresa` deve ser iniciado antes de `amqsgama`
Serviço MA0C Results usando RFH2	`amqsgr2a.c` `amqsrr2a.c`	Serviço Results construído usando cabeçalhos RFH2 1. Requer as filas definidas no `amqsgama.tst` e no `amqsresa.tst` 2. `amqsresa` deve ser iniciado antes de `amqsgama`
Amostra Publish/ Subscribe da saída de roteamento	`amqspsra.c`	Demonstra como mudar o destino da fila ou do gerenciador de filas para uma mensagem de publicar/assinar em uma saída de roteamento.

Programa de amostra para o Java

A amostra Java `MQPubSubApiSample.java` combina publicador e assinantes em um único programa. Seus arquivos de origem e de classe compilada são encontrados na pasta de amostras `wmqjava`.

Se optar por executar no modo de cliente, primeiro consulte "Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms" na página 1081 para obter detalhes.

Execute a amostra a partir da linha de comandos usando o comando Java, se você tiver um ambiente Java configurado. Também é possível executar a amostra a partir da área de trabalho do Eclipse IBM MQ Explorer que tem um ambiente de trabalho de programação Java já configurado.

Talvez seja necessário mudar algumas das propriedades do programa de amostra para executá-lo. Você faz isso fornecendo parâmetros para a JVM ou editando a origem.

As instruções em "Executando a amostra Java MQPubSubApiSample" na página 1114 mostram como executar a amostra a partir da área de trabalho do Eclipse.

Executando a amostra Java MQPubSubApiSample
Como executar MQPubSubApiSample usando o Java Development Tools a partir da plataforma Eclipse.

Antes de começar

Abra o ambiente de trabalho do Eclipse. Crie um novo diretório de área de trabalho e selecione-o. Feche a janela de boas-vindas.

Siga as etapas em "Configurando um gerenciador de filas para aceitar conexões de clientes no Multiplatforms" na página 1081 antes de executar como um cliente.

Sobre esta tarefa

O programa de amostra de publicação/assinatura Java é um programa IBM MQ MQI client Java. A amostra é executada sem modificação usando um gerenciador de filas padrão que está atendendo na porta 1414. A tarefa descreve esse caso simples e indica em termos gerais como fornecer parâmetros e modificar a amostra para atender às diferentes configurações do IBM MQ. O exemplo é ilustrado em execução no Windows. Os caminhos de arquivo serão diferentes em outras plataformas.

Procedimento

1. Importar os programas de amostra Java
   a) No ambiente de trabalho, clique em **Janela** > **Abrir perspectiva** > **Outro** > **Java** e clique em **OK**.
   b) Alterne para a visualização **Package Explorer**.
   c) Clique com o botão direito no espaço em branco na visualização **Package Explorer**. Clique em **Novo** > **Projeto Java**.
   d) No **`Project name`** tipo de campo `MQ Java Samples`. Clique em **Avançar**.
   e) No painel **`Java Settings`** , alterne para a guia **Bibliotecas** .
   f) Clique em **Incluir JARs externos**.
   g) Navegue até *MQ_INSTALLATION_PATH* `\java\lib` em que *MQ_INSTALLATION_PATH* é a pasta de instalação do IBM MQ e selecione `com.ibm.mq.jar` e `com.ibm.mq.jmqi.jar`
   h) Clique em **Abrir** > **Concluir.**
   i) Clique com o botão direito do mouse em `src` na visualização **Package Explorer**.
   j) Selecione **Importar...** > **Geral** > **Sistema de Arquivos** > **Avançar** > **Procurar**... e procure o caminho *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` em que *MQ_INSTALLATION_PATH* é o diretório de instalação IBM MQ .
   k) No painel **Importar**, Figura 130 na página 1115, clique em `samples` (não selecione a caixa de seleção).
   l) Selecione `MQPubSubApiSample.java`. O campo **`Into folder`** deve conter `MQ Java Samples/src`.. Clique em **Concluir**.

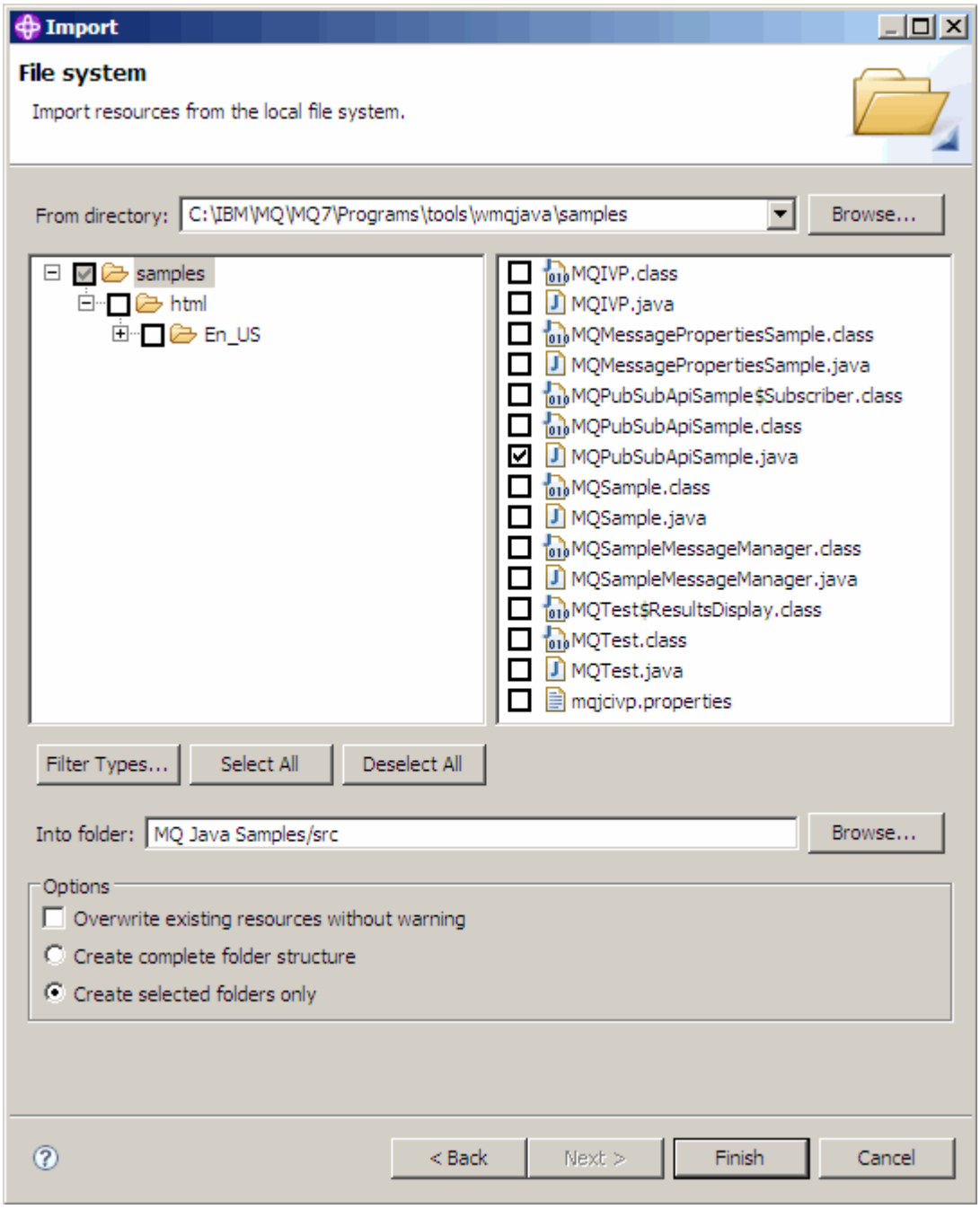


Figura 130. Importação do sistema de arquivos

2. Execute o programa de amostra de publicar/assinar.

   Há duas maneiras de executar o programa, dependendo de se você precisa mudar os parâmetros padrão.

   - A primeira opção executa o programa sem fazer nenhuma mudança:
     - No menu principal da área de trabalho, expanda a pasta `src`. Clique com o botão direito do mouse em **MQPubSubApiSample.java Executar como** > **1. Java Aplicativo**
   - A segunda opção executa o programa com parâmetros ou com código-fonte modificado para seu ambiente:
     - Abra o `MQPubSubApiSample.java` e estude o construtor `MQPubSubApiSample`.

– Modifique os atributos do programa.

  Esses atributos podem ser modificados usando o comutador -D JVM ou fornecendo um valor padrão para System pProperty editando o código-fonte.

  - topicObject
  - queueManagerName
  - subscriberCount

  Esses atributos podem ser mudados somente editando o código-fonte no construtor.

  - nome do host
  - port
  - channel

  Para configurar System pProperties, codifique um valor padrão no acessador, por exemplo:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

  Ou forneça o parâmetro para a JVM usando a opção `-D`, conforme mostrado nas etapas a seguir:

a. Copie o nome completo do System.Property que deseja configurar, por exemplo: `com.ibm.mq.pubSubSample.queueManagerName`.
b. Na área de trabalho, clique com o botão direito em **Executar** > **Abrir diálogo de execução**. Clique duas vezes em Aplicativo Java em **Criar, gerenciar e executar aplicativos** e clique na guia **(x) = Argumentos**.
c. Na área de janela **Argumentos da VM:**, digite `-D` e cole o nome de System.property, `com.ibm.mq.pubSubSample.queueManagerName`, seguido por `=QM3`. Clique em **Aplicar** > **Executar**.
d. Inclua argumentos adicionais como uma lista separada por vírgulas ou como linhas adicionais na área de janela, sem separadores de vírgula.

   Por exemplo: `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3, -Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

O programa de amostra Publish Exit

AMQSPSE0 é um programa C de amostra de uma saída para interceptar uma publicação antes de ser entregue a um assinante. A saída pode então, por exemplo, alterar os cabeçalhos, a carga útil ou o destino da da mensagem ou impedir que a mensagem seja publicada a um assinante.

Para executar a amostra, execute as tarefas a seguir:

1. Configure o gerenciador de filas:
   - Linux AIX Nos sistemas AIX and Linux, inclua uma sub-rotina como esta no arquivo `qm.ini`:

```
PublishSubscribe:
PublishExitPath=Module
PublishExitFunction=EntryPoint
```

     em que o módulo é *MQ_INSTALLATION_PATH*`/samp/bin/amqspse.`. O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.
   - Windows No Windows, configure os atributos equivalentes no registro.
2. Certifique-se de que o Module esteja acessível para o IBM MQ.
3. Reinicie o gerenciador de filas para selecionar a configuração.

4. No processo do aplicativo a ser rastreado, descreva onde os arquivos de rastreio devem ser gravados. Por exemplo:
   - Linux AIX Em sistemas AIX and Linux , assegure-se de que o diretório `/var/mqm/trace` exista e exporte a variável de ambiente **MQPSE_TRACE_LOGFILE** :

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows No Windows, assegure-se de que o diretório `C:\temp` exista e configure a variável de ambiente **MQPSE_TRACE_LOGFILE** :

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

Os programas de amostra Put

Os programas de amostra Put colocam mensagens em uma fila usando a chamada MQPUT.

Consulte "Recursos demonstrados nos programas de amostra em multiplataformas" na página 1072 para obter os nomes desses programas.

Design do programa de amostra Put

O programa usa a chamada MQOPEN com a opção MQOO_OUTPUT para abrir a fila de destino para colocar mensagens.

Se ele não puder abrir a fila, o programa envia uma mensagem de erro contendo o código de razão retornado pela chamada MQOPEN. Para manter o programa simples, nesta e em chamadas MQI subsequentes, o programa usa valores padrão para muitas das opções.

Para cada linha de entrada, o programa lê o texto em um buffer e usa a chamada MQPUT para criar uma mensagem de datagrama que contém o texto dessa linha. O programa continua até que ele atinja o final da entrada ou a chamada MQPUT falhará. Se o programa atingir o final da entrada, ele fechará a fila usando a chamada MQCLOSE.

Executando os programas de amostra Put

Executando as amostras amqsput e amqsputc

ALW

A amostra **amqqsput** é o programa para colocar mensagens usando ligações locais e a amostra **amqsputc** é o programa para colocar mensagens usando ligações clientes. Cada um desses programas aceita os parâmetros posicionais a seguir:

1. O nome da fila de destino (obrigatório)
2. O nome do gerenciador de filas (opcional)

   Se um gerenciador de filas não for especificado, o **amqsput** se conectará ao gerenciador de filas padrão e o **amqsputc** se conectará ao gerenciador de filas identificado pela variável de ambiente *MQSERVER* ou pelo arquivo de definição de canal do cliente...
3. As opções de abertura (opcional)

   Se as opções de abertura não forem especificadas, a amostra usará um valor de 8208, que é a combinação dessas duas opções:

   - MQOO_OUTPUT
   - MQOO_FAIL_IF_QUIESCING
4. As opções de fechamento (opcional)

   Se as opções de fechamento não forem especificadas, a amostra usará um valor de 0, que é MQCO_NONE.

5. O nome do gerenciador de filas de destino (opcional)

   Se um gerenciador de filas de destino não for especificado, o campo `ObjectQMgrName` no MQOD será deixado em branco.

6. O nome da fila dinâmica (opcional)

   Se um nome de fila dinâmica não for especificado, o campo `DynamicQName` no MQOD será deixado em branco.

Use as variáveis de ambiente a seguir para fornecer credenciais que são usadas para autenticar com o gerenciador de filas:

MQSAMP_USER_ID
Configure como o ID do usuário a ser usado para autenticação de conexão, se você desejar usar um ID do usuário e uma senha para autenticar com o gerenciador de filas. O programa solicita a senha para acompanhar o ID do usuário.

Linux V 9.4.0 AIX **MQSAMP_TOKEN**
Configure como um valor não em branco se desejar fornecer um token de autenticação para autenticar com o gerenciador de filas. O programa solicita o token de autenticação. Os tokens de autenticação podem ser usados somente pela amostra **amqsputc** que usa ligações do cliente

Para executar esses programas, insira um dos seguintes comandos:

- `amqsput` *`myqueue qmanagername`*
- `amqsputc` *`myqueue qmanagername`*

em que *myqueue* é o nome da fila na qual as mensagens serão colocadas e *qmanagername* é o gerenciador de filas que possui *myqueue*.

Executando a amostra amq0put

ALW

A versão em COBOL não tem nenhum parâmetro. Ela se conecta ao gerenciador de filas padrão e quando executá-la, será solicitado o seguinte:

```
Please enter the name of the target queue
```

Ela obtém entrada de StdIn e inclui cada linha de entrada na fila de destino. Uma linha em branco indica que não há mais dados.

Executando a amostra C AMQSPUT4 (IBM i)

IBM i

O programa C AMQSPUT4, disponível apenas para a plataforma IBM i, cria mensagens lendo dados de um membro de um arquivo de origem.

Deve-se especificar o nome do arquivo como um parâmetro quando iniciar o programa. A estrutura do arquivo deve ser:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

Uma amostra de entrada para as amostras put é fornecida no PUT do membro AMQSDATA do arquivo QMQMSAMP da biblioteca.

Nota: Lembre-se de que os nomes de filas fazem distinção entre maiúsculas e minúsculas. Todas as filas criadas pelo programa de criação de arquivo de amostra AMQSAMP4 têm nomes criados em caracteres maiúsculos.

O programa C coloca mensagens na fila nomeada na primeira linha do arquivo; é possível usar a fila fornecida SYSTEM.SAMPLE.LOCAL. O programa coloca o texto de cada uma das linhas a seguir em mensagens separadas do datagrama e para ao ler uma linha em branco no fim do arquivo.

Usando o arquivo de dados de exemplo, o comando é:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

Executando a amostra COBOL AMQ0PUT4 (IBM i)

IBM i

O programa COBOL AMQ0PUT4, disponível apenas na plataforma IBM i, cria mensagens aceitando dados do teclado.

Para iniciar o programa, chame o programa e forneça o nome de sua fila de destino como um parâmetro de programa. O programa aceita entrada do teclado em um buffer e cria uma mensagem de datagrama para cada linha de texto. O programa para quando você insere uma linha em branco no teclado.

Os programas de amostra Reference Message

As amostras de Mensagem de Referência permitem que um grande objeto seja transferido de um nó para outro (geralmente em sistemas diferentes) sem a necessidade de o objeto ser armazenado em filas IBM MQ nos nós de origem ou de destino.

Um conjunto de programas de amostra é fornecido para demonstrar como Reference Messages pode ser colocado em uma fila, recebido por saídas de mensagens e obtido de uma fila. Os programas de amostra usam Reference Messages para mover arquivos. Se desejar mover outros objetos como bancos de dados ou se desejar executar verificações de segurança, defina sua própria saída com base na amostra amqsxrm.

A versão do programa de amostra de saída Reference Message a ser usada depende da plataforma na qual o canal está em execução:

- Em todas as plataformas, use amqsxrma na extremidade de envio.
- Use amqsxrma na extremidade de recebimento se o receptor estiver em execução sob qualquer plataforma, exceto IBM i.
- IBM i Se o receptor estiver em execução sob o IBM i, use amqsxrm4.

.

IBM i *Notas para usuários do IBM i*

Para receber uma Mensagem de Referência usando a saída de mensagem de amostra, especifique um arquivo no sistema de arquivos raiz do IFS ou de qualquer subdiretório de modo que um arquivo de fluxo possa ser criado.

A saída da mensagem de amostra no IBM i cria o arquivo, converte os dados para EBCDIC e define a página de códigos para a página de códigos de seu sistema. É possível, então, copiar esse arquivo para o sistema de arquivos QSYS.LIB usando o comando CPYFRMSTMF. Por exemplo:

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

O comando CPYFRMSTMF não cria o arquivo. Deve-se criá-lo antes de executar este comando.

Se enviar um arquivo a partir de QSYS.LIB, nenhuma mudança será necessária nas amostras. Para qualquer sistema de arquivos diferente, assegure que o CCSID especificado no campo CodedCharSetId na estrutura MQRMH corresponda os dados em massa que você está enviando.

Ao usar o sistema de arquivos integrado, crie módulos de programa com o conjunto de opções SYSIFCOPT(*IFSIO). Se desejar mover arquivos de registro do banco de dados ou de comprimento fixo, defina o sua própria saída com base na amostra AMQSXRM4 fornecida.

O método recomendado de transferência de um arquivo de banco de dados é convertê-lo para a estrutura IFS, usando o comando CPYTOSTMF e, em seguida, enviar a Reference Message anexando o arquivo IFS. Se você optar por transferir um arquivo de banco de dados referindo-se a ele a partir do IFS, mas não convertê-lo para a estrutura IFS, deve-se especificar o nome do membro. A integridade de dados não é garantida se esse método for escolhido.

Multi *Executando as amostras Reference Message*

Use este exemplo para descobrir como executar o aplicativo de amostra Reference Message AMQSPRM no AIX, Linux, and Windows ou AMQSPRMA no IBM i. O exemplo mostra como Reference Messages pode ser colocado em uma fila, recebido por saídas de mensagens e obtido de uma fila.

As amostras Reference Message são executadas da seguinte forma:

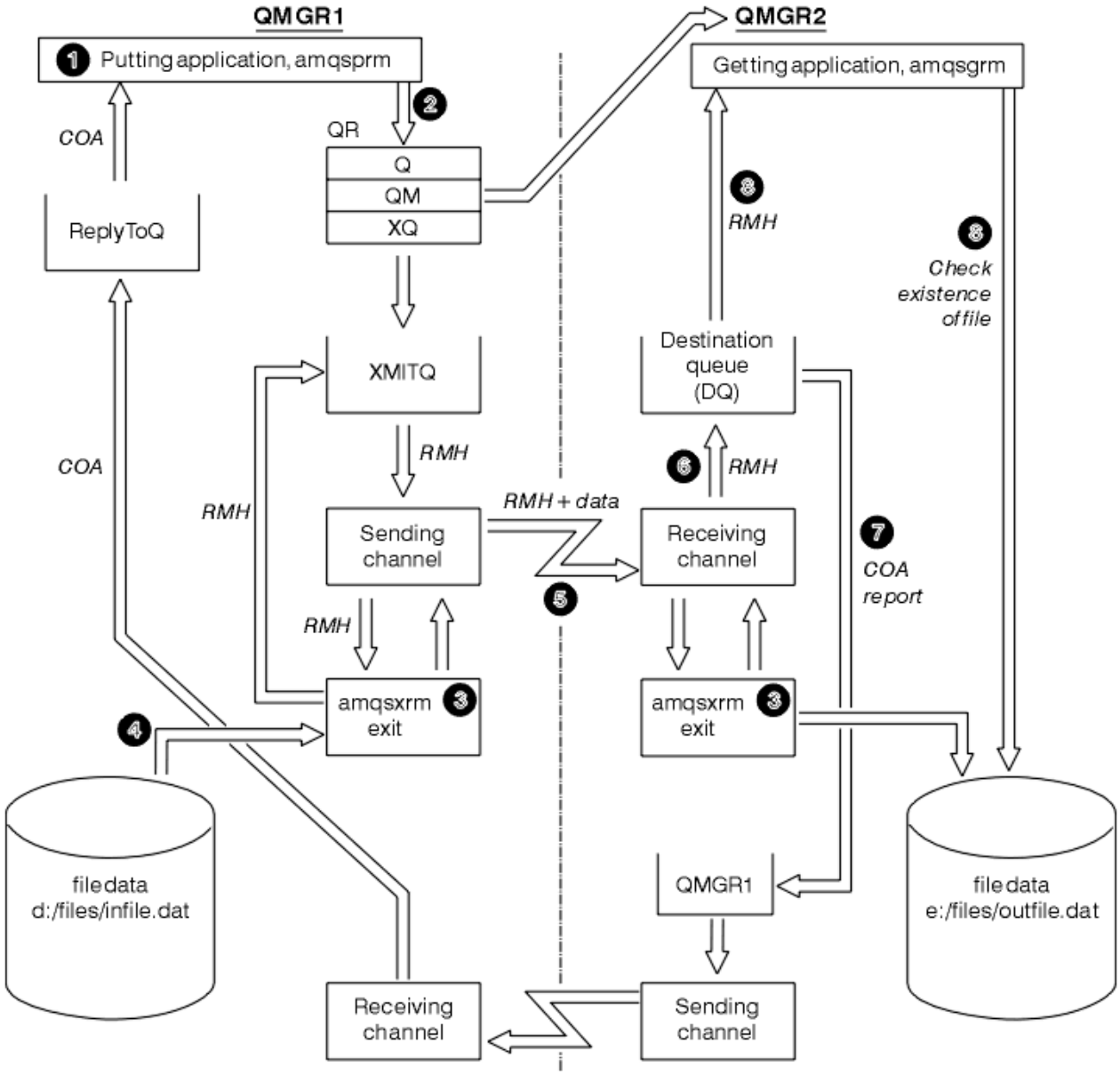


Figura 131. Executando as amostras Reference Message

1. Configure o ambiente para iniciar os listeners, canais e monitores acionadores e definir os canais e filas.

   Para os propósitos de descrever como configurar o Reference Message, este exemplo refere-se à máquina de envio como MACHINE1 com um gerenciador de filas chamado QMGR1 e a máquina de destino como MACHINE2 com um gerenciador de filas chamado QMGR2.

   Nota: As definições a seguir permitem que uma Mensagem de referência seja construída para enviar um arquivo com um tipo de objeto de FLATFILE do gerenciador de filas QMGR1 para QMGR2 e para recriar o arquivo conforme definido na chamada para AMQSPRM(ou AMQSPRMA no IBM i). A Reference Message (incluindo o arquivo de dados) é enviada usando canal CHL1 e a fila de transmissão XMITQ e é colocada na fila DQ. Relatórios de exceções e COA são enviados de volta a QMGR1 usando o canal REPORT e a fila de transmissão QMGR1.

   O aplicativo que recebe o Reference Message (AMQSGRM ou AMQSGRMA no IBM i) é acionado usando a fila de inicialização INITQ e o processo PROC. Assegure que os campos CONNAME estejam configurados corretamente e que o campo MSGEXIT reflita sua estrutura de diretórios, dependendo do tipo de máquina e onde o produto IBM MQ está instalado.

IBM i As definições do MQSC têm usado um estilo do AIX para definir as saídas, portanto, se você estiver usando MQSC no IBM i, será necessário modificá-las de forma apropriada. É importante observar que os dados da mensagem FLATFILE fazem distinção entre maiúsculas e minúsculas e a amostra não funcionará a menos que esteja em maiúsculas.

Na máquina MACHINE1, gerenciador de filas QMGR1

Sintaxe do MQSC

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **Sintaxe de comando do IBM i**

Nota: Se você não especificar um nome de gerenciador de filas, o sistema usa o gerenciador de filas padrão.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
            REPLACE(*YES) TRPTYPE(*TCP) +
            CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
            MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ   QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
            REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
            MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ   QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
            REPLACE(*YES) RMTQNAME(DQ) +
            RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

Na máquina MACHINE2, gerenciador de filas QMGR2

Sintaxe do MQSC

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i **Sintaxe de comando do IBM i**

Nota: IBM i No IBM i, se você não especificar um nome do gerenciador de filas, o sistema usa o gerenciador de filas padrão.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
```

```
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. Quando os objetos do IBM MQ tiverem sido criados:
   a. Quando aplicável para a plataforma, inicie o listener para os gerenciadores de filas de envio e de recebimento
   b. Inicie os canais CHL1 e REPORT
   c. No gerenciador de filas de recebimento, inicie o monitor acionador para a fila de inicialização INITQ
3. Chame o programa de amostra Put Reference Message AMQSPRM (AMQSPRMA no IBM i) a partir da linha de comandos usando os parâmetros a seguir:

   -m
   Nome do gerenciador de filas local; o padrão usado é o gerenciador de filas padrão

   -i
   Nome e local do arquivo de origem

   -o
   Nome e local do arquivo de destino

   -q
   Nome da fila

   -g
   O nome do gerenciador de filas no qual a fila definida no parâmetro -q existe. Isso é padronizado para o gerenciador de filas especificado no parâmetro -m.

   -t
   Tipo de objeto

   -w
   Intervalo de espera, ou seja, o tempo de espera para relatórios de exceções e COA do gerenciador de filas de recebimento

   Por exemplo, para usar a amostra com os objetos definidos anteriormente, você usaria os parâmetros a seguir:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   O aumento do tempo de espera concede tempo para que um arquivo grande seja enviado por uma rede antes que o programa que está colocando as mensagens atinja o tempo limite.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

Usuários do IBM i: IBM i No IBM i, conclua as etapas a seguir:

   a. Utilize o seguinte comando:

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

Isso supõe que o arquivo original `rmsg1` esteja no diretório `/refmsgs` do IFS e que você deseja que o arquivo de destino seja `rmsgx` no diretório `/refmsgs` do IFS no sistema de destino.

b. Crie seu próprio diretório usando o comando CRTDIR em vez de usar o diretório raiz.

c. Ao chamar o programa que coloca os dados, lembre-se de que o nome do arquivo de saída precisa refletir a convenção de nomenclatura do IFS; por exemplo, /TEST/FILENAME cria um arquivo chamado FILENAME no diretório TEST.

Nota:

IBM i No IBM i, é possível usar uma barra (/) ou um traço (-) ao especificar parâmetros. Por exemplo:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX Para as plataformas AIX and Linux, deve-se usar duas barras invertidas (\\) em vez de uma para indicar o diretório do arquivo de destino. Portanto, o comando **amqsprm** é semelhante a este:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

Executar o programa Put Reference Message faz o seguinte:

- A Reference Message é colocada na fila QR no gerenciador de filas QMGR1.
- O arquivo de origem e o caminho são `d:\files\infile.dat` e existem no sistema no qual o comando de exemplo é emitido.
- Se a fila QR for uma fila remota, a Reference Message será enviada para outro gerenciador de filas, em um sistema diferente, no qual um arquivo é criado com o nome e o caminho `e:\files\outfile.dat`. Os conteúdos desse arquivo são os mesmos que do arquivo de origem.
- amqsprm espera 30 segundos por um relatório de COA do gerenciador de filas de destino.
- O tipo de objeto é `flatfile`, portanto, o canal usado para mover as mensagens da fila QR deve especificar isso no campo *MsgData*.

4. Ao definir seus canais, selecione a saída de mensagem em ambas as extremidades de envio e de recebimento para ser amqsxrm.

Windows Isso é definido no Windows conforme a seguir:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX Isso é definido no AIX and Linux conforme a seguir:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Se você especificar um nome de caminho, especifique o nome completo. Se você omitir o nome do caminho, assume-se que o programa esteja no caminho especificado no arquivo `qm.ini` (ou, no IBM MQ for Windows, o caminho especificado no registro).

5. A saída do canal lê o da Reference Message e localiza o arquivo ao qual ele se refere.
6. A saída do canal pode então segmentar o arquivo antes de enviá-lo para o canal juntamente com o cabeçalho.

Linux AIX No AIX and Linux, altere o proprietário do grupo do diretório de destino para 'mqm' de modo que a saída da mensagem de amostra possa criar o arquivo nesse diretório. Além

disso, mude as permissões do diretório de destino para permitir que membros do grupo mqm gravem nele. Os dados do arquivo não são armazenados nas filas do IBM MQ.

7. Quando o último segmento do arquivo for processado pela saída de mensagem de recebimento, a Reference Message será colocada na fila de destino especificada por amqsprm. Se essa fila for acionada (ou seja, a definição especificar os atributos de fila **Trigger**, **InitQ** e **Process**), o programa especificado pelo parâmetro PROC da fila de destino será acionado. O programa a ser acionado deve ser definido no campo ApplId do atributo **Process**.
8. Quando a Reference Message atingir a fila de destino (DQ), um relatório de COA será enviado de volta ao aplicativo de put (amqsprm).
9. A amostra Get Reference Message, amqsgrm, obtém mensagens da fila especificada na mensagem do acionador de entrada e verifica a existência do arquivo.

Design da amostra Put Reference Message (amqsprma.c, AMQSPRM4)
Este tópico fornece uma descrição detalhada de uma amostra Put Reference Message.

Essa amostra cria uma Reference Message que refere-se a um arquivo e a coloca em uma fila especificada:

1. A amostra se conecta a um gerenciador de filas locais usando MQCONN.
2. Em seguida, abre (MQOPEN) uma fila modelo que é usada para receber mensagens de relatório.
3. A amostra constrói uma Reference Message que contém os valores necessários para mover o arquivo, por exemplo, os nomes dos arquivos de origem e de destino e o tipo de objeto. Como exemplo, a amostra enviada com o IBM MQ constrói uma Reference Message para enviar o arquivo `d:\x\file.in` de `QMGR1` para `QMGR2` e para recriar o arquivo como `d:\y\file.out` usando os parâmetros a seguir:

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   Em que `QR` é uma definição de fila remota que refere-se a uma fila de destino em `QMGR2`.

   Nota: Para as plataformas AIX and Linux, use duas barras invertidas (\\) em vez de uma para denotar o diretório do arquivo de destino. Portanto, o comando **amqsprm** é semelhante a este:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```

4. A Reference Message é colocada (sem nenhum dado de arquivo) na fila especificada pelo parâmetro /q. Se essa for uma fila remota, a mensagem será colocada na fila de transmissão correspondente.
5. A amostra espera o período de tempo especificado no parâmetro /w (que usa como padrão 15 segundos), para relatórios COA, que, juntamente com os relatórios de exceções, são enviados de volta à fila dinâmica criada no gerenciador de filas locais (QMGR1).

Design da amostra Reference Message Exit (amqsxrma.c, AMQSXRM4)
Esta amostra reconhece Mensagens de Referência com um tipo de objeto que corresponde ao tipo de objeto no campo de dados do usuário de saída de mensagem da definição do canal.

Para essas mensagens, ocorre o seguinte:

- No canal emissor ou do servidor, o comprimento especificado de dados é copiado do deslocamento especificado do arquivo especificado para o espaço restante no buffer do agente após a Reference Message. Se o término do arquivo não for atingido, a Mensagem de referência será colocada de volta na fila de transmissão após a atualização do campo *DataLogicalOffset*.
- No canal solicitante ou receptor, se o campo *DataLogicalOffset* for zero e o arquivo especificado não existir, ele será criado. Os dados que seguem a Reference Message são incluídos no final do arquivo especificado. Se a Reference Message não for a última do arquivo especificado, será descartada. Caso contrário, será retornada à saída de canal, sem os dados anexados, para ser colocada na fila de destino.

Para os canais de remetente e servidor, se o campo *DataLogicalLength* na Mensagem de referência de entrada for zero, a parte restante do arquivo, de *DataLogicalOffset* até o término do arquivo, deverá ser enviada ao longo do canal. Se não for zero, somente o comprimento especificado será enviado.

Se ocorrer um erro (por exemplo, se a amostra não puder abrir um arquivo), MQCXP. *ExitResponse* é configurado como MQXCC_SUPPRESS_FUNCTION para que a mensagem que está sendo processada seja colocada na fila de mensagens não entregues em vez de continuar na fila de destino. Um código de feedback é retornado em MQCXP. *Feedback* e retornou para o aplicativo que colocou a mensagem no campo *Feedback* do descritor de mensagens de uma mensagem de relatório. Isso porque o aplicativo de colocação solicitou relatórios de exceção configurando-se MQRO_EXCEÇÃO no campo *Report* do MQMD.

Se a codificação ou *CodedCharacterSetId* (CCSID) da Mensagem de referência for diferente da do gerenciador de filas, a Mensagem de referência será convertida para a codificação local e o CCSID. Em nossa amostra, amqsprm, o formato do objeto é MQFMT_STRING, portanto, amqsxrm converte os dados do objeto para o CCSID local na extremidade de recebimento antes que os dados sejam gravados no arquivo.

Não especifique o formato do arquivo que está sendo transferido como MQFMT_STRING se o arquivo contiver caracteres de multibyte (por exemplo, DBCS ou Unicode). Isso ocorre porque um caractere de multibyte pode ser dividido quando o arquivo for segmentado na extremidade de envio. Para transferir e converter esse arquivo, especifique o formato como algo diferente de MQFMT_STRING de forma que a saída de Reference Message não o converta e converta o arquivo na extremidade de recebimento quando a transferência for concluída.

Compilando a amostra Reference Message Exit
Para compilar a amostra de saída Reference Message, use o comando para a plataforma na qual o IBM MQ está instalado.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para compilar amqsxrma, use os seguintes comandos:

no AIX

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

no IBM i

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

Nota:

1. Para criar seu módulo para que ele use o sistema de arquivos IFS, inclua a opção `SYSIFCOPT(*IFSIO)`
2. Para criar o programa para usar com canais não encadeados, use o comando a seguir: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)`
3. Para criar o programa para usar com canais encadeados, use o comando a seguir: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)`

no Linux

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

no Windows

Windows O IBM MQ agora fornece a biblioteca mqm com pacotes do cliente, bem como pacotes do servidor, portanto, o exemplo a seguir usa `mqm.lib` em vez de `mqmvx.lib`:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Conceitos relacionados

"Gravando programas de saída do canal" na página 976
É possível usar as seguintes informações para ajudá-lo a gravar programas de saída do canal.

Design da amostra Get Reference Message (amqsgrma.c, AMQSGRM4)
Este tópico explica o design da amostra Get Reference Message.

A lógica do programa é a seguinte:

1. A amostra é acionada e extrai os nomes da fila e do gerenciador de filas da mensagem do acionador de entrada.
2. Em seguida, conecta-se ao gerenciador de filas especificado usando MQCONN e abre a fila especificada usando MQOPEN.
3. A amostra emite MQGET com um intervalo de espera de 15 segundos em um loop para obter mensagens da fila.
4. Se uma mensagem for uma Reference Message, a amostra verifica a existência do arquivo que foi transferido.
5. Em seguida, fecha a fila e desconecta do gerenciador de filas.

Os programas de amostra Request

Os programas de amostra Request demonstram o processamento do cliente/servidor. As amostras são os clientes que colocam mensagens de solicitação em uma fila do servidor de destino que é processado por um programa do servidor. Eles aguardam o programa do servidor para colocar uma mensagem de resposta em uma fila de responder para.

As amostras Request colocam uma série de mensagens de solicitação na fila do servidor de destino usando a chamada MQPUT. Essas mensagens especificam a fila local, SYSTEM.SAMPLE.REPLY, como a fila de resposta, que pode ser uma fila local ou remota. Os programas aguardam mensagens de resposta e, em seguida, exibem-nas. As respostas serão enviadas apenas se a fila do servidor de destino estiver sendo processada por um aplicativo do servidor ou se um aplicativo for acionado para esse propósito (os programas de amostra Inquire, Set e Echo são projetados para serem acionados). A amostra C aguarda 1 minuto (a amostra COBOL aguarda 5 minutos) pela chegada da primeira resposta (para permitir que o tempo para um aplicativo do servidor seja acionado) e 15 segundos por respostas subsequentes, mas ambas as amostras podem ser encerradas sem obter nenhuma resposta. Consulte "Recursos demonstrados nos programas de amostra em multiplataformas" na página 1072 para obter os nomes dos programas de amostra de Request.

Executando os programas de amostra Request

Executando as amostras amqsreq0.c, amqsreq e amqsreqc

A versão de C do programa aceita três parâmetros:

1. O nome da fila do servidor de destino (necessário)
2. O nome do gerenciador de filas (opcional)
3. A fila de resposta (opcional)

Por exemplo, insira um dos seguintes:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

em que `myqueue` é o nome da fila do servidor de destino, `qmanagername` é o nome do gerenciador de filas que possui `myqueue` e `replyqueue` é o nome da fila de resposta.

Se você omitir o nome do gerenciador de filas, supõe-se que o gerenciador de filas padrão possui a fila. Se omitir o nome da fila de resposta, a fila de resposta padrão será fornecida.

Executando a amostra amq0req0.cbl

A versão em COBOL não tem nenhum parâmetro. Ela se conecta ao gerenciador de filas padrão e quando executá-la, será solicitado o seguinte:

```
Please enter the name of the target server queue
```

O programa usa sua entrada de StdIn e inclui cada linha na fila de servidor de destino, tendo cada linha de texto como o conteúdo de uma mensagem de solicitação. O programa termina quando uma linha nula é lida.

Executando a amostra AMQSREQ4

O programa C cria mensagens obtendo dados de stdin (o teclado) com uma entrada de finalização de tempo em branco. O programa aceita até três parâmetros: o nome da fila de destino (obrigatório), o nome do gerenciador de filas (opcional) e o nome da fila de resposta (opcional). Se nenhum nome de gerenciador de filas for especificado, o gerenciador de filas padrão será usado. Se nenhuma fila de resposta for especificada, a fila SYSTEM.SAMPLE.REPLY será usada.

Aqui está um exemplo de como chamar o programa de amostra de C, especificando a fila de resposta, mas deixando o gerenciador de filas padrão:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

Nota: Lembre-se de que os nomes de filas fazem distinção entre maiúsculas e minúsculas. Todas as filas criadas pelo programa de criação de arquivo de amostra AMQSAMP4 têm nomes criados em caracteres maiúsculos.

Executando a amostra AMQ0REQ4

O programa em COBOL cria mensagens, aceitando dados do teclado. Para iniciar o programa, chame o programa e especifique o nome da fila de destino como um parâmetro. O programa aceita entrada do teclado em um buffer e cria uma mensagem de solicitação para cada linha de texto. O programa para quando você insere uma linha em branco no teclado.

Executando a amostra Request usando o acionamento

Se a amostra for usada com o acionamento e um dos programas de amostra Inquire, Set ou Echo, a linha de entrada deverá ser o nome da fila que você deseja que o programa acionado acesse.

ALW *Executando a amostra de solicitação usando o acionamento no AIX, Linux, and Windows*

No AIX, Linux, and Windows, inicie o programa monitor acionador RUNMQTRM em uma sessão e, em seguida, inicie o programa amqsreq em outra sessão.

Para executar as amostras usando acionamento:

1. Inicie o programa do monitor acionador em uma sessão RUNMQTRM (a fila de inicialização SYSTEM.SAMPLE.TRIGGER está disponível para uso).
2. Inicie o programa amqsreq em outra sessão.
3. Certifique-se de que tenha definido uma fila do servidor de destino.

   As filas de amostra disponíveis para uso como a fila do servidor de destino para a amostra de solicitação nas quais colocar mensagens são:

   - SYSTEM.SAMPLE.INQ - para o programa de amostra Inquire
   - SYSTEM.SAMPLE.SET - para o programa de amostra Set
   - SYSTEM.SAMPLE.ECHO - para o programa de amostra Echo

   Essas filas têm um tipo de acionador FIRST, portanto, se já houver mensagens nas filas antes de você executar a amostra Request, os aplicativos do servidor não serão acionados pelas mensagens que você enviar.
4. Certifique-se de que tenha definido uma fila para o programa de amostra Inquire, Set ou Echo para uso.

Isso significa que o monitor acionador está pronto quando a amostra de solicitação envia uma mensagem.

Nota: As definições de processo de amostra criadas usando RUNMQSC e o arquivo amqscos0.tst acionam as amostras de C. Mude as definições de processo em amqscos0.tst e use RUNMQSC com este arquivo atualizado para usar versões em COBOL.

Figura 132 na página 1130 demonstra como usar as amostras Request e Inquire juntas.

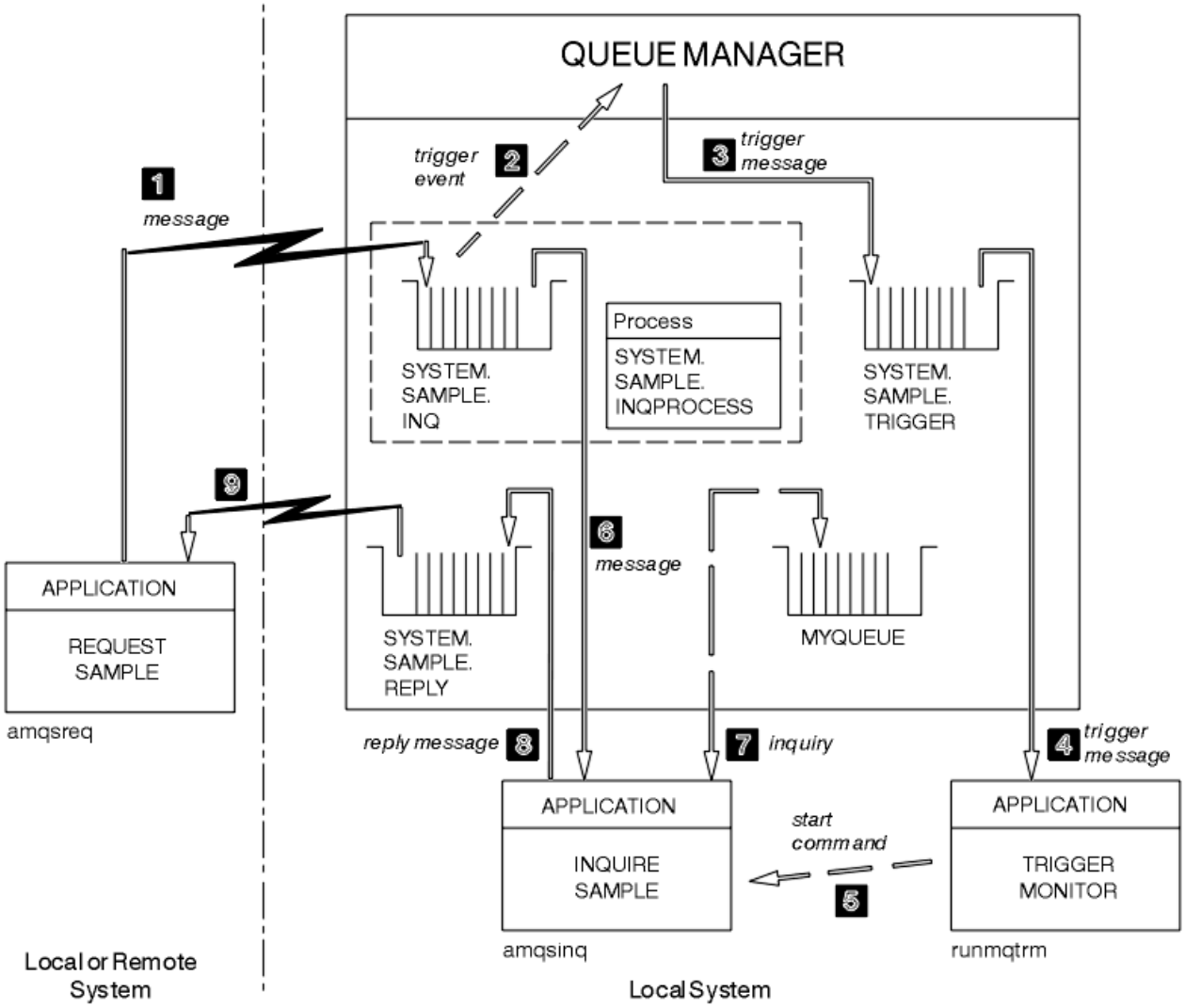


Figura 132. Amostras Request e Inquire usando acionamento

Em Figura 132 na página 1130, a amostra Request coloca mensagens na fila do servidor de destino, SYSTEM.SAMPLE.INQ e a amostra Inquire consulta a fila, MYQUEUE. Como alternativa, é possível usar uma das filas de amostra definidas quando executou amqscos0.tst ou qualquer outra fila que tenha definido para a amostra Inquire.

Nota: Os números em Figura 132 na página 1130 mostram a sequência de eventos.

Para executar as amostras Request e Inquire usando acionamento:

1. Verifique se as filas que deseja usar estão definidas. Execute amqscos0.tst para definir as filas de amostra e definir uma fila MYQUEUE.
2. Execute o comando do monitor acionador RUNMQTRM:

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. Execute a amostra Request

```
amqsreq SYSTEM.SAMPLE.INQ
```

Nota: O objeto do processo define o que deve ser acionado. Se o cliente e o servidor não estiverem em execução na mesma plataforma, qualquer processo iniciado pelo monitor acionador deverá definir

ApplType, caso contrário, o servidor usará suas definições padrão (ou seja, o tipo de aplicativo que está normalmente associado à máquina servidor) e causará uma falha.

Para obter uma lista de tipos de aplicativos, consulte ApplType.

4. Insira o nome da fila que deseja que a amostra Inquire use:

```
MYQUEUE
```

5. Insira uma linha em branco (para finalizar o programa Request).
6. A amostra Request exibirá, então, exibir uma mensagem contendo os dados do programa Inquire obtidos a partir de MYQUEUE.

É possível usar mais de uma fila; neste caso, insira os nomes das outras filas na etapa "4" na página 1131.

Para obter mais informações sobre acionamento, consulte "Iniciando aplicativos IBM MQ usando acionadores" na página 877.

IBM i *Executando a amostra de solicitação usando o acionamento no IBM i*

No IBM i, inicie o servidor acionador de amostra, AMQSERV4, em uma tarefa e, em seguida, inicie AMQSREQ4 em outra. Isso significa que o servidor do acionador está pronto quando o programa de amostra Request envia uma mensagem.

Nota:

1. As definições de amostra criadas por AMQSAMP4 acionam as versões C das amostras. Se desejar acionar as versões COBOL, mude as definições de processo SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESS e SYSTEM.SAMPLE.SETPROCESS. É possível usar o comando CHGMQMPRC (para obter detalhes, consulte Mudar processo MQ (CHGMQMPRC)) para fazer isso ou editar e executar sua própria versão do AMQSAMP4.
2. O código-fonte para AMQSERV4 é fornecido para a linguagem C apenas. No entanto, uma versão compilada (que é possível usar com as amostras COBOL) é fornecida na biblioteca QMQM.

Você poderia colocar a sua solicitação de mensagens nessas filas de servidor de amostra:

- SYSTEM.SAMPLE.ECHO (para os programas de amostra Echo)
- SYSTEM.SAMPLE.INQ (para os programas de amostra Inquire)
- SYSTEM.SAMPLE.SET (para os programas de amostra Set)

Um fluxograma do programa SYSTEM.SAMPLE.ECHO é mostrado em Figura 133 na página 1133. Usando o arquivo de dados de exemplo, o comando para emitir a solicitação do programa C para este servidor é:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

Nota: Esta fila de amostra tem um tipo de acionador de FIRST, portanto, se já houver mensagens na fila antes de executar a amostra de Solicitação, os aplicativos do servidor não serão acionados pelas mensagens que você enviar.

Se você deseja tentar exemplos adicionais, será possível tentar as variações a seguir:

- Use AMQSTRG4 (ou sua linha de comandos STRMQMTRM equivalente, para obter detalhes, consulte Start MQ Trigger Monitor (STRMQMTRM)) em vez de AMQSERV4 para enviar a tarefa, mas possíveis atrasos de envio de tarefa podem tornar menos fácil de acompanhar o que está acontecendo.
- Execute os programas de amostra SYSTEM.SAMPLE.INQUIRE e SYSTEM.SAMPLE.SET. Usando o arquivo de dados de exemplo, os comandos para emitir as solicitações do programa C para esses servidores são, respectivamente:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

Essas filas de amostra também têm um tipo de acionador de FIRST.

Design do programa de amostra Request
O programa abre a fila do servidor de destino de forma que possa colocar mensagens. Ele usa a chamada MQOPEN com a opção MQOO_OUTPUT. Se não for possível abrir a fila, o programa exibe uma mensagem de erro contendo o código de razão retornado pela chamada MQOPEN.

O programa abre, então, a fila de resposta chamada SYSTEM.SAMPLE.REPLY para que possa receber mensagens de resposta. Para isso, o programa usa a chamada MQOPEN com a opção MQOO_INPUT_EXCLUSIVE. Se não puder abrir a fila, o programa exibirá uma mensagem de erro contendo o código de razão retornado pela chamada MQOPEN.

Para cada linha de entrada, o programa então lê o texto em um buffer e usa a chamada MQPUT para criar uma mensagem de solicitação que contém o texto dessa linha. Nessa chamado, o programa usa a opção de relatório MQRO_EXCEPTION_WITH_DATA para solicitar que quaisquer mensagens de relatório enviadas sobre a mensagem de solicitação incluirão os primeiros 100 bytes dos dados da mensagem. O programa continua até que ele atinja o final da entrada ou a chamada MQPUT falhará.

O programa usa, então, a chamada MQGET para remover mensagens de resposta da fila e exibe os dados contidos nas respostas. A chamada MQGET usa as opções MQGMO_WAIT, MQGMO_CONVERT e MQGMO_ACCEPT_TRUNCATED. O *WaitInterval* é de 5 minutos na versão em COBOL e de 1 minuto na versão em C, para a primeira resposta (para conceder tempo para que um aplicativo do servidor seja acionado) e 15 segundos para as respostas subsequentes. O programa espera esses períodos se não houver nenhuma mensagem na fila. Se nenhuma mensagem chegar antes desse intervalo expirar, a chamada falhará e retornará o código de razão MQRC_NO_MSG_AVAILABLE. A chamada também usa a opção MQGMO_ACCEPT_TRUNCATED_MSG de forma que as mensagens mais longas do que o tamanho do buffer declarado sejam truncadas.

O programa demonstra como limpar os campos *MsgId* e *CorrelId* da estrutura MQMD após cada chamada MQGET porque a chamada configura esses campos para os valores contidos na mensagem que ela recupera. Desmarcar esses campos significa que sucessivas chamadas MQGET recuperam as mensagens na ordem em que elas são retidas na fila.

O programa continua até que a chamada MQGET retorne o código de razão MQRC_NO_MSG_AVAILABLE ou a chamada MQGET falhe. Se a chamada falhar, o programa exibirá uma mensagem de erro que contém o código de razão.

O programa fecha, então, a fila do servidor de destino e a fila de resposta usando a chamada MQCLOSE.

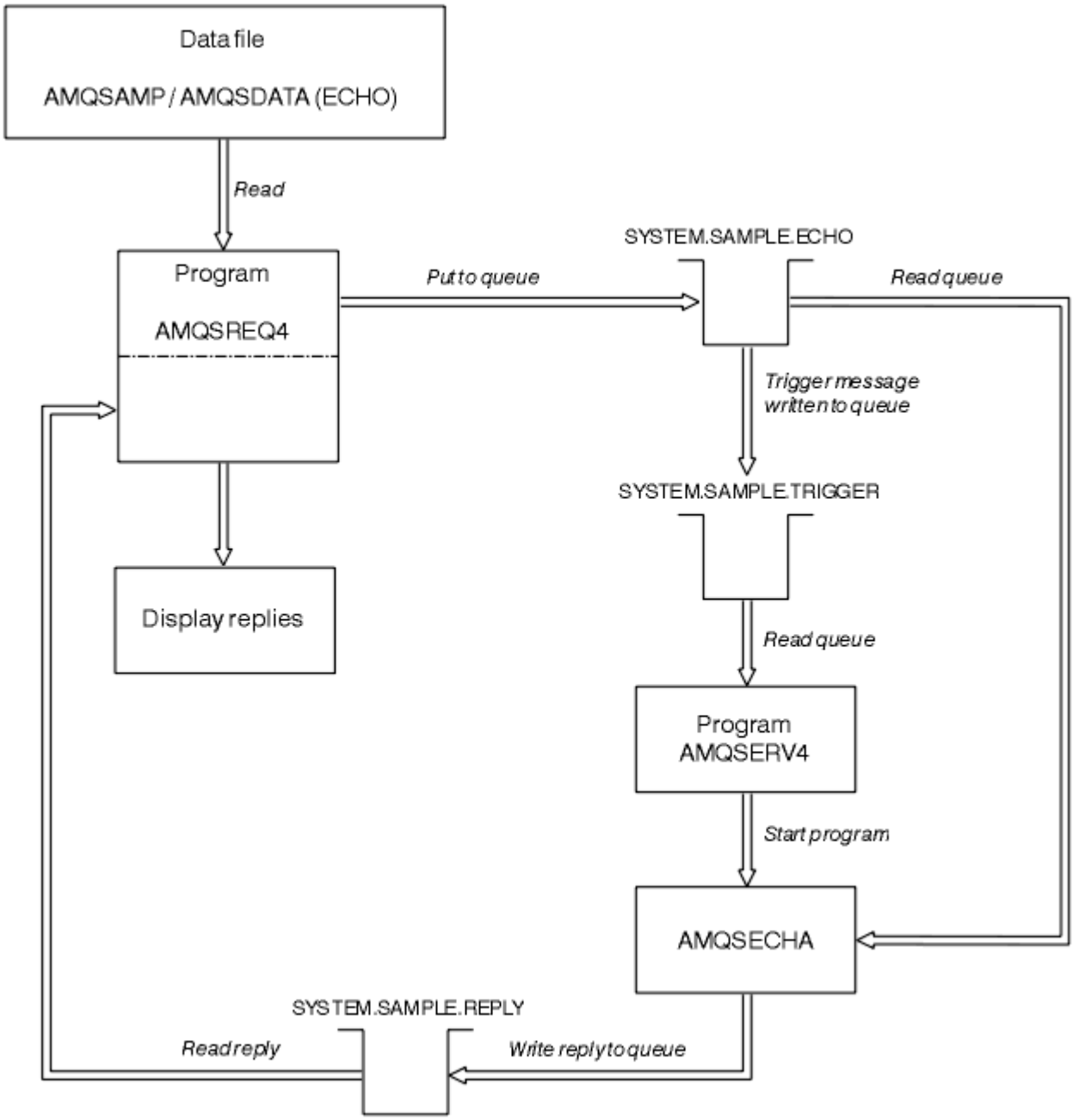


Figura 133. Fluxograma do programa IBM i Cliente/Server (Echo) de amostra

Os programas de amostra Set

Os programas de amostra Set inibem operações put em uma fila usando a chamada MQSET para mudar o atributo **InhibitPut** da fila. Além disso, aprenda sobre o design dos programas de amostra Set.

Consulte “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072 para obter os nomes desses programas.

Os programas são destinados a serem executados como programas acionados, portanto, sua única entrada é uma estrutura MQTMC2 (mensagem do acionador) que contém o nome de uma fila de destino com atributos que devem ser consultados. A versão de C também usa o nome do gerenciador de filas. A versão COBOL usa o gerenciador de filas padrão.

Para o processo de acionamento funcionar, assegure que o programa de amostra Set que deseja usar seja acionado por mensagens que chegam na fila SYSTEM.SAMPLE.SET. Para fazer isso, especifique o nome do programa de amostra Set que deseja usar no campo *ApplicId* da definição de processo

SYSTEM.SAMPLE.SETPROCESS. A fila de amostra tem um tipo de acionador FIRST; se já houver mensagens na fila antes da execução da amostra Request, a amostra Set não será acionada pelas mensagens enviadas.

Quando a definição tiver sido configurada corretamente:

- ALW Para sistemas AIX, Linux, and Windows, inicie o programa **runmqtrm** em uma sessão; em seguida, inicie o programa amqsreq em outra.
- IBM i Para o IBM i, inicie o programa AMQSERV4 em uma sessão, em seguida, inicie o programa AMQSREQ4 em outra. Seria possível usar AMQSTRG4 em vez de AMQSERV4, mas os potenciais atrasos de envio de tarefa poderiam tornar mais difícil seguir o que está acontecendo.

Use os programas de amostra Request para enviar mensagens de solicitação, cada uma contendo apenas um nome de fila, para a fila SYSTEM.SAMPLE.SET. Para cada mensagem de solicitação, os programas de amostra Set enviam uma mensagem de resposta que contém uma confirmação de que as operações put foram inibidas na fila especificada. As respostas são enviadas à fila de resposta especificada na mensagem de solicitação.

Design do programa de amostra Set

O programa abre a fila denominada na estrutura da mensagem do acionador que foi passada quando ele foi iniciado. (Por questão de clareza, chamaremos isso de *fila de solicitações*.) O programa usa a chamada MQOPEN para abrir essa fila para entrada compartilhada.

O programa usa a chamada MQGET para remover as mensagens dessa fila. Essa chamada usa as opções MQGMO_ACCEPT_TRUNCATED_MSG e MQGMO_WAIT com um intervalo de espera de 5 segundos. O programa testa o descritor de cada mensagem para ver se é uma mensagem de solicitação; se não for, o programa descarta a mensagem e exibe uma mensagem de aviso.

Para cada mensagem de solicitação removida da fila de solicitações, o programa lê o nome da fila (que chamaremos de *fila de destino*) contido nos dados e abre essa fila usando a chamada MQOPEN com a opção MQOO_SET. O programa usa, então, a chamada MQSET para configurar o valor do atributo **InhibitPut** da fila de destino para MQQA_PUT_INHIBITED.

Se a chamada MQSET for bem-sucedida, o programa usará a chamada MQPUT1 para colocar uma mensagem de resposta na fila de resposta. Essa mensagem contém a sequência `PUT inhibited`.

Se a chamada MQOPEN ou MQSET não for bem-sucedida, o programa usará a chamada MQPUT1 para colocar uma mensagem `report` na fila de resposta. No campo *Feedback* do descritor da mensagem de relatório está o código de razão retornado pela chamada MQOPEN ou pela chamada MQSET, dependendo de qual falhou.

Após a chamada MQSET, o programa fecha a fila de destino usando a chamada MQCLOSE.

Quando não houver nenhuma mensagem restante na fila de solicitações, o programa fecha essa fila e desconecta do gerenciador de filas.

O programa de amostra TLS

AMQSSSLC é um programa C de amostra que demonstra como usar as estruturas MQCNO e MQSCO para fornecer informações de conexão do cliente TLS na chamada MQCONNX. Isso permite que um aplicativo MQI cliente forneça a definição de seu canal de conexão do cliente e as configurações de TLS no tempo de execução sem uma tabela de definição de canal de cliente (CCDT).

Se um nome de conexão for fornecido, o programa constrói uma definição de canal de conexão do cliente em uma estrutura MQCD.

Se o nome de stem do arquivo de repositório de chaves for fornecido, o programa construirá uma estrutura MQSCO; se uma URL do replicador OCSP também for fornecida, o programa construirá uma estrutura MQAIR de registro de informações de autenticação.

O programa, então, conecta-se ao gerenciador de filas usando MQCONNX. Ele consulta e imprime o nome do gerenciador de filas ao qual ele está conectado.

Esse programa destina-se a ser vinculado como um aplicativo cliente de MQI. No entanto, ele pode ser vinculado como um aplicativo MQI regular, nesse caso, ele simplesmente se conecta a um gerenciador de fila local e ignora as informações de conexão do cliente

Se a passphrase para acessar o repositório de chaves não estiver armazenada em arquivo, você deverá fornecer a passphrase para **amqssslc** quando o aplicativo for executado. É possível fornecer a passphrase:

- Solicitando que **amqssslc** solicite a passphrase ou
- Usando a variável de ambiente *MQKEYRPWD* ou
- Usando o atributo **SSLKeyRepositoryPassword** no arquivo de configuração do cliente

Para obter mais informações sobre como fornecer a senha do repositório de chaves para aplicativos IBM MQ MQI client , consulte Fornecendo a senha do repositório de chaves para um IBM MQ MQI client em AIX, Linux, and Windows

amqssslc aceita os parâmetros a seguir, todos os quais são opcionais:

-m QmgrName
Nome do gerenciador de filas ao qual se conectar

-c ChannelName
Nome do canal a ser usado

-x ConnName
Nome de conexão do servidor

Parâmetros TLS:

-k KeyReposFileName
O nome do arquivo do repositório de chave Se a extensão de arquivo não for fornecida, ela será assumida como .kdb Por exemplo:

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
A sequência CipherSpec do canal TLS correspondente ao **SSLCIPH** na definição de canal SVRCONN no gerenciador de filas.

-f
Especifica que apenas algoritmos certificados por FIPS 140-2 devem ser usados.

-b VALUE1[,VALUE2...]
Especifica que apenas algoritmos em conformidade com o Conjunto B devem ser usados. Este parâmetro é uma lista separada por vírgula de um ou mais dos valores a seguir: NONE,128_BIT,192_BIT. Esses valores têm o mesmo significado que aqueles para a variável de ambiente **MQSUITEB** e a configuração **EncryptionPolicySuiteB** equivalente na sub-rotina SSL do arquivo de configuração do cliente.

-p Policy
Especifica a política de validação de certificado a ser usada. Este pode ser um dos valores a seguir:

QUALQUER
Aplique cada uma das políticas de validação de certificado suportadas pela biblioteca de soquetes seguros e aceite a sequência de certificados se alguma das políticas considerar a sequência de certificados válida. Esta configuração pode ser usada para retrocompatibilidade máxima com certificados digitais mais antigos que não estão em conformidade com os padrões de certificados modernos.

RFC5280
Aplique apenas a política de validação de certificado em conformidade com RFC 5280. Esta configuração fornece validação mais estrita do que a configuração ANY, mas rejeita alguns certificados digitais mais antigos.

O valor padrão é ANY.

-l CertLabel
O rótulo certificado a ser usado para a conexão segura.

Nota: Deve-se especificar o valor usando caracteres minúsculos.

-w
Especifica que o **amqssslc** solicita que a passphrase do repositório de chaves seja fornecida

-i
Especifica que o **amqssslc** solicita a chave inicial usada para criptografar a passphrase do repositório de chaves a ser fornecida

Especifique essa opção se um arquivo de chaves inicial foi especificado quando a passphrase do repositório de chaves foi criptografada usando o utilitário **runmqicred**

Parâmetro de revogação de certificado do OCSP:

-o URL
A URL do respondente de OCSP

Também é possível configurar uma das seguintes variáveis de ambiente para fornecer credenciais que são usadas para autenticar com o gerenciador de filas:

MQSAMP_USER_ID
Configure como o ID do usuário a ser usado para autenticação de conexão, se você desejar usar um ID do usuário e uma senha para autenticar com o gerenciador de filas. O programa solicita a senha para acompanhar o ID do usuário.

MQSAMP_TOKEN
Configure como um valor não em branco se desejar fornecer um token de autenticação para autenticar com o gerenciador de filas. O programa solicita o token de autenticação.

Executando o programa de amostra TLS
Para executar o programa de amostra TLS, deve-se configurar primeiro seu ambiente TLS. Em seguida, você executa a amostra a partir da linha de comandos, fornecendo diversos parâmetros.

Sobre esta tarefa

As instruções a seguir executam o programa de amostra usando certificados pessoais. Ao variar o comando, é possível, por exemplo, usar certificados de CA e verificar seus status usando um respondente OCSP. Consulte as instruções dentro da amostra.

Procedimento

1. Crie um gerenciador de filas com o nome QM1. Para obter mais informações, consulte crtmqm.
2. Crie um repositório de chaves para o gerenciador de filas. Para obter mais informações, consulte Configurando um repositório de chaves no AIX, Linux, and Windows.
3. Crie um repositório de chaves para o cliente. Chame-o de *clientkey.kdb*.

   Armazene a senha do repositório de chaves em um arquivo stash ao criar o repositório de chaves.
4. Crie um certificado pessoal para o gerenciador de filas. Para obter mais informações, consulte Criando um certificado pessoal autoassinado no AIX, Linux, and Windows.
5. Crie um certificado pessoal para o cliente.
6. Extraia o certificado pessoal do repositório de chaves do servidor e inclua o mesmo no repositório do cliente. Para obter mais informações, consulte Extraindo a parte pública de um certificado autoassinado de um repositório de chaves no AIX, Linux, and Windows e Incluindo um certificado de autoridade de certificação (ou a parte pública de um certificado autoassinado) em um repositório de chaves em sistemas AIX, Linux, and Windows.
7. Extraia o certificado pessoal do repositório de chaves do cliente e inclua o mesmo no repositório de chaves do servidor.
8. Crie um canal de conexão do servidor usando o comando MQSC:

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

Para obter mais informações, consulte Canal de conexão do servidor

9. Defina e inicie um listener do canal no gerenciador de filas. Para obter mais informações, consulte DEFINE LISTENER e START LISTENER.
10. Execute o programa de amostra usando o comando a seguir:

```
AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
-k "C:\Program Files\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
-o http://dummy.OCSP.responder
```

Resultados

O programa de amostra executa as ações a seguir:

1. Conecta-se a qualquer gerenciador de filas especificado ou ao gerenciador de filas padrão, usando quaisquer opções especificadas.
2. Abre o gerenciador de filas e consulta sobre seu nome.
3. Fecha o gerenciador de filas.
4. Desconecta do gerenciador de filas.

Se o programa de amostra for executado com sucesso, ele exibe uma saída semelhante ao exemplo a seguir:

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Program Files\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

Se o programa de amostra encontrar um problema, ele exibirá uma mensagem de erro apropriada, por exemplo, se você especificar uma URL respondente OCSP inválida, receberá a mensagem a seguir:

```
MQCONNX ended with reason code 2553
```

Para obter uma lista de códigos de razão, consulte conclusão da API e códigos de razão.

Os programas de amostra Triggering

A função fornecida na amostra de acionamento é um subconjunto daquele fornecido no monitor acionador no programa **runmqtrm**.

Consulte “Recursos demonstrados nos programas de amostra em multiplataformas” na página 1072 para obter os nomes desses programas.

Design da amostra de acionamento

O programa de amostra de acionamento abre a fila de inicialização usando a chamada MQOPEN com a opção MQOO_INPUT_AS_Q_DEF. Ele recebe mensagens da fila de inicialização usando a chamada MQGET com as opções MQGMO_ACCEPT_TRUNCATED_MSG e MQGMO_WAIT, especificando um intervalo de espera ilimitado. O programa limpa os campos *MsgId* e *CorrelId* antes de cada chamada MQGET para obter mensagens em sequência.

Quando recupera uma mensagem da fila de inicialização, o programa testa a mensagem verificando o seu tamanho para certificar-se de que ela seja do mesmo tamanho que uma estrutura MQTM. Se esse teste falhar, o programa exibe um aviso.

Para mensagens válidas do acionador, a amostra de acionamento copia dados destes campos: *ApplicId*, *EnvrData*, *Version* e *ApplType*. Os últimos dois desses campos são numéricos, de modo que o programa cria substituições de caracteres a serem usadas em uma estrutura MQTMC2 para sistemas IBM i, AIX, Linux, and Windows.

A amostra de acionamento emite um comando inicial para o aplicativo especificado no campo *ApplicId* da mensagem do acionador e passa uma estrutura MQTMC2 ou MQTMC (uma versão em caractere da mensagem do acionador).

- ALW Nos sistemas AIX, Linux, and Windows, o campo *EnvData* é usado como uma extensão à sequência de caracteres de comando de chamada.
- IBM i No IBM i, ele é usado como parâmetros de envio de tarefa, por exemplo, a prioridade da tarefa ou a descrição da tarefa.

Por último, o programa fecha a fila de iniciação.

Finalizando os programas de amostra de acionamento no IBM i

IBM i

Um programa de monitor de acionador pode ser encerrado pela opção 2 sysrequest (ENDRQS) ou inibindo gets da fila do acionador.

Se a fila do acionador de amostra for usada, o comando será:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

Importante: Antes de iniciar o acionamento novamente nessa fila, deve-se inserir o comando a seguir:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Executando os programas de amostra Triggering
Este tópico contém informações sobre a execução de programas de amostra Triggering.

Executando as amostras amqstrg0.c, amqstrg e amqstrgc

O programa aceita dois parâmetros:

1. O nome da fila de inicialização (necessário)
2. O nome do gerenciador de filas (opcional)

Se um gerenciador de filas não for especificado, ele se conectará ao padrão. Uma fila de inicialização de amostra terá sido definida quando amqscos0.tst foi executado; o nome dessa fila é SYSTEM.SAMPLE.TRIGGER e ela pode ser usada quando esse programa for executado.

Nota: A função nessa amostra é um subconjunto da função de acionamento integral que é fornecida no programa runmqtrm.

Executando a amostra AMQSTRG4

IBM i

Este é um monitor acionador para o ambiente do IBM i. Ele envia uma tarefa do IBM i para cada aplicativo a ser iniciado. Isso significa que há processamento adicional associado a cada mensagem do acionador.

AMQSTRG4 (em QCSRC) aceita dois parâmetros: o nome da fila de inicialização que deve atender e o nome do gerenciador de filas (opcional). AMQSAMP4 (em QCLSRC) define uma fila de inicialização de amostra, SYSTEM.SAMPLE.TRIGGER, que é possível usar ao experimentar os programas de amostra.

Usando a fila do acionador de exemplo, o comando a ser emitido é:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Como alternativa, é possível usar a CL equivalente STRMQMTRM; para obter detalhes, consulte Start MQ Trigger Monitor (STRMQMTRM).

Executando a amostra AMQSERV4

IBM i

Este é um servidor acionador para o ambiente do IBM i. Para cada mensagem do acionador, esse servidor executa o comando inicial em sua própria tarefa para iniciar o aplicativo especificado. O servidor acionador pode chamar transações do CICS.

AMQSERV4 aceita dois parâmetros: o nome da fila de inicialização que deve atender e o nome do gerenciador de filas (opcional). AMQSAMP4 define uma fila de inicialização de amostra, SYSTEM.SAMPLE.TRIGGER, que é possível usar para experimentar os programas de amostra.

Usando a fila do acionador de exemplo, o comando a ser emitido é:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Design do servidor acionador
O design do servidor acionador é semelhante ao do monitor acionador, com algumas exceções

O design do servidor acionador é semelhante ao do monitor acionador, exceto que o servidor acionador:

- Permite MQAT_CICS, assim como aplicativos MQAT_OS400.
- IBM i Chama aplicativos IBM i em sua própria tarefa (ou usa STRCICSUSR para iniciar aplicativos CICS) em vez de enviar uma tarefa do IBM i.
- Para aplicativos CICS, substitui o *EnvData*, por exemplo, para especificar a região CICS, da mensagem do acionador no comando STRCICSUSR.
- Abre a fila de inicialização para entrada compartilhada, de forma que muitos servidores acionadores possam ser executados ao mesmo tempo.

Nota: Programas iniciados por AMQSERV4 não devem usar a chamada MQDISC, pois isso para o servidor acionador. Se programas iniciados por AMQSERV4 usarem a chamada MQCONN, eles obtêm o código de razão MQRC_ALREADY_CONNECTED.

ALW *Usando as amostras TUXEDO no AIX, Linux, and Windows*

Aprenda sobre os programas de amostra Put e Get para o TUXEDO e como construir o ambiente do servidor no TUXEDO.

Antes de começar

Antes de executar essas amostras, deve-se construir o ambiente do servidor.

Sobre esta tarefa

Nota: Em toda esta seção, o caractere de barra invertida (\) é usado para dividir os comandos longos em mais de uma linha. Não insira esse caractere. Insira cada comando como uma única linha.

ALW *Construindo o ambiente do servidor*
Informações sobre como construir o ambiente do servidor para IBM MQ para plataformas diferentes.

Antes de começar

Supõe-se que você tenha um ambiente funcional do TUXEDO.

AIX *Construindo o ambiente do servidor para AIX (32 bits)*
Como construir o ambiente do servidor para IBM MQ for AIX (32 bits).

Procedimento

1. Crie um diretório (por exemplo, APPDIR) no qual o ambiente do servidor seja construído e execute todos os comandos neste diretório.
2. Exporte as variáveis de ambiente a seguir, em que TUXDIR é o diretório raiz para TUXEDO e *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. Inclua a linha a seguir no arquivo TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. Execute os comandos:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. Edite ubbstxcx.cfg e inclua detalhes do nome da máquina, diretórios ativos e o gerenciador de filas conforme necessário:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Crie o TLOGDEVICE:

```
$tmadmin -c
```

   Um prompt é exibido. Nesse prompt, insira:

```
> crdl -z /APPDIR/TLOG1
```

7. Inicie o gerenciador de filas:

```
$ strmqm
```

8. Inicie o Tuxedo:

```
$ tmboot -y
```

Como proceder a seguir

Agora é possível usar os programas doputs e dogets para colocar mensagens em uma fila e recuperá-las de uma fila.

AIX *Construindo o ambiente do servidor para AIX (64 bits)*
Como construir o ambiente do servidor para IBM MQ for AIX (64 bits).

Procedimento

1. Crie um diretório (por exemplo, APPDIR) no qual o ambiente do servidor seja construído e execute todos os comandos neste diretório.
2. Exporte as variáveis de ambiente a seguir, em que TUXDIR representa o diretório raiz do TUXEDO e *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Inclua a linha a seguir no arquivo TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. Execute os comandos:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. Edite ubbstxcx.cfg e inclua detalhes do nome da máquina, diretórios ativos e o gerenciador de filas conforme necessário:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Crie o TLOGDEVICE:

```
$tmadmin -c
```

Um prompt é exibido. Nesse prompt, insira:

```
> crdl -z /APPDIR/TLOG1
```

7. Inicie o gerenciador de filas:

```
$ strmqm
```

8. Inicie o Tuxedo:

```
$ tmboot -y
```

Como proceder a seguir

Agora é possível usar os programas doputs e dogets para colocar mensagens em uma fila e recuperá-las de uma fila.

Windows *Construindo o ambiente do servidor para Windows (32 bits)*
Construindo o ambiente do servidor para IBM MQ for Windows (32 bits).

Sobre esta tarefa

Nota: Mude os campos identificados como *VARIABLES* no seguinte, para os caminhos de diretório:

Tabela 164. Campos a serem mudados para caminhos de diretório

Campo	Caminho de diretório
MQMDIR	O caminho do diretório especificado quando o IBM MQ foi instalado, por exemplo `g:\Program Files\IBM\MQ`.
TUXDIR	O caminho do diretório especificado quando o TUXEDO foi instalado, por exemplo `f:\tuxedo`.
APPDIR	O caminho do diretório a ser utilizado para o aplicativo de amostra, por exemplo `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figura 134. Exemplo do arquivo ubbstxcn.cfg para IBM MQ for Windows

Nota: Mude o nome da máquina *MachineName* e os caminhos de diretório para corresponder à sua instalação. Mude também o nome do gerenciador de filas *MYQUEUEMANAGER* para o nome do gerenciador de filas ao qual você deseja se conectar.

O arquivo `ubbconfig` de amostra do IBM MQ for Windows está listado em Figura 134 na página 1143. Ele é fornecido como `ubbstxcn.cfg` no diretório de amostras IBM MQ.

O makefile de amostra (consulte Figura 135 na página 1144) fornecido para IBM MQ for Windows é chamado `ubbstxmn.mak` e é mantido no diretório de amostras IBM MQ.

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figura 135. makefile de amostra do TUXEDO para o IBM MQ for Windows

Para construir o ambiente do servidor e as amostras, conclua as etapas a seguir.

Procedimento

1. Crie um diretório de aplicativo no qual construir o aplicativo de amostra, por exemplo:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copie os arquivos de amostra a seguir do diretório de amostra do IBM MQ para o diretório do aplicativo:

   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`

3. Edite cada um desses arquivos para configurar os nomes e caminhos de diretórios usados em sua instalação.
4. Edite `ubbstxcn.cfg` (consulte Figura 134 na página 1143) para incluir detalhes do nome da máquina e do gerenciador de filas ao qual você deseja se conectar.
5. Inclua a linha a seguir no arquivo TUXEDO *TUXDIR*`udataobj\rm`:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   A nova entrada deve ser uma linha no arquivo.
6. Configure as variáveis de ambiente a seguir:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Crie um dispositivo TLOG para TUXEDO.

   Para fazer isso, chame `tmadmin -c` e insira o comando a seguir:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Configure o diretório atual como *APPDIR* e chame o makefile de amostra `amqstxmn.mak` como um makefile de projeto externo. Por exemplo, com o Microsoft Visual C++, emita o comando a seguir:

```
msvc amqstxmn.mak
```

   Selecione **build** para construir todos os programas de amostra.

Windows *Construindo o ambiente do servidor para Windows (64 bits)*

Como construir o ambiente do servidor para IBM MQ for Windows (64 bits).

Sobre esta tarefa

Nota: Mude os campos identificados como *VARIABLES* no seguinte, para os caminhos de diretório:

Tabela 165. Campos a serem mudados para caminhos de diretório

Campo	**Caminho de diretório**
MQMDIR	O caminho do diretório especificado quando o IBM MQ foi instalado, por exemplo `g:\Program Files\IBM\MQ`.
TUXDIR	O caminho do diretório especificado quando o TUXEDO foi instalado, por exemplo `f:\tuxedo`.
APPDIR	O caminho do diretório a ser utilizado para o aplicativo de amostra, por exemplo `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figura 136. Exemplo do arquivo ubbstxcn.cfg para IBM MQ for Windows

Nota: Mude o nome da máquina *MachineName* e os caminhos de diretório para corresponder à sua instalação. Mude também o nome do gerenciador de filas *MYQUEUEMANAGER* para o nome do gerenciador de filas ao qual você deseja se conectar.

O arquivo ubbconfig de amostra para o IBM MQ for Windows está listado em Figura 136 na página 1146. Ele é fornecido como `ubbstxcn.cfg` no diretório de amostras IBM MQ.

O makefile de amostra (consulte Figura 137 na página 1147) fornecido para o IBM MQ for Windows é chamado `ubbstxmn.mak` e é mantido no diretório de amostras IBM MQ.

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figura 137. makefile de amostra do TUXEDO para o IBM MQ for Windows

Para construir o ambiente do servidor e as amostras, conclua as etapas a seguir.

Procedimento

1. Crie um diretório de aplicativo no qual construir o aplicativo de amostra, por exemplo:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copie os arquivos de amostra a seguir do diretório de amostra do IBM MQ para o diretório do aplicativo:

   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`

3. Edite cada um desses arquivos para configurar os nomes e caminhos de diretórios usados em sua instalação.
4. Edite o `ubbstxcn.cfg` (consulte Figura 136 na página 1146) para incluir detalhes do nome da máquina e do gerenciador de filas ao qual você deseja se conectar.
5. Inclua a linha a seguir no arquivo TUXEDO *TUXDIR*`udataobj\rm`

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   A nova entrada deve ser uma linha no arquivo.
6. Configure as variáveis de ambiente a seguir:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Crie um dispositivo TLOG para TUXEDO. Para fazer isso, chame `tmadmin -c` e insira o comando:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Configure o diretório atual como *APPDIR* e chame o makefile de amostra `amqstxmn.mak` como um makefile de projeto externo. Por exemplo, com o Microsoft Visual C++, emita o comando a seguir:

```
msvc amqstxmn.mak
```

   Selecione **build** para construir todos os programas de amostra.

ALW *Programa do servidor de amostra para TUXEDO*

O programa do servidor de amostra (amqstxsx) é projetado para ser executado com os programas de amostra Put (amqstxpx.c) e Get (amqstxgx.c). O programa do servidor de amostra é executado automaticamente quando o TUXEDO é iniciado.

Nota: Deve-se iniciar o gerenciador de filas antes de iniciar o TUXEDO.

O servidor de amostra fornece dois serviços do TUXEDO, MPUT1 e MGET1:

- O serviço MPUT1 é conduzido pela amostra PUT e usa MQPUT1 no ponto de sincronização para colocar uma mensagem em uma unidade de trabalho controlada pelo TUXEDO. Aceita os parâmetros QName e Message Text, que são fornecidos pela amostra PUT.
- O serviço MGET1 abre e fecha a fila toda vez que obtém uma mensagem. Aceita os parâmetros QName e Message Text, que são fornecidos pela amostra GET.

Quaisquer mensagens de erro, códigos de razão e mensagens de status são gravados no arquivo de log do TUXEDO.

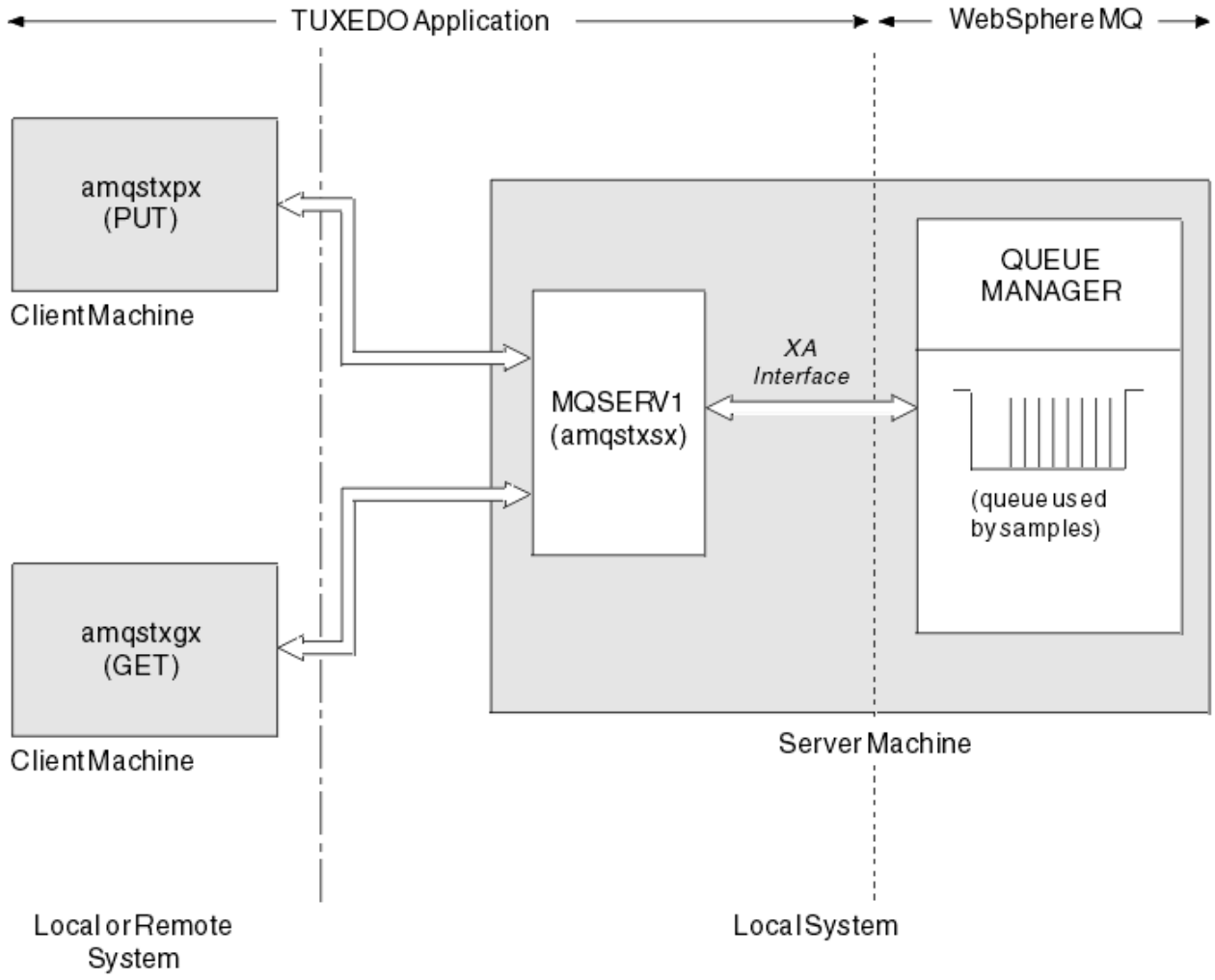


Figura 138. Como as amostras do TUXEDO funcionam juntas

ALW *Programa de amostra Put para TUXEDO*

Esta amostra permite colocar uma mensagem em uma fila várias vezes, em lotes, demonstrando a indicação de sincronização usando o TUXEDO como o gerenciador de recursos.

O programa do servidor de amostra `amqstxsx` deve estar em execução para a amostra put ter sucesso; o programa de amostra do servidor se conecta ao gerenciador de filas e usa a interface XA. Para executar a amostra, insira:

- `doputs -n queuename -b batchsize -c trancount -t message`

Por exemplo:

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

Isso coloca 30 mensagens na fila denominada `myqueue`, em seis lotes, cada um com cinco mensagens. Se houver algum problema, ele faz um lote de mensagens de saída, caso contrário, ele os confirma.

Quaisquer mensagens de erro serão gravadas no arquivo de log TUXEDO e no stderr. Quaisquer códigos de razão serão gravados no stderr.

ALW *Amostra Get para TUXEDO*

Esta amostra permite obter mensagens de uma fila em lotes.

O programa do servidor de amostra `amqstxsx` deve estar em execução para que a amostra Get seja bem-sucedida; o programa do servidor de amostra conecta-se ao gerenciador de filas e usa a interface XA. Para executar a amostra, insira o seguinte comando:

- `dogets -n queuename -b batchsize -c trancount`

Por exemplo:

- `dogets -n myqueue -b 6 -c 4`

Isso tira 24 mensagens da fila denominada `myqueue` em seis lotes, cada um com quatro mensagens. Se isso for executado após o exemplo de colocação, que coloca 30 mensagens na `myqueue`, haverá apenas seis mensagens em `myqueue`. O número de lotes e o tamanho do lote podem variar entre a colocação das mensagens e a obtenção delas.

Quaisquer mensagens de erro serão gravadas no arquivo de log TUXEDO e no stderr. Quaisquer códigos de razão serão gravados no stderr.

Windows *Usando a saída de segurança SSPI no Windows*

Este tópico descreve como usar os programas de saída do canal SSPI em sistemas Windows. O código de saída fornecido está em dois formatos: objeto e origem.

Código de objeto

O arquivo de código do objeto é chamado `amqrspin.dll`. Para o cliente e o servidor, ele é instalado como uma parte padrão de IBM MQ for Windows na pasta *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME*. Por exemplo, `C:\Program Files\IBM\MQ\exits\installation2`. É carregado como uma saída de usuário padrão. É possível executar a saída do canal de segurança fornecida e usar os serviços de autenticação em sua definição do canal.

Para fazer isso, especifique um dos seguintes:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

Para fornecer suporte a um canal restrito, especifique o seguinte no canal SVRCONN:

```
SCYDATA('remote_principal_name')
```

em que *remote_principal_name* está no formato *DOMAIN**user*. O canal seguro é estabelecido somente se o nome do principal remoto corresponder a *remote_principal_name*.

Para usar os programas de saída de canal fornecidos entre os sistemas que operam dentro de um domínio de segurança Kerberos, crie um **servicePrincipalName** para o gerenciador de filas.

Código-fonte

O arquivo de código-fonte de saída é chamado `amqsspin.c`. Está em `C:\Program Files\IBM\MQ\Tools\c\Samples`.

Se você modificar o código-fonte, deverá recompilar a origem modificada.

É possível compilar e vincular a ele da mesma maneira que qualquer outro canal de saída para a plataforma relevante, exceto que os cabeçalhos SSPI precisam ser acessados no tempo de compilação e as bibliotecas de segurança SSPI, juntamente com quaisquer bibliotecas associadas recomendadas, precisam ser acessadas no tempo do link.

Antes de executar o comando a seguir, certifique-se de que `cl.exe`, e a biblioteca Visual C++ e a pasta `include` estejam disponíveis em seu caminho. Por exemplo:

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

Nota: O código-fonte não inclui nenhuma provisão para rastreio ou manipulação de erros. Se você modificar e usar o código-fonte, inclua suas próprias rotinas de rastreio e de manipulação de erro.

Executando as amostras usando filas remotas

É possível demonstrar o enfileiramento remoto ao executar as amostras em gerenciadores de filas conectadas.

O programa amqscos0.tst fornece uma definição local de uma fila remota (SYSTEM.SAMPLE.REMOTE) que usa um gerenciador de filas remotas denominado OTHER. Para usar essa definição de amostra, mude OTHER para o nome do segundo gerenciador de filas que você deseja usar. Deve-se também configurar um canal de mensagens entre os dois gerenciadores de filas; para obter informações sobre como fazer isso, consulte Definindo os canais.

Os programas de amostra Request colocam seu próprio nome do gerenciador de filas locais no campo *ReplyToQMgr* de mensagens que eles enviam. As amostras Inquire e Set enviam mensagens de resposta para a fila e para o gerenciador de filas de mensagens nomeado nos campos *ReplyToQ* e *ReplyToQMgr* das mensagens de solicitação que processam.

O programa de amostra de Cluster Queue Monitoring (AMQSCLM)

Essa amostra usa os recursos de balanceamento de carga de trabalho do cluster IBM MQ integrados para direcionar mensagens para instâncias de filas que têm aplicativos consumidores conectados. Esta direcionamento automático evita o acúmulo de mensagens em uma instância de uma fila de clusters para os quais nenhum aplicativo de consumo está conectado.

Visão Geral

É possível configurar um cluster que possui mais de uma definição para a mesma fila em gerenciadores de filas diferentes. Esta configuração proporciona o benefício de maior disponibilidade e balanceamento de carga de trabalho. No entanto, não há recurso integrado em IBM MQ para modificar dinamicamente a distribuição de mensagens em um cluster com base no estado de aplicativos anexados. Por essa razão, um aplicativo consumidor deve estar sempre conectado a cada instância de uma fila para assegurar que as mensagens são processadas.

O programa de amostra de monitoramento da fila de clusters monitora o estado de aplicativos anexados. O programa ajusta dinamicamente a configuração de balanceamento de carga de trabalho integrada para direcionar mensagens para instâncias de uma fila em cluster com aplicativos consumidores anexados.

Em determinadas situações este programa pode ser usado para diminuir a necessidade de um aplicativo consumidor estar sempre conectado a todas as ocorrências de uma fila. Também reenvia mensagens que se tornam enfileiradas em uma instância de uma fila sem aplicativos consumidores anexados. re-envio de mensagens permite que as mensagens sejam roteadas ao redor de um aplicativo de consumo que está temporariamente encerrado.

O programa é projetado para ser usado quando os aplicativos consumidores são de longa execução, em vez de aplicativos que conectam e desconectam frequentemente.

O programa de amostra de monitoramento da fila de clusters é o programa executável compilado do arquivo de amostra C `amqsclma.c`.

Informações adicionais sobre clusters e carga de trabalho podem ser localizadas em Usando clusters para gerenciamento de carga de trabalho

AMQSCLM: projetando e planejando o uso da amostra
Informações sobre como o programa de amostra de monitoramento de fila de clusters funciona, pontos a serem considerados ao configurar um sistema para execução do programa de amostra e modificações que podem ser feitas no código-fonte de amostra.

Design

O programa de amostra de monitoramento da fila de clusters monitora filas locais em cluster que têm aplicativos de consumo conectados. O programa monitora as filas especificadas pelo usuário. O nome da fila pode ser específico, por exemplo `APP.TEST01` ou genérico. Nomes genéricos devem estar em um formato que esteja de acordo com o PCF (Formato de comando programável). Exemplos de nomes genéricos são `APP.TEST*` ou `APP*`.

Cada gerenciador de filas em um cluster que tem uma instância de uma fila local a ser monitorada requer que uma instância do programa de amostra de monitoramento de fila de clusters esteja conectada a ele.

Roteamento de mensagem dinâmico

O programa de amostra de monitoramento da fila de clusters usa o valor de **IPPROCS** (aberto para contagem de processo de entrada) de uma fila para determinar se essa fila tem algum consumidor. Um valor maior que 0 indica que a fila tem pelo menos um aplicativo de consumo conectado. Essas filas estão ativas. Um valor igual a 0 indica que a fila não tem programas de consumo conectados. Essas filas estão inativas.

Para uma fila em cluster com várias instâncias em um cluster, o IBM MQ usa a propriedade de prioridade de carga de trabalho do cluster **CLWLPRTY** de cada instância da fila para determinar para quais instâncias enviar mensagens. O IBM MQ envia mensagens para as instâncias disponíveis de uma fila com o valor de **CLWLPRTY** mais alto.

O programa de amostra do monitoramento da fila de clusters ativa uma fila de clusters ao configurar o valor do **CLWLPRTY** local como 1. O programa desativa uma fila de clusters ao configurar seu valor **CLWLPRTY** como 0.

A tecnologia de armazenamento em cluster do IBM MQ propaga a propriedade **CLWLPRTY** atualizada de uma fila em cluster para todos os gerenciadores de filas relevantes no cluster. Por exemplo,

- Um gerenciador de filas com um aplicativo conectado que coloca mensagens na fila.
- Um gerenciador de filas que tem uma fila local com o mesmo nome no mesmo cluster.

A propagação é feita usando os gerenciadores de filas de repositório completo do cluster. Novas mensagens para a fila de clusters são direcionadas para as instâncias com o valor de **CLWLPRTY** mais alto no cluster.

Transferência de mensagem enfileirada

A modificação dinâmica do valor de **CLWLPRTY** influencia o roteamento de novas mensagens. Essa modificação dinâmica não afeta as mensagens já enfileiradas em uma instância da fila sem consumidores

conectados ou mensagens que passaram pelo mecanismo de balanceamento de carga de trabalho antes de um valor modificado de **CLWLPRTY** ter sido propagado pelo cluster. Como resultado, as mensagens permanecem em qualquer fila inativa e não serão processadas por um aplicativo de consumo. Para resolver isso, o programa de amostra de monitoramento da fila de clusters é capaz de obter mensagens de uma fila local sem nenhum consumidor e enviar essas mensagens para instâncias remotas da mesma fila nas quais há consumidores conectados.

O programa de amostra de monitoramento da fila de clusters transfere mensagens de uma fila local inativo para uma ou mais filas remotas ativas que estão obtendo mensagens (usando **MQGET**) e colocando mensagens (usando **MQPUT**) na mesma fila em cluster. Essa transferência faz com que o gerenciamento de carga de trabalho de cluster do IBM MQ selecione uma instância de destino diferente, com base em um valor mais alto de CLWLPRTY do que da instância da fila local. Persistência de mensagem e contexto são preservados durante a transferência de mensagem. Ordem de mensagens e quaisquer opções vinculantes não são preservadas.

Planejando

O programa de amostra de monitoramento da fila de clusters modifica a configuração de cluster quando há uma mudança na conectividade de aplicativos de consumo. Modificações são transmitidas dos gerenciadores de filas onde o programa de amostra de monitoramento da fila de clusters está monitorando as filas para os gerenciadores de filas de repositório completo no cluster. Os gerenciadores de filas de repositório completo processam as atualizações da configuração e reenviam as mesmas a todos os gerenciadores de filas relevantes no cluster. Gerenciadores de filas relevantes incluem os gerenciadores de filas que têm filas em cluster com o mesmo nome (em que uma instância do programa de amostra de monitoramento de fila de clusters está em execução) e qualquer gerenciador de filas em que um aplicativo abriu a fila de clusters para colocar mensagens na mesma nos últimos 30 dias.

As mudanças são processadas de forma assíncrona em todo o cluster. Portanto, após cada mudança, diferentes gerenciadores de filas no cluster podem ter diferentes visualizações da configuração por um período de tempo.

O programa de amostra de monitoramento da fila de clusters é adequado somente para sistemas nos quais os aplicativos de consumo conectam ou desconectam com pouca frequência; por exemplo, aplicativos de consumo de longa execução. Quando usado para monitorar sistemas nos quais aplicativos de consumo estão conectados somente por períodos curtos, a latência incorrida ao distribuir as atualizações de configuração pode resultar em gerenciadores de filas no cluster terem uma visualização incorreta das filas às quais consumidores estão conectados. Essa latência pode resultar em mensagens roteadas incorretamente.

Ao monitorar muitas filas, uma taxa relativamente baixa de mudança dos consumidores conectados entre todas as filas pode aumentar o tráfego de configuração de cluster por todo o cluster. O aumento do tráfego de configuração de cluster pode resultar em carga excessiva em um ou mais dos gerenciadores de filas a seguir.

- Os gerenciadores de filas em que o programa de amostra de monitoramento de fila de cluster está em execução
- Os gerenciadores de filas de repositório completo
- Um gerenciador de filas com um aplicativo conectado que coloca mensagens na fila
- Um gerenciador de filas que tem uma fila local com o mesmo nome no mesmo cluster

O uso do processador nos gerenciadores de filas de repositório completo deve ser avaliado. O uso do processador adicional é visível como tráfego de mensagens na fila do repositório completo SYSTEM.CLUSTER.COMMAND.QUEUE. Se mensagens se acumularem nessa fila, isso indica que os gerenciadores de filas do repositório completo não são capazes de acompanhar o ritmo de mudança da configuração de cluster no sistema.

Quando várias filas estão sendo monitoradas pelo programa de amostra de monitoramento de fila de clusters, há uma quantia de trabalho executada pelo programa de amostra e pelo gerenciador de filas. Esse trabalho é executado mesmo quando não há mudanças nos consumidores conectados. O argumento

-i pode ser modificado para reduzir o uso do processador do programa de amostra no sistema local, reduzindo a frequência do ciclo de monitoramento.

Para ajudar a detectar atividade excessiva, o programa de amostra de monitoramento de fila de clusters relata o tempo médio de processamento por intervalo de pesquisa, o tempo de processamento decorrido e o número de mudanças na configuração. Os relatórios são entregues em uma mensagem informativa, **CLM0045I**, a cada 30 minutos ou a cada 600 intervalos de pesquisa, o que ocorrer primeiro.

Requisitos de uso de monitoramento da fila de clusters

O programa de amostra de monitoramento de fila de clusters tem requisitos e restrições. É possível modificar o código-fonte de amostra fornecido para mudar algumas dessas restrições em como pode ser usado. Os exemplos listados nesta seção detalham as modificações que podem ser feitas.

- O programa de amostra de monitoramento de fila de clusters é projetado para ser usado para monitorar filas quando aplicativos de consumo estão conectados ou não conectados. Se o sistema tiver aplicativos de consumo que estão frequentemente conectando e desconectando, o programa de amostra poderá gerar atividade excessiva de configuração de cluster em todo o cluster. Isso pode ter um impacto no desempenho dos gerenciadores de filas no cluster.
- O programa de amostra de monitoramento de fila de clusters depende do sistema IBM MQ subjacente e da tecnologia de cluster. O número de filas que estão sendo monitoradas, a frequência de monitoramento e a frequência da mudança do estado de cada fila afeta o carregamento no sistema geral. Esses fatores devem ser considerados ao selecionar as filas a serem monitoradas e o intervalo de pesquisa do monitoramento.
- Uma instância do programa de amostra de monitoramento de fila de clusters deve ser conectada a cada gerenciador de filas do cluster que tem uma instância de uma fila a ser monitorada. Não é necessário conectar o programa de amostra a gerenciadores de filas no cluster que não têm as filas.
- O programa de amostra de monitoramento de fila de clusters deve ser executado com autorização adequada para acessar todos os recursos necessários do IBM MQ. Por exemplo,
  - O gerenciador de filas ao qual ser conectado
  - O SYSTEM.ADMIN.COMMAND.QUEUE
  - Todas as filas a serem monitoradas quando a transferência de mensagem é executada
- O servidor de comandos deve estar em execução para cada gerenciador de filas com o programa de amostra de monitoramento de fila de clusters conectado.
- Cada instância do programa de amostra de monitoramento de fila de cluster requer uso exclusivo de uma fila local (sem cluster) no gerenciador de filas ao qual está conectado. Essa fila local é usada para controlar o programa de amostra e receber mensagens de resposta de consultas feitas ao servidor de comandos do gerenciador de filas.
- Todas as filas a serem monitoradas por uma única instância do programa de amostra de monitoramento de fila de clusters devem estar no mesmo cluster. Se um gerenciador de filas tiver filas em vários clusters que requerem monitoramento, várias instâncias do programa de amostra serão necessárias. Cada instância precisa de uma fila local para mensagens de controle e de resposta.
- Todas as filas a serem monitoradas devem estar em um único cluster. Filas configuradas para usar uma lista de nomes de cluster não são monitoradas.
- Ativar a transferência de mensagens de filas inativas é opcional. Ela se aplica a todas as filas que estão sendo monitoradas pela instância do programa de amostra de monitoramento de fila de clusters. Se somente um subconjunto das filas que estão sendo monitoradas requerem a transferência de mensagem ativada, duas instâncias do programa de amostra de monitoramento de fila de clusters são necessárias. Um programa de amostra tem transferência de mensagem ativada e o outro tem transferência de mensagem desativada. Cada instância do programa de amostra precisa de uma fila local para mensagens de controle e de resposta.
- O balanceamento de carga de trabalho de cluster do IBM MQ irá, por padrão, enviar mensagens para instâncias de filas em cluster que residem no mesmo gerenciador de filas ao qual um aplicativo de put está conectado. Deve ser desativado enquanto a fila local estiver inativa nas circunstâncias a seguir:

  - Os aplicativos de put se conectam a gerenciadores de filas que têm instâncias de uma fila inativa que estão sendo monitoradas
  - Mensagens enfileiradas estão sendo transferidas de filas inativas para filas ativas.

  A preferência de balanceamento de carga de trabalho local na fila pode ser desativada estaticamente por meio da configuração do valor de CLWLUSEQ para ANY. Nesta configuração, as mensagens colocadas em filas locais são distribuídas para instâncias de filas locais e remotas para balancear a carga de trabalho, mesmo quando houver aplicativos de consumo locais. Como alternativa, o programa de amostra de monitoramento de fila de clusters pode ser configurado para definir temporariamente o valor de **CLWLUSEQ** para ANY enquanto a fila não tiver consumidores conectados, o que resulta em somente mensagens locais indo para instâncias locais de uma fila enquanto essa fila estiver ativa.
- O sistema e os aplicativos IBM MQ não devem usar **CLWLPRTY** para as filas a serem monitoradas ou canais que estão sendo usados. Caso contrário, as ações do programa de amostra de monitoramento de fila de clusters nos atributos da fila **CLWLPRTY** podem ter efeitos indesejados.
- O programa de amostra de monitoramento de fila de clusters registra informações de tempo de execução em um conjunto de arquivos de relatório. Um diretório para armazenar esses relatórios é necessário e o programa de amostra de monitoramento de fila de clusters deve ter autorização para gravar nele.

AMQSCLM: preparando e executando a amostra
A amostra de monitoramento de fila de clusters pode ser executada localmente conectada a um gerenciador de filas ou como um cliente conectado por um canal. A amostra deve estar em execução sempre que o gerenciador de filas estiver em execução; quando em execução localmente, ela poderá ser configurada como um serviço do gerenciador de filas para iniciar e parar automaticamente a amostra com o gerenciador de filas.

Antes de começar

As etapas a seguir devem ser concluídas antes de executar a amostra de monitoramento da fila de clusters.

1. Crie uma fila de trabalho em cada gerenciador de filas para o uso interno da amostra.

   Cada instância da amostra precisa de uma fila não cluster local para uso interno exclusivo. É possível escolher o nome da fila. O exemplo usa o nome AMQSCLM.CONTROL.QUEUE. Por exemplo, no Windows, é possível criar essa fila usando o comando **MQSC** a seguir:

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   Você pode deixar os valores de **MAXDEPTH** e **MAXMSGL** como padrão.
2. Crie um diretório para logs de mensagens de erro e informações.

   A amostra grava mensagens de diagnóstico para arquivos de relatório. Deve-se escolher um diretório no qual armazenar os arquivos. Por exemplo, no Windows, é possível criar um diretório usando o comando a seguir:

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   Os arquivos de relatório criados pela amostra têm a convenção de nomenclatura a seguir:

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```
3. (Opcional) Defina a amostra de monitoramento da fila de cluster como um serviço do IBM MQ.

   Para monitorar filas, a amostra deve sempre estar em execução. Para assegurar que a amostra de monitoramento de fila de clusters sempre esteja em execução, é possível definir a amostra como um serviço de gerenciador de filas. Definir a amostra como um serviço significa que AMQSCLM é iniciado

quando o gerenciador de filas é iniciado. É possível usar o exemplo a seguir para definir a amostra de monitoramento da fila de cluster como um serviço do IBM MQ.

```
define service(AMQSCLM) +
      descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
      control(qmgr) +
      servtype(server) +
      startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
      startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
c:\AMQSCLM\rpts') +
      stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
      stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
```

Definição	Descrição
service	Especifica o nome do serviço. É possível escolher o nome do serviço.
descr	Especifica uma descrição textual do serviço.
control	Indica que o serviço é iniciado e parado ao mesmo tempo que o gerenciador de filas.
servtype	Indica um objeto de serviço do servidor, significando que somente uma instância pode ser executada ao mesmo tempo para este gerenciador de filas.
startcmd	Especifica o local e o nome do programa.
startarg	Especifica os argumentos da amostra. Observe o uso do *+QMNOME+*. O nome do gerenciador de filas é substituído automaticamente.
stdout	O nome completo do arquivo para o qual a saída padrão é redirecionada. A amostra grava nesse arquivo apenas mensagens confirmando que a amostra foi encerrada. A de amostra faz isso porque o arquivo de erro padrão já foi fechado em um estágio anterior do processo de finalização de amostra.
stderr	O nome do arquivo completo para o qual a saída de erro padrão é redirecionada. A amostra grava no arquivo de erro padrão quaisquer mensagens de erro anteriores ao término da amostra.

Sobre esta tarefa

Esta tarefa permite iniciar e parar a amostra de monitoramento da fila de clusters de diferentes maneiras. Ela também permite executar a amostra em um modo que gera arquivos de relatório que contêm informações estatísticas sobre as filas que estão sendo monitoradas.

O programa de amostra pode ser executado usando o comando a seguir.

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

A tabela lista os argumentos que podem ser usados com a amostra de monitoramento de fila de clusters, juntamente com informações adicionais sobre cada um.

Argumento	Variável	Informações adicionais
-m	**QMgrName**	O gerenciador de filas a monitorar.
-c	**ClusterName**	O cluster que contém as filas a monitorar.
-q	**QNameMask**	A fila, ou filas, a monitorar. Um * à direita monitora todas as filas com nomes que correspondem a zero ou mais caracteres à direita.
-f	**QListFile**	O caminho completo e o nome do arquivo de um arquivo que contém uma lista de nomes de filas ou máscaras do nome da fila para monitorar. O arquivo deve conter um nome de fila/máscara por linha. É possível especificar **-q** ou **-f**, mas não ambos.
-r	**MonitorQName**	A fila local que está sendo usada exclusivamente pela amostra.
-l	**ReportDir**	O caminho do diretório no qual armazenar mensagens de informações registradas em um conjunto de agrupamento[9]arquivos de relatório.
-t		(Opcional) Ativa a transferência de mensagens enfileiradas de filas locais inativas para as filas ativas. Se não estiver ativada, somente novas mensagens que entram no cluster são roteadas dinamicamente para instâncias ativas de uma fila.
-u	**ActiveVal**	(Opcional) Alterna automaticamente a propriedade **CLWLUSEQ** de uma instância de fila monitorada para ANY quando está inativa e para o valor de **ActiveVal** quando ativa. **ActiveVal** pode ser LOCAL ou QMGR. Se esse argumento não for definido em um sistema no qual os aplicativos put se conectam ao mesmo gerenciador de filas ou onde a transferência de mensagem estiver ativada, as filas monitoradas deverão ter um valor **CLWLUSEQ** de ANY ou QMGR com o gerenciador de filas que tem um valor de ANY.
-i	**Interval**	(Opcional) O intervalo de tempo em segundos, no qual o monitor verifica as filas. O padrão é 300 segundos (5 minutos).
-d		(Opcional) Ativa a saída de diagnóstico adicional. A saída de depuração pode ser útil ao configurar inicialmente o sistema ou ao trabalhar com o código de amostra.
-s		(Opcional) Ativa a saída estatística mínima por intervalo.
-v		(Opcional) Registrar informações do relatório para `standard out`, além dos arquivos de relatório.

Exemplos da lista de argumento:

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

Arquivo de lista de filas de exemplo:

```
Q1
QUEUE.*
```

[9] Para cada gerenciador de filas e combinação de filas, um arquivo de log de tamanho fixo é gerado que, quando cheio, é sobrescrito. O criador de logs sempre grava no mesmo arquivo e também mantém as duas versões anteriores do arquivo.

```
ABC
ABD
```

Procedimento

1. Inicie a amostra de monitoramento de fila de clusters. É possível iniciar a amostra de uma das maneiras a seguir:
   - Use um prompt de comandos com as autorizações do usuário apropriadas.
   - Use o comando MQSC **START SERVICE**, se a amostra estiver configurada como um serviço do IBM MQ.

   A lista de argumentos é a mesma em ambos os casos.

   A amostra não começa a monitorar as filas por 10 segundos depois que o programa for inicializado. Esse atraso permite que os aplicativos de consumo se conectem às filas monitoradas primeiro, impedindo mudanças desnecessárias no estado ativo da fila.
2. Pare a amostra de monitoramento de fila de clusters. A amostra é interrompida automaticamente quando o gerenciador de filas é interrompido, está parando, em quiesce ou se a conexão com o gerenciador de filas é interrompida. Há maneiras de parar a amostra sem encerrar o gerenciador de filas:
   - Configure a fila local usada exclusivamente pela amostra para desativar a função Get.
   - Envie uma mensagem com um **CorrelId** de `"STOP CLUSTER MONITOR\0\0\0\0"`, para a fila local usada exclusivamente pela amostra.
   - Finalize o processo de amostra. Isso pode resultar na perda de mensagens não persistentes sendo transferidas para filas ativas. Também pode resultar na fila local usada pela amostra sendo mantida aberta por um número de segundos após o encerramento. Esta situação impede que uma nova instância da amostra de monitoramento da fila de clusters seja iniciada imediatamente.

   Se a amostra foi iniciada como um serviço do IBM MQ, **STOP SERVICE** não terá efeito. É possível usar um dos métodos de finalização descrito como um mecanismo **STOP SERVICE** configurado no gerenciador de filas.

Como proceder a seguir

Verifique o status da amostra.

Se o relatório estiver ativado, é possível revisar o status dos arquivos de relatório. Use o comando a seguir para revisar o arquivo de relatório mais atual:

```
QMgrName.ClusterName.RPT01.LOG
```

Para revisar os arquivos de relatório mais antigos, use os comandos a seguir:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

Os arquivos de relatório aumentam até o tamanho máximo de aproximadamente 1 MB. Quando o arquivo RPT01 se torna cheio, um novo arquivo RPT01 é criado. O antigo arquivo RPT01 é renomeado para RPT02. RPT02 é renomeado para RPT03. O RPT03 antigo é descartado.

A amostra cria mensagens de informações nas situações a seguir:

- na inicialização
- na finalização
- ao marcar uma fila como **ACTIVE** ou **INACTIVE**
- quando ela enfileira novamente as mensagens de uma fila inativa para uma ou mais instâncias ativas

A amostra cria uma mensagem de erro *CLMnnnnE* para relatar um problema que requer atenção.

A cada 30 minutes minutos, o tempo médio de processamento de relatórios de amostra por intervalo de pesquisa e tempo de processamento decorrido. Essas informações são mantidas na mensagem CLM0045I.

Quando as mensagens de estatística estão ativadas **-s**, a amostra relata as informações estatísticas a seguir sobre cada verificação de fila:

- Tempo gasto para processar as filas (em milissegundos)
- Número de filas verificadas
- Número de mudanças feitas ativas/inativas
- Número de mensagens transferidas

Essas informações são relatadas na mensagem CLM0048I.

Os arquivos de relatório podem aumentar rapidamente no modo de depuração e se quebrar rapidamente. Nessa situação, o limite de tamanho de 1 MB para arquivos individuais poderá ser excedido.

AMQSCLM: resolução de problemas
As seções a seguir contêm informações sobre os cenários que podem ser encontrados ao usar a amostra. As informações sobre possíveis explicações para um cenário e opções sobre como resolvê-lo são fornecidas.

Cenário: AMQSCLM não está iniciando

Explicação em potencial: sintaxe incorreta.

Ação: verifique a saída de erro padrão para a sintaxe correta

Explicação em potencial: o gerenciador de filas não está disponível.

Ação: verifique o arquivo de relatório para o ID de mensagem CLM0010E.

Explicação em potencial: não é possível abrir ou criar o arquivo ou arquivos de relatório.

Ação: verifique a saída de erro padrão para mensagens de erro durante a inicialização.

Cenário: AMQSCLM não está mudando uma fila para ACTIVE ou INACTIVE

Explicação em potencial: a fila não está na lista de filas a serem monitoradas

Ação: verifique os valores de parâmetro **-q** e **-f**.

Explicação em potencial: a fila não é uma fila local no cluster correto.

Ação: verifique se a fila é local e se está no cluster correto.

Explicação em potencial: AMQSCLM não está em execução para este gerenciador de filas e cluster.

Ação: inicie AMQSCLM para o gerenciador de filas e cluster relevantes.

Explicação em potencial: a fila é deixada INACTIVE, **CLWLPRTY** =0, porque não tem consumidores. Como alternativa, é deixada como ACTIVE **CLWLPRTY** >=1, porque tem pelo menos um consumidor.

Ação: verifique se os aplicativos de consumo estão conectados à fila.

Explicação em potencial: o servidor de comandos do gerenciador de filas não está em execução.

Ação: verifique se há erros nos arquivos de relatórios.

Cenário: mensagens não estão sendo roteadas em torno de filas INACTIVE

Explicação em potencial: as mensagens são colocadas diretamente no gerenciador de filas que tem a fila inativa e o valor **CLWLUSEQ** da fila não é ANY e o argumento **-u** não está sendo usado para AMQSCLM.

Ação: verifique o valor de **CLWLUSEQ** do gerenciador de filas relevante ou assegure que o argumento **-u** seja usado para AMQSCLM.

Explicação em potencial: não há filas ativas em quaisquer gerenciadores de filas. As mensagens têm a carga de trabalho igualmente balanceada entre todas as filas inativas até que uma fila se torne ativa.

Ação: verifique o status das filas em todos os gerenciadores de filas.

Explicação em potencial: mensagens são colocadas em um gerenciador de filas diferente no cluster daquele que tem a fila inativa e o valor de **CLWLPRTY** 0 atualizado não é propagado para o gerenciador de filas do aplicativo de put.

Ação: verifique se os canais de cluster entre o gerenciador de filas monitorado e o gerenciador de filas de repositório completo estão em execução. Verifique se os canais entre o gerenciador de filas de put e o gerenciador de filas de repositório completo estão em execução. Verifique os logs de erros dos gerenciadores de filas monitorado, de put e de repositório completo.

Explicação em potencial: as instâncias da fila remota estão ativas (CLWLPRTY=1), mas as mensagens não podem ser roteadas para essas instâncias de filas porque o canal emissor de cluster do gerenciador de filas locais não está em execução.

Ação: verifique o status dos canais emissores de cluster do gerenciador de filas locais para o gerenciador, ou gerenciadores, de filas remotas com uma instância ativa da fila.

Cenário: AMQSCLM não está transferindo mensagens de uma fila inativa

Explicação em potencial: a transferência de mensagem não está ativada (**-t**).

Ação: assegure que a transferência de mensagem esteja ativada (**-t**).

Explicação em potencial: a fila não está na lista de filas a serem monitoradas.

Ação: verifique os valores de parâmetro **-q** e **-f**.

Explicação em potencial: AMQSCLM não está em execução para este ou outros gerenciadores de filas no cluster que têm instâncias da mesma fila.

Ação: inicie AMQSCLM.

Explicação em potencial: a fila tem **CLWLUSEQ** = LOCAL ou **CLWLUSEQ** = QMGR e o argumento **-u** não está configurado.

Ação: Configure o parâmetro **-u** ou mude a configuração da fila ou do gerenciador de filas para ANY.

Explicação em potencial: não há instâncias ativas da fila no cluster.

Ação: verifique instâncias da fila com um valor de **CLWLPRTY** igual a 1 ou maior.

Possível explicação: as instâncias de fila remota têm consumidores (**IPPROCS** >=1), mas estão inativos nesses gerenciadores de filas (**CLWLPRTY** =0) porque AMQSCLM não está monitorando essas instâncias remotas.

Ação: assegure que AMQSCLM esteja em execução nesses gerenciadores de filas e/ou que a fila esteja na lista de filas a serem monitoradas verificando os valores dos parâmetros **-q** e **-f**.

Explicação em potencial: as instâncias de filas remotas estão ativas (**CLWLPRTY** =1), mas são vistas como inativas no gerenciador de filas locais (**CLWLPRTY** =0). Essa situação é atribuída ao valor atualizado de **CLWLPRTY** não ter sido propagado para esse gerenciador de filas.

Ação: assegure que os gerenciadores de filas remotas estejam conectados a pelo menos um dos gerenciadores de filas de repositório completo no cluster. Assegure que os gerenciadores de filas de repositório completo estejam funcionando corretamente. Verifique se os canais entre os gerenciadores de filas de repositório completo e os gerenciadores de filas monitorados estão em execução.

Explicação em potencial: As mensagens não estão confirmadas, portanto, elas não são recuperáveis.

Ação: verifique se o aplicativo de envio está funcionando corretamente.

Explicação em potencial: AMQSCLM não tem acesso à fila local na qual as mensagens estão enfileiradas.

Ação: verifique se AMQSCLM está em execução como um usuário com autorização suficiente para acessar a fila.

Explicação em potencial: o servidor de comandos do gerenciador de filas não está em execução.

Ação: inicie o servidor de comandos do gerenciador de filas.

Explicação em potencial: AMQSCLM encontrou um erro.

Ação: verifique se há erros nos arquivos de relatórios.

Explicação em potencial: as instâncias da fila remota estão ativas (CLWLPRTY=1), mas as mensagens não podem ser transferidas para essas instâncias de filas porque o canal emissor de cluster do gerenciador de filas locais não está em execução. Isso é frequentemente acompanhado de um aviso CLM0030W no log de relatório do amqsclm.

Ação: verifique o status dos canais emissores de cluster do gerenciador de filas locais para o gerenciador, ou gerenciadores, de filas remotas com uma instância ativa da fila.

ALW *Programa de amostra para Connection Endpoint Lookup (CEPL)*

A amostra do IBM MQ Connection Endpoint Lookup fornece um módulo de saída simples mas poderoso que oferece aos usuários do IBM MQ uma maneira de recuperar as definições de conexão a partir de um repositório LDAP, como o Tivoli Directory Server.

O Tivoli Directory Server v6.3 Client deve ser instalado para que CEPL possa ser usado.

Um conhecimento de trabalho de administração do IBM MQ nas plataformas suportadas é necessário para usar essa amostra.

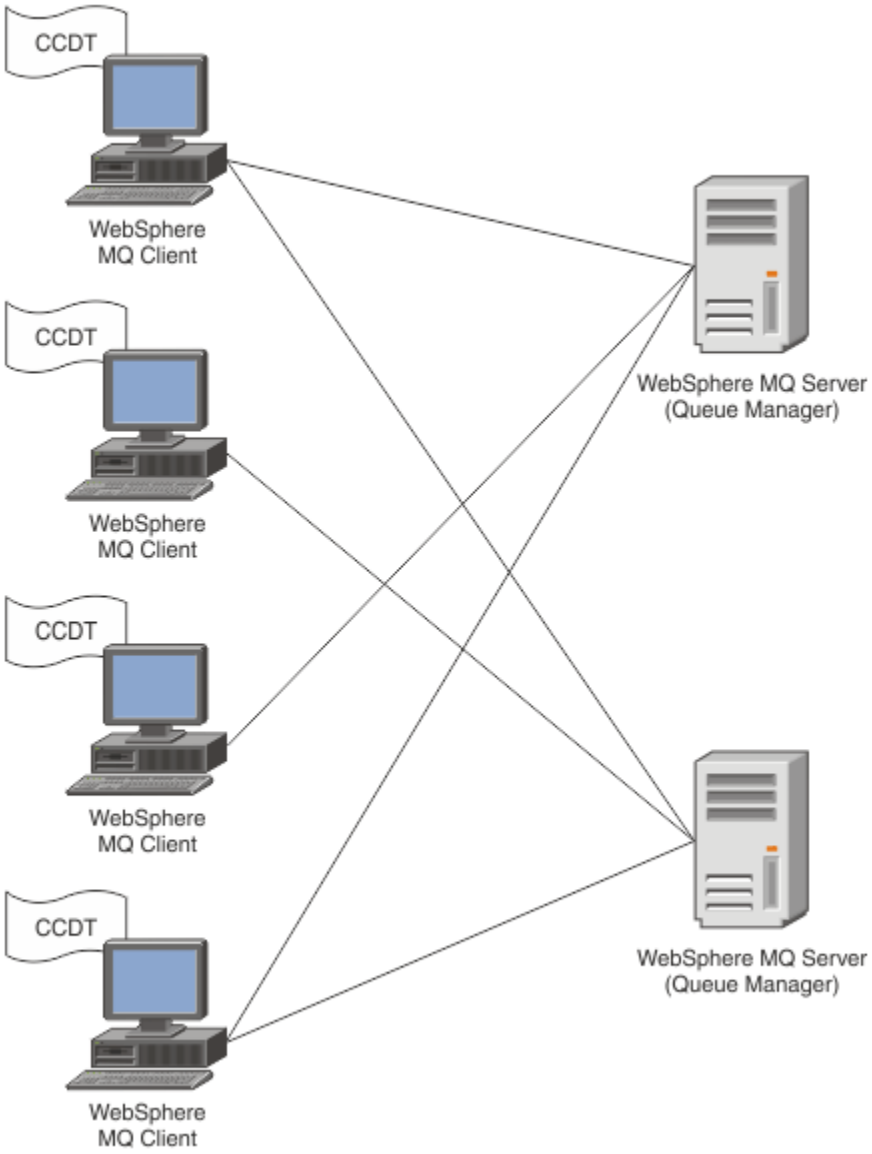
Windows Linux AIX *Introdução*

Configure um repositório global, por exemplo, um diretório LDAP (Lightweight Directory Access Protocol), para armazenar as definições de conexão do cliente para ajudar na manutenção e administração.

Usando um aplicativo IBM MQ Client para estabelecer uma conexão com um Gerenciador de filas por meio de uma Tabela de definição de conexão de cliente (CCDT).

A CCDT é criada por meio da interface IBM MQ MQSC Administration padrão. O usuário deve estar conectado a um Gerenciador de filas para criar definições de conexão de cliente, embora os dados contidos na definição não sejam restritos ao Gerenciador de filas. O arquivo

da CCDT gerado deve ser distribuído manualmente entre máquinas clientes e aplicativos.

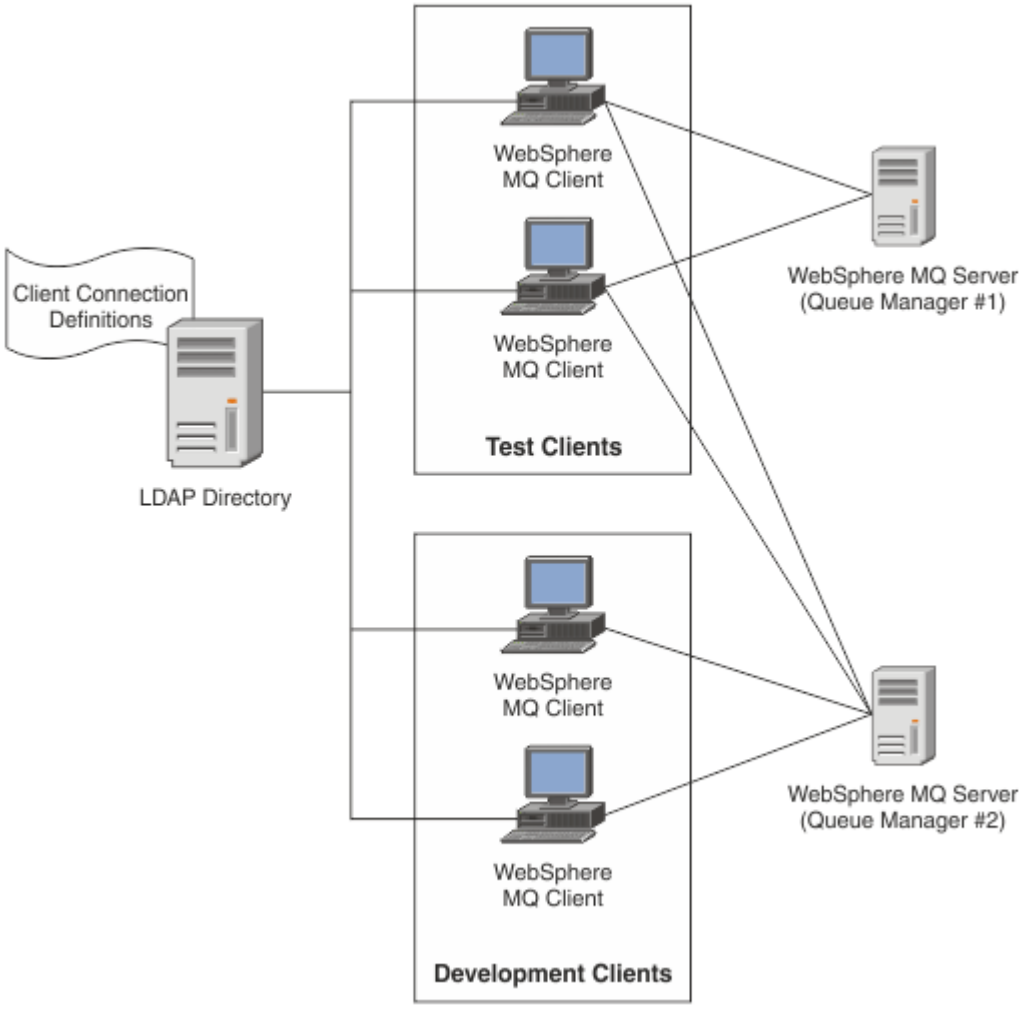


O arquivo da CCDT deve ser distribuído para cada cliente IBM MQ. Quando milhares de clientes podem existir local ou globalmente, logo se tornaria difícil manter e administrar. Uma abordagem mais flexível é necessária para ajudar a assegurar que cada cliente tenha as definições de cliente corretas disponíveis para eles.

Uma dessas abordagens é armazenar as definições de conexão de cliente em um repositório global, como um diretório LDAP (Lightweight Directory Access Protocol). Um diretório LDAP também pode fornecer segurança adicional, indexação e recursos de procura, permitindo, assim, que cada cliente acesse somente as definições de conexão referentes a eles.

O diretório LDAP pode ser configurado de forma que somente definições específicas estejam disponíveis para determinados grupos de usuários. Por exemplo, o Clientes de teste podem acessar o Gerenciador de

filas nº 1 e nº 2, enquanto que os Clientes de desenvolvimento podem acessar somente o Gerenciador de



filas nº 2.

O módulo de saída pode consultar um repositório LDAP, por exemplo, o IBM Tivoli Directory Server, para recuperar definições de canal. Usando as definições de conexão, um aplicativo cliente IBM MQ pode estabelecer conexão com um gerenciador de filas.

O módulo de saída é um módulo de saída preconnect que permite a definição de canal seja obtida durante a chamada MQCONN/MQCONNX a partir de um repositório LDAP.

O módulo de saída e o esquema podem ser implementados por:

- Clientes que já construíram uma base de qualificação usando a tecnologia baseada no arquivo existente da CCDT e desejam aliviar os custos de administração e distribuição.
- Os clientes que já usam sua próprias tecnologia proprietária para distribuir as definições de conexão de cliente.
- Clientes novos ou existentes que atualmente não empregam qualquer tipo de solução de conexão de cliente e desejam usar os recursos oferecidos pelo IBM MQ.
- Clientes novos ou existentes que desejam usar diretamente ou ajustar seu modelo de sistema de mensagens em linha com qualquer arquitetura de negócios LDAP atual.

ALW *Ambientes suportados*

Verifique se você tem um sistema operacional suportado e o software relevante antes de executar a amostra Connection Endpoint Lookup.

O programa de amostra para o IBM MQ Connection Endpoint Lookup requer o software a seguir:

- IBM WebSphere MQ 7.0 ou posterior

- Tivoli Directory Server V6.3 Client ou posterior

Sistemas operacionais suportados:

1. Windows (7/8/2008/2012)
2. AIX
3. Linux
   - RHEL v4 e v5 no System p
   - SUSE v9 e v10 no System p
   - RHEL v4 e v5 do x86-64 de 32 bits e 64 bits
   - SUSE v9 e v10 x86-64 de 32 bits e 64 bits

Nota: A amostra não está disponível para as plataformas a seguir:

- z/OS
- IBM i

ALW *Instalação e configuração*
Instalando e configurando o módulo de saída e o esquema Connection Endpoint.

Instalando o módulo de saída

Durante a instalação do IBM MQ, o módulo de saída é instalado sob `tools/samples/c/preconnexit/bin`. Para plataformas de 32 bits, o módulo de saída deve ser copiado ao `exit/`*`installation_name`*`/` antes que ele possa ser usado. Para plataformas de 64 bits, o módulo de saída deve ser copiado para exit64/*installation_name*/ para que possa ser usado.

Instalando o esquema Connection Endpoint

A saída usa o esquema Connection Endpoint, `ibm-amq.schema`. O arquivo de esquema deve ser importado para qualquer servidor LDAP antes que a saída possa ser usada. Após importar o esquema, valores dos atributos deverão ser incluídos.

Segue um exemplo para importar o esquema Connection Endpoint. O exemplo supõe que o IBM Tivoli Directory Server (ITDS) esteja sendo usado.

- Certifique-se de que o IBM Tivoli Directory Server esteja em execução e, em seguida, copie ou envie por FTP o arquivo `ibm-amq.schema` para o servidor ITDS.
- No servidor ITDS, insira o comando a seguir para instalar o esquema no armazenamento do ITDS, em que *LDAP ID* e *LDAP password* são o DN raiz e a senha do servidor LDAP:

  `ldapadd -D "`*`LDAP ID`*`" -w "`*`LDAP password`*`" -f ibm-amq.schema`
- Em uma janela de comandos, insira o comando a seguir ou use uma ferramenta de terceiro para procurar o esquema para verificação:

```
ldapsearch objectclass=ibm-amqClientConnection
```

Consulte a documentação do Servidor LDAP para obter detalhes adicionais sobre como importar o arquivo de esquema.

Configuração

Uma nova seção chamada PreConnect deve ser incluída no arquivo de configuração do cliente, por exemplo `mqclient.ini`. A seção PreConnect contém as palavras-chave a seguir:

Módulo
O nome do módulo que contém o código de saída da API. Se esse campo contiver o caminho completo do módulo, ele será usado no estado em que se encontra. Caso contrário, a pasta `exit` ou `exit64` na instalação do IBM MQ é procurada.

Função
O nome do ponto de entrada funcional na biblioteca que contém o código de saída `LdapPreConnect` .. A definição de função adere ao protótipo de função de sua empresa.

Atenção: Você deve remover as aspas na instrução de função ao especificar seu ponto de entrada de saída real.

Data
URI do repositório LDAP contendo definições de canal.

O fragmento a seguir é um exemplo das alterações necessárias no arquivo `mqclient.ini`.

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW *Visão geral da saída e do esquema*
Sintaxe e parâmetros usados para estabelecer uma conexão com um gerenciador de filas.

O IBM MQ 9.3 define a sintaxe a seguir para um ponto de entrada em um módulo de saída.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Durante a execução da chamada MQCONN/X, o IBM MQ C Client carrega o módulo de saída que contém uma implementação da sintaxe da função. Em seguida, chama uma função de saída para recuperar as definições de canal. As definições de canal recuperadas são então usadas para estabelecer a conexão com um gerenciador de filas.

Parâmetros

pExitParms
Tipo: entrada/saída PMQNXP

A estrutura do parâmetro de saída PreConnection. A estrutura é alocada e mantida pelo responsável pela chamada da saída.

```
struct tagMQNXP
{
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     ExitId;              /* Type of exit */
  MQLONG     ExitReason;          /* Reason for invoking exit */
  MQLONG     ExitResponse;        /* Response from exit */
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

pQMgrName
Tipo: entrada/saída PMQCHAR

Nome do gerenciador de filas. Na entrada, esse parâmetro é a cadeia de filtros fornecida para a chamada de API MQCONN por meio do parâmetro **QMgrName**. Esse campo pode ficar em branco, ser explícito ou conter determinados caracteres curingas. O campo é mudado pela saída. O parâmetro é NULL quando a saída é chamada com MQXR_TERM.

ppConnectOpts

Tipo: entrada/saída ppConnectOpts

Opções que controlam a ação de MQCONNX. Esse é um ponteiro para uma estrutura de opções de conexão MQCNO que controla a ação da chamada de API MQCONN. O parâmetro é NULL quando a saída é chamada com MQXR_TERM. O cliente MQI sempre fornece uma estrutura MQCNO para a saída, mesmo que ela não tenha sido fornecida originalmente pelo aplicativo. Se um aplicativo fornecer uma estrutura MQCNO, o cliente fará uma duplicata para passá-la para a saída em que é modificado. O cliente retém a propriedade do MQCNO. Um MQCD referenciado por meio do MQCNO tem precedência sobre qualquer definição de conexão fornecida por meio da matriz. O cliente usa a estrutura MQCNO para se conectar ao gerenciador de filas e os outros são ignorados.

pCompCode

Tipo: entrada/saída PMQLONG

Código de conclusão. Ponteiro para um MQLONG que recebe o código de conclusão de saídas. Deve ser um dos valores a seguir:

- MQCC_OK - Conclusão bem-sucedida
- MQCC_WARNING - Aviso (conclusão parcial)
- MQCC_FAILED - Falha na chamada

pReason

Tipo: entrada/saída PMQLONG

Razão que qualifica pCompCode. Ponteiro para um MQLONG que recebe o código de razão de saída. Se o código de conclusão for MQCC_OK, o único valor válido será: MQRC_NONE - (0, x '000') Nenhuma razão para o relatório.

Se o código de conclusão for MQCC_FAILED ou MQCC_WARNING, a função de saída poderá configurar o campo de código de razão como qualquer valor MQRC_* válido.

ALW *Informações de contexto de LDAP do MQ*

A saída usa a estrutura de dados a seguir para informações de contexto.

MQNLDAPCTX

A estrutura MQNLDAPCTX tem o protótipo C a seguir.

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX *Código de amostra para construir a saída de consulta de endpoint de conexão*

É possível usar os fragmentos de código de amostra para compilar a origem no AIX, no Linux ou no Windows.

Compilando a origem

É possível compilar a origem com todas as bibliotecas do cliente LDAP, por exemplo, bibliotecas do IBM Tivoli Directory Server V6.3 Client. Esta documentação supõe que você esteja usando as bibliotecas do Tivoli Directory Server V6.3 Client.

Nota: A biblioteca de saída preconnect é suportada com os servidores LDAP a seguir:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Os fragmentos de código a seguir descrevem como compilar as saídas:

Windows **Compilando a saída na plataforma Windows**

É possível usar o fragmento a seguir para compilar a origem de saída:

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

Nota: Se você estiver usando as bibliotecas do cliente IBM Tivoli Directory Server V6.3 que são compiladas com o compilador Microsoft Visual Studio 2003, poderá obter avisos enquanto estiver compilando as bibliotecas do cliente IBM Tivoli Directory Server V6.3 com o compilador Microsoft Visual Studio 2012 ou mais recente.

Linux AIX **Compilando a saída no AIX, no Linux**

O fragmento de código a seguir é para compilar a origem de saída no Linux. Algumas opções do compilador podem diferir no AIX.

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

O IBM Tivoli Directory Server envia bibliotecas de links estáticas e dinâmicas, mas é possível usar somente um tipo de biblioteca. Este script supõe que você esteja usando as bibliotecas estáticas.

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl

amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW *Chamada do módulo de saída PreConnect*

O módulo de saída PreConnect pode ser chamado com três códigos de razão diferentes: o código de razão MQXR_INIT para inicializar e estabelecer uma conexão com um servidor LDAP, o código de razão MQXR_PRECONNECT para recuperar as definições de canal a partir de um servidor LDAP ou o código de razão MQXR_TERM quando a saída deve ser limpa.

MQXR_INIT

A saída é chamada com o código de razão MQXR_INIT para inicializar e estabelecer uma conexão com um servidor LDAP.

Antes da chamada MQXR_INIT, o campo pExitDataPtr da estrutura do MQNXP é preenchido com o atributo Data a partir da sub-rotina PreConnect dentro do arquivo `mqclient.ini` (ou seja, o LDAP).

Uma URL de LDAP consiste em pelo menos o protocolo, o nome do host, o número da porta e o DN base para a procura. A saída analisa a URL de LDAP contida no campo pExitDataPtr, aloca uma estrutura LDAP Lookup Context MQNLDAPCTX e a preenche apropriadamente. O endereço dessa estrutura está armazenado no campo pExitUserAreaPtr. Falha em analisar corretamente a URL de LDAP resulta no erro MQCC_FAILED.

Nesse momento, a saída conecta e se liga ao servidor LDAP usando os parâmetros **MQNLDAPCTX**. Os identificadores de API de LDAP resultantes também são armazenados nessa estrutura.

MQXR_PRECONNECT

O módulo de saída é chamado com o código de razão MQXR_PRECONNECT para recuperar as definições de canal a partir de um servidor LDAP.

A saída procura no servidor LDAP definições de canal que correspondem ao filtro fornecido. Se o **QMgrNameparameter** contiver um nome de gerenciador de filas específico, a pesquisa retornará todas as definições de canal às quais o valor do atributo LDAP **ibm-amqQueueManagerName** corresponde, para o determinado nome do gerenciador de filas.

Se o parâmetro **QMgrName** for '*' ou ' ' (em branco), então a pesquisa retornará todas as definições de canal para as quais o atributo endpoint **ibm-amqIsClientDefault Connection** é configurado como TRUE.

Após uma procura bem-sucedida, a saída prepara um ou uma matriz de definições de MQCD e retorna para o responsável pela chamada.

MQXR_TERM

A saída é chamada com esse código de razão quando a saída deve ser limpa. Durante essa limpeza, a saída se desconecta do servidor LDAP e libera toda a memória alocada e mantida pela saída, incluindo a estrutura MQNLDAPCTX, a matriz de ponteiros e cada MQCD a qual se refere. Todos os outros campos são configurados para os valores padrão. Os parâmetros de saída **pQMgrName** e **ppConnectOpts** ficam sem uso durante uma saída com o código de razão MQXR_TERM e podem ser NULL.

Referências relacionadas

Sub-rotina PreConnect do arquivo de configuração do cliente

ALW *Esquemas LDAP*

Os dados de conexão do cliente são armazenados em um repositório global denominado diretório LDAP (Lightweight Directory Access Protocol). Um cliente IBM MQ usa um diretório LDAP para obter as definições de conexão. A estrutura das definições de conexão do cliente IBM MQ dentro do diretório LDAP é conhecida como o esquema LDAP. Um esquema LDAP é uma coleta de definições de tipo de atributo, definições de classe de objeto e outras informações que um servidor usa para determinar se uma asserção de valor de atributo ou filtro corresponde aos atributos de uma entrada e se deve permitir, incluir e modificar operações.

Armazenando dados no diretório LDAP

As definições de conexão do cliente estão localizadas sob uma ramificação específica dentro da árvore de diretórios conhecida como o ponto de conexão. Como todos os outros nós dentro de um diretório LDAP, o ponto de conexão possui um Nome Distinto (DN) associado a ele. É possível usar este nó como o ponto

de início para quaisquer consultas que você fizer no diretório. Use a filtragem ao consultar o diretório LDAP para retornar um subconjunto de definições de conexão do cliente. É possível restringir o acesso a subárvores com base nas permissões concedidas em outras partes da árvore de diretórios – por exemplo, para os usuários, departamentos ou grupos.

Definindo seus próprios atributos e classes

Armazene a definição de canal do cliente, modificando o esquema LDAP. Todas as definições de dados LDAP requerem objetos e atributos. Os objetos e atributos são identificados por um número de identificador de objeto (OID) que identifica exclusivamente o objeto ou atributo. Todas as classes dentro de um esquema LDAP herdarão de forma direta ou indireta do objeto superior. O objeto de definição de canal do cliente contém os atributos do objeto superior. Todas as definições de dados LDAP requerem objetos e atributos:

- definições de objeto são coletas de atributos LDAP.
- Atributos são tipos de dados LDAP.

A descrição de cada atributo e como eles são mapeados para as propriedades normais do IBM MQ são descritos em Atributos LDAP.

ALW *Atributos LDAP*

Atributos LDAP definidos são específicos do IBM MQ e são mapeados diretamente para as propriedades de conexão do cliente.

Atributos da sequência do diretório do canal do cliente IBM MQ

Os atributos da sequência de caracteres com seu mapeamento para propriedades do IBM MQ são listados na tabela a seguir. Os atributos podem conter valores da sintaxe de directoryString (Unicode codificado por UTF-8, ou seja, um sistema de codificação de byte variável que inclui IA5/ASCII como um subconjunto). A sintaxe é especificada por seu número de identificação de objeto (OID).

Tabela 166. Atributos da sequência do diretório do canal do cliente IBM MQ

Atributo LDAP	Descrição	Propriedade IBM MQ
CN	O nome comum que consiste no nome do canal e no nome do gerenciador de filas de definição.	
ibm-amqChannelName	O nome da definição do canal.	CHANNEL
ibm-amqConnectionName	O identificador de conexão de comunicação.	CONNAME
ibm-amqDescription	A descrição do canal.	DESCR
ibm-amqLocalAddress	O endereço de comunicação local do canal.	LOCLADDR
ibm-amqModeName	O nome do modo de LU 6.2.	MODENAME
ibm-amqPassword	A senha que pode ser usada.	PASSWORD
ibm-amqQueueManagerName	O nome do gerenciador de filas ou do grupo de gerenciadores de filas ao qual um aplicativo cliente IBM MQ pode solicitar conexão.	QMNAME
ibm-amqSecurityExitUserData	Os dados do usuário que são passados à saída de segurança.	SCYDATA
ibm-amqSecurityExitName	O nome do programa de saída a ser executado pela saída de segurança do canal.	SCYEXIT
ibm-amqSslCipherSpec	Um único CipherSpec para uma conexão TLS.	SSLCIPH
ibm-amqSslPeerName	Verifica o Nome distinto (DN) do certificado do gerenciador de filas ou cliente peer na outra extremidade de um canal do IBM MQ.	SSLPEER

Tabela 166. Atributos da sequência do diretório do canal do cliente IBM MQ (continuação)		
Atributo LDAP	**Descrição**	**Propriedade IBM MQ**
ibm-amqTransactionProgramName	O nome do programa de transação.	TPNAME
ibm-amqUserID	O ID do usuário a ser usado pelo MCA ao tentar iniciar uma sessão SNA segura com um MCA remoto.	USERID

Atributos de número inteiro de conexão do clienteIBM MQ

Os atributos com valores predefinidos (por exemplo, um tipo enumerado) são armazenados como números inteiros padrão. Esses valores são armazenados no diretório LDAP como valores de números inteiros e não usando o nome de constante associado.

Tabela 167. Atributos de número inteiro do diretório do canal do cliente IBM MQ		
Atributo LDAP	**Descrição**	**Propriedade IBM MQ**
ibm-amqConnectionAffinity	Determina se os aplicativos clientes, que se conectam várias vezes por meio do mesmo nome do gerenciador de filas, usam o mesmo canal do cliente.	AFFINITY
ibm-amqClientChannelWeight	Um peso para influenciar qual definição de canal de conexão do cliente é usada.	CLNTWGHT
ibm-amqHeartBeatInterval	O tempo aproximado entre fluxos de pulsação que devem ser passados de um MCA de envio quando não há mensagens na fila de transmissão.	HBINT
ibm-amqKeepAliveInterval	Um valor de tempo limite para um canal.	KAINT
ibm-amqMaximumMessageLength	O comprimento máximo de uma mensagem que pode ser transmitida no canal.	MAXMSGL
ibm-amqSharingConversations	O número máximo de conversas que compartilham cada instância do canal TCP/IP.	SHARECNV
ibm-amqTransportType	O tipo de transporte a ser usado.	TRPTYPE

Atributo booleano do canal do cliente IBM MQ

Esse atributo booleano não é mapeado para nenhuma propriedade do IBM MQ. A sintaxe desse atributo indica um valor booleano.

Tabela 168. Atributo booleano do canal do cliente IBM MQ	
Atributo LDAP	**Descrição**
ibm-amqIsClientDefault	Esse atributo booleano é definido para resolver o problema de procura de entradas cujo atributo ibm-amqQueueManagerName não foi definido.

Atributos da lista de canal do cliente IBM MQ

As propriedades do IBM MQ são armazenadas como atributo de valor único de lista separada por vírgula no diretório LDAP. Os atributos são definidos da mesma maneira que os outros atributos de sequência do diretório. Os atributos da lista juntamente com seu mapeamento para as propriedades do IBM MQ estão descritos na tabela a seguir.

Tabela 169. Atributos da lista de canal do cliente IBM MQ		
Atributo LDAP	**Descrição**	**Propriedade IBM MQ**
ibm-amqHeaderCompression	Uma lista de técnicas de compactação de dados de cabeçalho suportadas pelo canal.	COMPHDR
ibm-amqMessageCompression	Uma lista de técnicas de compactação de dados de mensagem suportadas pelo canal.	COMPMSG
ibm-amqSendExitUserData	Os dados do usuário que são passados à saída de envio.	SENDDATA
ibm-amqSendExitUserName	O nome do programa de saída a ser executado pela saída de envio do canal.	SENDEXIT
ibm-amqReceiveExitUserData	Os dados do usuário que são passados à saída de recebimento.	RCVDATA
ibm-amqReceiveExitName	O nome do programa de saída de usuário a ser executado pelo canal recebe a saída de usuário.	RCVEXIT

ALW *Nome Comum*

O nome comum (CN) consiste no nome do canal e no nome do gerenciador de filas de definição.

É um atributo preexistente.

O formato do CN é:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

Por exemplo:

```
CN=TC1(QM_T1)
```

É possível especificar somente um valor para esse atributo.

Este atributo é um atributo de sequência, e os valores não fazem distinção entre maiúsculas e minúsculas. A subsequência correspondente é ignorada. A correspondência de subsequência é uma regra de correspondência usada no subesquema que especifica o comportamento do atributo em um filtro de procura, usando uma subsequência (por exemplo, CN=jim*, em que CN é um atributo) e contém um ou mais curingas.

ALW *ibm-amqChannelName*

Esse atributo especifica o nome da definição de canal.

Esse atributo possui um único valor de sequência de caracteres com um máximo de 20 caracteres que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A correspondência de subsequência é uma regra de correspondência usada no subesquema que especifica o comportamento do atributo em um filtro de procura, usando uma subsequência e contendo um ou mais curingas.

ALW *ibm-amqDescription*

Este atributo do LDAP fornece a descrição do canal.

Esse atributo tem um valor de sequência única com um máximo de 64 bytes, que não faz distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqConnectionName*
Esse atributo de LDAP é o identificador de conexão de comunicações. Ele especifica os links de comunicações específicas que serão usadas por esse canal.

Esse atributo possui um único valor de sequência de caracteres com um máximo de 264 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqLocalAddress*
Este atributo especifica o endereço de comunicações local para o canal.

Esse atributo tem um único valor de sequência com um máximo de 48 caracteres, que não faz distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqModeName*
Este atributo é para uso com conexões LU 6.2. Ele fornece definição adicional para as características da sessão da conexão quando uma alocação de sessão de comunicação é executada.

Esse atributo tem um único valor da sequência de caracteres de exatamente oito caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqPassword*
Esse atributo LDAP especifica uma senha que possa ser usada pelo MCA ao tentar iniciar uma sessão segura de LU 6.2 com um MCA remoto.

Esse atributo tem um único valor de número inteiro com no máximo 12 dígitos. Não é um atributo pré-existente.

ALW *ibm-amqQueueManagerName*
Esse atributo especifica o nome do gerenciador de filas ou grupo de gerenciadores de filas para o qual um aplicativo cliente IBM MQ pode solicitar conexão.

Esse atributo tem um único valor de sequência com um máximo de 48 caracteres, que não faz distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

Referências relacionadas
“ibm-amqIsClientDefault” na página 1173
Esse atributo booleano soluciona o problema de procura de entradas quando o atributo ibm-amqQueueManagerName não tiver sido definido.

ALW *ibm-amqSecurityExitUserData*
Este atributo LDAP especifica os dados do usuário que são transmitidos para a saída de segurança.

Esse atributo tem um único valor de sequência com um máximo de 999 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqSecurityExitName*
Este atributo LDAP especifica o nome do programa de saída a ser executado pela saída de segurança do canal.

Deixe em branco se nenhuma saída de segurança do canal estiver em vigor.

Esse atributo tem um único valor de sequência com um máximo de 999 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Este atributo não é um pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqSslCipherSpec*
Este atributo LDAP especifica um único CipherSpec para uma conexão TLS.

Esse atributo tem um único valor de sequência com um máximo de 32 caracteres, que não faz distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqSslPeerName*
Este atributo LDAP é usado para verificar o DN (Nome Distinto) do certificado do gerenciador de filas ou cliente peer na outra extremidade de um canal do IBM MQ.

Este atributo LDAP tem um único valor de sequência com um máximo de 1024 bytes, que não fazem distinção entre maiúsculas e minúsculas. Não é um pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqTransactionProgramName*
Este atributo LDAP especifica o nome do programa de transação. Ele deve ser usado com conexões LU 6.2.

Este atributo possui um único valor de sequência com um máximo de 64 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqUserID*
Esse atributo LDAP especifica o ID do usuário a ser usado pelo MCA ao tentar iniciar uma sessão SNA segura com um MCA remoto.

Esse atributo tem um único valor de sequência de exatamente 12 caracteres, que não faz distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

ALW *ibm-amqConnectionAffinity*
Esse atributo LDAP especifica se aplicativos clientes, que se conectam várias vezes usando o mesmo nome de gerenciador de filas, usam o mesmo canal do cliente.

Esse atributo tem um único valor de número inteiro. Não é um atributo pré-existente.

ALW *ibm-amqClientChannelWeight*
Esse atributo LDAP especifica um peso que influencia qual definição de canal de conexão do cliente é usada.

O atributo de peso do canal do cliente é usado para influenciar a seleção de definições de canal do cliente quando mais de uma definição apropriada estiver disponível.

Esse atributo tem um único valor de número inteiro. Não é um atributo pré-existente.

ALW *ibm-amqHeartBeatInterval*
Esse atributo LDAP especifica o tempo aproximado entre fluxos de pulsação que devem ser passados a partir de um MCA de envio quando não houver mensagens na fila de transmissão.

Esse atributo tem um único valor de número inteiro. Não é um atributo pré-existente. O valor padrão é 1. O padrão é configurado na operação da variável de ambiente MQSERVER atual.

ALW *ibm-amqKeepAliveInterval*
Esse atributo LDAP é usado para especificar um valor de tempo limite para um canal.

O valor desse atributo é passado para a pilha de comunicações especificando a sincronização keep-alive para o canal. É possível usar essa opção para especificar um valor de keep-alive diferente para cada canal.

Esse atributo tem um único valor de número inteiro. Não é um atributo pré-existente.

ALW *ibm-amqMaximumMessageLength*
Esse atributo LDAP especifica o comprimento máximo de uma mensagem que pode ser transmitida no canal.

O valor padrão desse atributo é 104857600 de acordo com a operação da variável de ambiente MQSERVER atual. Esse atributo tem um valor de número inteiro único e não é um atributo pré-existente.

ALW *ibm-amqSharingConversations*
Esse atributo LDAP especifica o número máximo de conversas que compartilham cada instância do canal TCP/IP.

Esse atributo tem um único valor de número inteiro. Esse atributo não é um atributo pré-existente.

ALW *ibm-amqTransportType*
Esse atributo LDAP especifica o tipo de transporte a ser usado.

Esse atributo tem um único valor de número inteiro. Não é um atributo pré-existente.

ALW *ibm-amqIsClientDefault*
Esse atributo booleano soluciona o problema de procura de entradas quando o atributo ibm-amqQueueManagerName não tiver sido definido.

Módulos de saída Preconnect geralmente procuram nos servidores LDAP com o valor do atributo ibm-amqQueueManagerName como o critério de procura. Essa consulta retornaria todas as entradas em que o valor do atributo ibm-amqQueueManagerName corresponda ao nome do gerenciador de filas especificado na chamada MQCONN/X. No entanto, ao usar as tabelas de definição de canal do cliente (CCDT), é possível configurar o nome do gerenciador de filas em uma chamada MQCONN/X como em branco ou prefixar o nome com um asterisco (*). Se o nome do gerenciador de filas estiver em branco, o

cliente se conectará ao gerenciador de filas padrão. Se o nome tiver um asterisco (*) como prefixo para o gerenciador de filas, então, o cliente conecta qualquer gerenciador de filas.

De forma semelhante, o atributo ibm-amqQueueManagerName em uma entrada pode ser deixado indefinido. Nesse caso, é esperado que o cliente usando essas informações do endpoint possa se conectar a qualquer gerenciador de filas. Por exemplo, uma entrada contém as linhas a seguir:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

Neste exemplo, o cliente tenta se conectar ao gerenciador de filas especificado em execução em `myhost`.

No entanto, em Servidores LDAP, uma procura não é feita em um valor de atributo que não foi definido. Por exemplo, se uma entrada contiver as informações de conexão, exceto ibm-amqQueueManagerName, então, os resultados da procura não incluiriam esta entrada. Para superar esse problema, é possível configurar ibm-amqIsClientDefault. Esse é um atributo booleano e supõe-se que tenha um valor de FALSE, se não definido.

Para entradas em que ibm-amqQueueManagerName não foi definido e que se espera que façam parte da procura, configure ibm-amqIsClientDefault para TRUE. Quando um espaço em branco ou um asterisco (*) é especificado como o nome do gerenciador de filas em uma chamada para MQCONN/X, a saída preconnect procura no servidor LDAP todas as entradas em que o valor do atributo ibm-amqIsClientDefault está configurado para TRUE.

Nota: Não configure ou defina o atributo ibm-amqQueueManagerName se ibm-amqIsClientDefault estiver configurado para TRUE.

Referências relacionadas

“ibm-amqQueueManagerName” na página 1171
Esse atributo especifica o nome do gerenciador de filas ou grupo de gerenciadores de filas para o qual um aplicativo cliente IBM MQ pode solicitar conexão.

ALW *ibm-amqHeaderCompression*

Este atributo LDAP é uma lista de técnicas de compactação de dados de cabeçalho suportadas pelo canal.

O tamanho máximo desse atributo é de 48 caracteres. Não é um atributo pré-existente.

É possível especificar somente um valor para esse atributo.

Esse atributo de lista é especificado como sequências de diretório usando um formato separado por vírgula. Por exemplo, o valor especificado para **ibm-amqHeaderCompression** é `0` que é mapeado para `NONE`. Quaisquer valores que excedem o limite máximo permitido são ignorados pelo cliente. Por exemplo, ibm-amqHeaderCompression contém um máximo de 2 números inteiros na lista.

ALW *ibm-amqMessageCompression*

Esse atributo LDAP é uma lista de técnicas de compactação de dados de mensagem suportadas pelo canal.

O tamanho máximo desse atributo é de 48 caracteres. Não é um atributo pré-existente.

Esse atributo não suporta diversos valores.

V 9.4.0 Esse atributo de lista é especificado como sequências de diretório usando um formato separado por vírgula. Por exemplo, o valor especificado para esse atributo é 1,2,4,16,32 que é mapeado para a sequência de compactação subjacente RLE, ZLIBFAST, ZLIBHIGH, LZ4FASTe LZ4HIGH.

Quaisquer valores que excedem o limite máximo permitido são ignorados pelo cliente. Por exemplo, ibm-amqMessageCompression contém um máximo de 16 números inteiros na lista.

ALW *ibm-amqSendExitUserData*
Este atributo LDAP especifica dados do usuário que são transmitidos para a saída de envio.

Este atributo LDAP possui um valor de sequência de caracteres única, com um máximo de 999 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

Nota: ibm-amqSendExitName e **ibm-amqSendExitUserData** precisam ser sincronizados em pares. Os dados do usuário devem ser sincronizados com o nome de saída. Portanto, se um for especificado, o outro também deve ser especificado simetricamente, mesmo se não contiver dados.

ALW *ibm-amqSendExitName*
Esse atributo LDAP especifica o nome do programa de saída a ser executado pela saída de envio de canal.

Esse atributo tem um único valor de sequência com um máximo de 999 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

Nota: ibm-amqSendExitName e **ibm-amqSendExitUserData** devem ser sincronizados em pares. Os dados do usuário devem ser sincronizados com o nome de saída. Portanto, se um for especificado, o outro também deverá ser simetricamente especificado, mesmo que não contenha nenhum dado.

ALW *ibm-amqReceiveExitUserData*
Este atributo LDAP especifica os dados do usuário que são transmitidos para a saída de recebimento.

É possível executar uma sequência de saídas de recebimento. A sequência de dados do usuário para uma série de saídas é separada por uma vírgula, espaços ou ambos.

Esse atributo tem um único valor de sequência com um máximo de 999 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

Nota: ibm-amqReceiveExitName e **ibm-amqReceiveExitUserData** devem ser sincronizados em pares. Os dados do usuário devem ser sincronizados com o nome de saída. Portanto, se um for especificado, o outro também deverá ser simetricamente especificado, mesmo que não contenha nenhum dado.

ALW *ibm-amqReceiveExitName*
Este atributo LDAP especifica o nome do programa de saída de usuário a ser executado pela saída de usuário de recebimento do canal.

Este atributo é uma lista de nomes de programas que devem ser executados em sucessão. Deixe em branco se nenhuma saída de usuário de recebimento do canal estiver em vigor.

Esse atributo tem um único valor de sequência com um máximo de 999 caracteres, que não fazem distinção entre maiúsculas e minúsculas. Não é um atributo pré-existente.

A subsequência correspondente é ignorada. A subsequência correspondente é uma regra de correspondência usada em sub-esquema que especifica o comportamento do atributo em um filtro de procura.

Nota: ibm-amqReceiveExitName e **ibm-amqReceiveExitUserData** devem ser sincronizados em pares. Os dados do usuário devem ser sincronizados com o nome de saída. Portanto, se um for

especificado, o outro também deverá ser especificado simetricamente, mesmo que não contenha nenhum dado.

z/OS Using the sample programs for z/OS

The sample procedural applications that are delivered with IBM MQ for z/OS demonstrate typical uses of the Message Queue Interface (MQI).

About this task

IBM MQ for z/OS also provides sample data-conversion exits, described in "Escrevendo saídas de conversão de dados" on page 995.

All the sample applications are supplied in source form; several are also supplied in executable form. The source modules include pseudocode that describes the program logic.

Note: Although some of the sample applications have basic panel-driven interfaces, they do not aim to demonstrate how to design the look and feel of your applications. For more information about how to design panel-driven interfaces for non-programmable terminals, see the *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) and its addendum (GG22-9508). These provide guidelines to help you to design applications that are consistent both within the application and across other applications.

Procedure

- Use the following links to find out more about the sample programs:
  - "Features demonstrated in the sample applications for z/OS" on page 1176
  - "Preparing and running sample applications for the batch environment on z/OS" on page 1183
  - "Preparing sample applications for the TSO environment on z/OS" on page 1185
  - "Preparing the sample applications for the CICS environment on z/OS" on page 1187
  - "Preparing the sample application for the IMS environment on z/OS" on page 1191
  - "The Put samples on z/OS" on page 1192
  - "The Get samples on z/OS" on page 1194
  - "The Browse sample on z/OS" on page 1196
  - "The Print Message sample on z/OS" on page 1198
  - "The Queue Attributes sample on z/OS" on page 1202
  - "The Mail Manager sample on z/OS" on page 1203
  - "The Credit Check sample on z/OS" on page 1210
  - "The Message Handler sample on z/OS" on page 1221
  - "The Asynchronous Put sample on z/OS" on page 1224
  - "The Batch Asynchronous Consumption sample on z/OS" on page 1225
  - "The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS" on page 1227
  - "The Publish/Subscribe sample on z/OS" on page 1229
  - "The Set and Inquire message property sample on z/OS" on page 1232

Related tasks

"Usando os programas de amostra em multiplataformas" on page 1071
Estes programas processuais de amostra são entregues com o produto. As amostras são escritas em C e em COBOL, e demonstram os usos típicos do Message Queue Interface (MQI).

z/OS *Features demonstrated in the sample applications for z/OS*

This section summarizes the MQI features demonstrated in each of the sample applications, shows the programming languages in which each sample is written, and the environment in which each sample runs.

z/OS *Put samples on z/OS*

The Put samples demonstrate how to put messages on a queue using the MQPUT call.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

The program is delivered in COBOL and C, and runs in the batch and CICS environment. See Table 172 on page 1184 for the batch application and Table 179 on page 1188 for the CICS application.

z/OS *Get samples on z/OS*

The Get samples demonstrate how to get messages from a queue using the MQGET call.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

The program is delivered in COBOL and C, and runs in the batch and CICS environment. See Table 172 on page 1184 for the batch application and Table 179 on page 1188 for the CICS application.

z/OS *Browse sample on z/OS*

The Browse sample demonstrates how to use the Browse option to find a message, print it, then step through the messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET for browsing messages
- MQCLOSE
- MQDISC

The program is delivered in the COBOL, assembler, PL/I, and C languages. The application runs in the batch environment. See Table 173 on page 1184 for the batch application.

z/OS *Print Message sample on z/OS*

The Print Message sample demonstrates how to remove a message from a queue and print the data in the message, together with all the fields of its message descriptor. It can, optionally, display all of the message properties associated with each message.

By removing comment characters from two lines in the source module, you can change the program so that it browses, rather than removes, the messages on a queue. This program can usefully be used for diagnosing problems with an application that is putting messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET for removing messages from a queue (with an option to browse)

- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

The program is delivered in the C language. The application runs in the batch environment. See Table 174 on page 1184 for the batch application.

z/OS *Queue Attributes sample on z/OS*

The Queue Attributes sample demonstrates how to inquire about and set the values of IBM MQ for z/OS object attributes.

The application uses these MQI calls:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

The program is delivered in the COBOL, assembler, and C languages. The application runs in the CICS environment. See Table 180 on page 1188 for the CICS application.

z/OS *Mail Manager sample on z/OS*

Considerations to note when using Mail Manager sample.

The Mail Manager sample demonstrates these techniques:

- Using alias queues
- Using a model queue to create a temporary dynamic queue
- Using reply-to queues
- Using syncpoints in the CICS and batch environments
- Sending commands to the system-command input queue
- Testing return codes
- Sending messages to remote queue managers, both by using a local definition of a remote queue and by putting messages directly on a named queue at a remote queue manager

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Three versions of the application are provided:

- A CICS application written in COBOL
- A TSO application written in COBOL
- A TSO application written in C

The TSO applications use the IBM MQ for z/OS batch adapter and include some ISPF panels.

See Table 177 on page 1186 for the TSO application, and Table 181 on page 1189 for the CICS application.

z/OS *Credit Check sample on z/OS*

This information contains points to consider when using Credit Check sample.

The Credit Check sample is a suite of programs that demonstrates these techniques:

- Developing an application that runs in more than one environment
- Using a model queue to create a temporary dynamic queue
- Using a correlation identifier
- Setting and passing context information
- Using message priority and persistence
- Starting programs by using triggering
- Using reply-to queues
- Using alias queues
- Using a dead-letter queue
- Using a namelist
- Testing return codes

The application uses these MQI calls:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET for browsing and getting messages, using the wait and signal options, and for getting a specific message
- MQINQ
- MQSET
- MQCLOSE

The sample can run as a stand-alone CICS application. However, to demonstrate how to design a message queuing application that uses the facilities provided by both the CICS and IMS environments, one module is also supplied as an IMS batch message processing program.

The CICS programs are delivered in C and COBOL. The single IMS program is delivered in C.

See Table 182 on page 1189 for the CICS application, and Table 184 on page 1191 for the IMS application.

z/OS *The Message Handler sample on z/OS*

The Message Handler sample allows you to browse, forward, and delete messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE

- MQDISC

The program is delivered in C and COBOL programming languages. The application runs under TSO. SeeTable 178 on page 1187for the TSO application.

z/OS *Distributed queuing exit samples on z/OS*

A table of source programs of Distributed queuing exit samples.

The names of the source programs of the distributed queuing exit samples are listed in the following table:

Table 170. Source for the distributed queuing exit samples

Member name	For language	Description	Supplied in library
CSQ4BAX0	Assembler	Source program	SCSQASMS
CSQ4BCX1	C	Source program	SCSQC37S
CSQ4BCX2	C	Source program	SCSQC37S
CSQ4BCX4	C	Source program	SCSQC37S

Note: The source programs are link-edited with CSQXSTUB.

z/OS *Data-conversion exit samples on z/OS*

A skeleton is provided for a data-conversion exit routine, and a sample is shipped with IBM MQ illustrating the MQXCNVC call.

The names of the source programs of the data-conversion exit samples are listed in the following table:

Table 171. Source for the data conversion exit samples (assembler language only)

Member name	Description	Supplied in library
CSQ4BAX8	Source program	SCSQASMS
CSQ4BAX9	Source program	SCSQASMS
CSQ4CAX9	Source program	SCSQASMS

Note: The source programs are link-edited with CSQASTUB.

See “Escrevendo saídas de conversão de dados” on page 995 for more information.

z/OS *Publish/Subscribe samples on z/OS*

The Publish/Subscribe sample programs demonstrate the use of the publish and subscribe features in IBM MQ.

There are four C and two COBOL programming language sample programs demonstrating how to program to the IBM MQ Publish/Subscribe interface.

The applications use these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH

- MQDLTMH
- MQINQMP

The Public/Subscribe sample programs are delivered in the C and COBOL programming languages. The sample applications run in the batch environment. See Publish/Subscribe samples for the batch applications.

z/OS *Configuring a queue manager to accept client connections on z/OS*

Before you can run the sample applications, you must first create a queue manager. You can then configure the queue manager to securely accept incoming connection requests from applications that are running in client mode.

Before you begin

Ensure the queue manager already exists and has been started. Determine whether channel authentication records are already enabled by issuing the MQSC command:

```
DISPLAY QMGR CHLAUTH
```

Important: This task expects that channel authentication records are enabled. If this is a queue manager used by other users and applications, changing this setting will affect all other users and applications. If your queue manager does not make use of channel authentication records then step 4 can be replaced with an alternate authentication method (for example a security exit) which sets the MCAUSER to the *non-privileged-user-id* you will obtain in step "1" on page 1181.

You must know which channel name your application expects to use so that the application can be permitted to use the channel. You must also know which objects, for example queues or topics, your application expects to use so that your application can be permitted to use them.

About this task

This task creates a non-privileged user ID to be used for a client application which connects to the queue manager. Access is granted for the client application only to be able to use the channel it needs and the queue it needs by use of this user ID.

Procedure

1. Obtain a user ID on the system your queue manager is running on.

   For this task this user ID must not be a privileged administrative user. This user ID is the authority under which the client connection will run on the queue manager.
2. Start a listener program.
   a) Ensure that your channel initiator is started. If not, start it by issuing the **START CHINIT** command.
   b) Start the listener program by issuing the following command:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   where *nnnn* is your chosen port number.
3. If your application uses the SYSTEM.DEF.SVRCONN then this channel is already defined. If your application uses another channel, create it by issuing the MQSC command:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name is the name of your channel.
4. Create a channel authentication rule allowing only the IP address of your client system to use the channel by issuing the MQSC command:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

where

channel-name is the name of your channel.
client-machine-IP-address is the IP address of your client system. If your sample client application is running on the same machine as the queue manager then use an IP address of '127.0.0.1' if your application is going to connect using 'localhost'. If several different client machines are going to connect in, you can use a pattern or a range instead of a single IP address. See Generic IP addresses for details.
non-privileged-user-id is the user ID you obtained in step “1” on page 1181

5. If your application uses the SYSTEM.DEFAULT.LOCAL.QUEUE, then this queue is already defined. If your application uses another queue, create it by issuing the MQSC command:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

where *queue-name* is the name of your queue.

6. Grant access to connect to and inquire the queue manager:
   a) Ensure that your channel initiator is started. If not, start the channel initiator by issuing the START CHINIT command.
   b) Start a TCP listener, for example issue the following command:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

   where *nnnn* is your chosen port number.

7. If your application is a point-to-point application, that is it makes use of queues, grant access to allow inquiring and the putting and getting messages using your queue by the user ID to be used, by issuing the MQSC commands:

   Issue the RACF commands:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

   where

   qmgr-name is the name of your queue manager
   queue-name is the name of your queue.
   non-privileged-user-id is the user ID you obtained in step “1” on page 1181

8. If your application is a publish/subscribe application, that is it makes use of topics, grant access to allow publishing and subscribing using your topic by the user ID to be used, by issuing the following RACF commands:

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

   where

   qmgr-name is the name of your queue manager
   non-privileged-user-id is the user ID you obtained in step “1” on page 1181

This will give *non-privileged-user-id* access to any topic in the topic tree, alternatively, you can define a topic object using **DEFINE TOPIC** and grant accesses only to the part of the topic tree referenced by that topic object. For more information, see Controlling user access to topics.

What to do next

Your client application can now connect to the queue manager and put or get messages using the queue.

Related concepts

z/OS Authority to work with IBM MQ objects on z/OS

Related reference

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC

z/OS *Preparing and running sample applications for the batch environment on z/OS*

To prepare a sample application that runs in the batch environment, perform the same steps that you would when building any batch IBM MQ for z/OS application.

These steps are listed in "Building z/OS batch applications" on page 1036.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQLOAD load library.

Note: The assembler language version of the Browse sample uses data control blocks (DCBs), so you must link-edit it using RMODE(24).

The library members to use are listed in Table 172 on page 1184, Table 173 on page 1184, Table 174 on page 1184, and Table 175 on page 1185.

You must edit the run JCL supplied for the samples that you want to use (see Table 172 on page 1184, Table 173 on page 1184, Table 174 on page 1184, and Table 175 on page 1185).

The PARM statement in the supplied JCL contains a number of parameters that you need to modify. To run the C sample programs, separate the parameters by spaces; to run the assembler, COBOL, and PL/I sample programs, separate them by commas. For example, if the name of your queue manager is CSQ1 and you want to run the application with a queue named LOCALQ1, in the COBOL, PL/I, and assembler-language JCL, your PARM statement should look like this:

```
PARM=(CSQ1,LOCALQ1)
```

In the C language JCL, your PARM statement should look like this:

```
PARM=('CSQ1 LOCALQ1')
```

You are now ready to submit the jobs.

z/OS *Names of the sample batch applications on z/OS*

A summary of the programs that are supplied for sample batch applications.

The batch application programs are summarized in the following tables:

- Table 172 on page 1184 Put and Get samples
- Table 173 on page 1184 Browse sample
- Table 174 on page 1184 Print message sample
- Table 175 on page 1185 Publish/Subscribe samples
- Table 176 on page 1185 Other samples

Table 172. Batch Put and Get samples

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4BCJ1	C	Get source program	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Put source program	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	Sample run JCL for CSQ4BCJ1 and CSQBCK1	SCSQPROC	None
CSQ4BVJ1	COBOL	Get source program	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Put source program	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	Sample run JCL for CSQBVJ1 and CSQBVK1	SCSQPROC	None

Table 173. Batch Browse sample

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4BVA1	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	Sample run JCL for CSQ4BVA1	SCSQPROC	None
CSQ4BAA1	Assembler	Source program	SCSQASMS	SCSQLOAD
CSQ4BAAR	Assembler	Sample run JCL for CSQ4BAA1	SCSQPROC	None
CSQ4BCA1	C	Source program	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	Sample run JCL for CSQ4BCA1	SCSQPROC	None
CSQ4BPA1	PL/I	Source program	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	Sample run JCL for CSQ4BPA1	SCSQPROC	None

Table 174. Batch Print Message sample (C language only)

Member name	Description	Source file supplied in library	Executable file supplied in library
CSQ4BCG1	Source program	SCSQC37S	SCSQLOAD
CSQ4BCGR	Sample run JCL for CSQ4BCG1	SCSQPROC	None
CSQ4BCL1	Browse source program	SCSQC37S	SCSQLOAD
CSQ4BCLR	Sample run JCL for CSQ4BCL1	SCSQPROC	None

Table 175. Publish/Subscribe samples

Member name	For language	Description	Source file supplied in library	JCL in SCSQPROC	Executable file supplied in library
CSQ4BCP1	C	Publish to topic source program	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	Subscribe to topic and get messages source program	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	Subscribe to topic using a user provided destination and get messages source program	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	Subscribe to topic using extended options and get messages source program	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	Publish to topic source program	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	Subscribe to topic and get messages source program	SCSQCOBS	CSQ4BVPS	SCSQLOAD

Table 176. Other samples

Member name	For language	Description	Source file supplied in library	JCL in SCSQPROC	Executable file supplied in library
CSQ4BCS1	C	Asynchronous consumption source program	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	Asynchronous Put, and Check status source program	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	Inquire message properties source program	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	Set message properties source program	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS *Preparing sample applications for the TSO environment on z/OS*

To prepare a sample application that runs in the TSO environment, perform the same steps that you would when building any batch IBM MQ for z/OS application.

These steps are listed in “Building z/OS batch applications” on page 1036. The library members to use are listed in Table 177 on page 1186.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQLOAD load library.

For the Mail Manager sample application, ensure that the queues that it uses are available on your system. They are defined in the member **thlqual**.SCSQPROC(CSQ4CVD). To ensure that these queues are always available, you could add these members to your CSQINP2 initialization input data set, or use the CSQUTIL program to load these queue definitions.

z/OS *Names of the sample TSO applications on z/OS*

Information about the names of the programs that are supplied for each of the sample TSO applications, and the libraries where the source, JCL, and, for the Message Handler sample only, the executable files reside.

The TSO application programs are summarized in the following tables:

- Table 177 on page 1186 Mail manager sample
- Table 178 on page 1187 Message handler sample

These samples use ISPF panels. You must therefore include the ISPF stub, ISPLINK, when you link-edit the programs.

Table 177. TSO Mail Manager sample

Member name	For language	Description	Source file supplied in library
CSQ4CVD	independent	IBM MQ for z/OS object definitions	SCSQPROC
CSQ40	independent	ISPF messages	SCSQMSGE
CSQ4RVD1	COBOL	CLIST to initiate CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Source program for Menu program	SCSQCOBS
CSQ4TVD2	COBOL	Source program for Get Mail program	SCSQCOBS
CSQ4TVD4	COBOL	Source program for Send Mail program	SCSQCOBS
CSQ4TVD5	COBOL	Source program for Nickname program	SCSQCOBS
CSQ4VDP1-6	COBOL	Panel definitions	SCSQPNLA
CSQ4VD0	COBOL	Data definition	SCSQCOBC
CSQ4VD1	COBOL	Data definition	SCSQCOBC
CSQ4VD2	COBOL	Data definition	SCSQCOBC
CSQ4VD4	COBOL	Data definition	SCSQCOBC
CSQ4RCD1	C	CLIST to initiate CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Source program for Menu program	SCSQC37S
CSQ4TCD2	C	Source program for Get Mail program	SCSQC37S

Table 177. TSO Mail Manager sample (continued)			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4TCD4	C	Source program for Send Mail program	SCSQC37S
CSQ4TCD5	C	Source program for Nickname program	SCSQC37S
CSQ4CDP1-6	C	Panel definitions	SCSQPNLA
CSQ4TC0	C	Include file	SCSQC370

Table 178. TSO Message Handler sample				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4TCH0	C	Data definition	SCSQC370	None
CSQ4TCH1	C	Source program	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	Source program	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	Source program	SCSQC37S	SCSQLOAD
CSQ4RCH1	C and COBOL	CLIST to initiate CSQ4TCH1 or CSQ4TVH1	SCSQCLST	None
CSQ4CHP1	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP2	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP3	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP9	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4TVH0	COBOL	Data definition	SCSQCOBC	None
CSQ4TVH1	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	Source program	SCSQCOBS	SCSQLOAD

z/OS *Preparing the sample applications for the CICS environment on z/OS*

Before you run the CICS sample programs, log on to CICS using a LOGMODE of 32702. This is because the sample programs have been written to use a 3270 mode 2 screen.

To prepare a sample application that runs in the CICS environment, perform the following steps:

1. Create the symbolic description map and the physical screen map for the sample by assembling the BMS screen definition source (supplied in library **thlqual**.SCSQMAPS, where **thlqual** is the high-level qualifier used by your installation). When you name the maps, use the name of the BMS screen definition source (not available for Put and Get sample programs), but omit the last character of that name.
2. Perform the same steps that you would when building any CICS IBM MQ for z/OS application. These steps are listed in “Building CICS applications in z/OS” on page 1039. The library members to use are listed in Table 179 on page 1188, Table 180 on page 1188, Table 181 on page 1189, and Table 182 on page 1189.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQCICS load library.

3. Identify the map set, programs, and transaction to CICS by updating the CICS system definition (CSD) data set. The definitions that you require are in the member **thlqual**.SCSQPROC(CSQ4S100). For guidance on how to do this, see *The CICS-IBM MQ Adapter* section in the CICS Transaction Server for z/OS 4.1 product documentation at: CICS Transaction Server for z/OS 4.1, The CICS-IBM MQ adapter.

   Note: For the Credit Check sample application, you get an error message at this stage if you have not already created the VSAM data set that the sample uses.

4. For the Credit Check and Mail Manager sample applications, ensure that the queues that they use are available on your system. For the Credit Check sample, they are defined in the member **thlqual**.SCSQPROC(CSQ4CVB) for COBOL, and **thlqual**.SCSQPROC(CSQ4CCB) for C. For the Mail Manager sample, they are defined in the member **thlqual**.SCSQPROC(CSQ4CVD). To ensure that these queues are always available, you could add these members to your CSQINP2 initialization input data set, or use the CSQUTIL program to load these queue definitions.

   For the Queue Attributes sample application, you could use one or more of the queues that are supplied for the other sample applications. Alternatively, you could use your own queues. However, in the form that it is supplied, this sample works only with queues that have the characters CSQ4SAMP in the first eight bytes of their name.

z/OS *Names of the sample CICS applications on z/OS*

This topic provides a summary of the programs supplied for sample CICS applications.

The CICS application programs are summarized in the following tables:

- Table 179 on page 1188 Put and Get samples
- Table 180 on page 1188 Queue Attributes sample
- Table 181 on page 1189 Mail Manager sample (COBOL only)
- Table 182 on page 1189 Credit Check sample
- Table 183 on page 1190 Asynchronous Consumption and Publish/Subscribe samples

Table 179. CICS Put and Get samples

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4CCK1	C	Put source program	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Get source program	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Get source program	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Put source program	SCSQCOBS	SCSQCICS
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

Table 180. CICS Queue Attributes sample

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4CVC1	COBOL	Source program	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Message definition	SCSQCOBC	None

Table 180. CICS Queue Attributes sample (continued)

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4VCMS	COBOL	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4CAC1	Assembler	Source program	SCSQASMS	SCSQCICS
CSQ4AMSG	Assembler	Message definition	SCSQMACS	None
CSQ4ACMS	Assembler	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4CCC1	C	Source program	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Message definition	SCSQC370	None
CSQ4CCMS	C	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

Table 181. CICS Mail Manager sample (COBOL only)

Member name	Description	Source file supplied in library
CSQ4CVD	IBM MQ for z/OS object definitions	SCSQPROC
CSQ4CVD1	Source for Menu program	SCSQCOBS
CSQ4CVD2	Source for Get Mail program	SCSQCOBS
CSQ4CVD3	Source for Display Message program	SCSQCOBS
CSQ4CVD4	Source for Send Mail program	SCSQCOBS
CSQ4CVD5	Source for Nickname program	SCSQCOBS
CSQ4VDMS	BMS screen definition source	SCSQMAPS
CSQ4S100	CICS system definition data set	SCSQPROC
CSQ4VD0	Data definition	SCSQCOBC
CSQ4VD3	Data definition	SCSQCOBC
CSQ4VD4	Data definition	SCSQCOBC

Table 182. CICS Credit Check sample

Member name	For language	Description	Source file supplied in library
CSQ4CVB	independent	IBM MQ object definitions	SCSQPROC
CSQ4CCB	independent	IBM MQ object definitions	SCSQPROC
CSQ4CVB1	COBOL	Source for user-interface program	SCSQCOBS
CSQ4CVB2	COBOL	Source for credit application manager	SCSQCOBS
CSQ4CVB3	COBOL	Source for checking-account program	SCSQCOBS

Table 182. CICS Credit Check sample (continued)			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4CVB4	COBOL	Source for distribution program	SCSQCOBS
CSQ4CVB5	COBOL	Source for agency-query program	SCSQCOBS
CSQ4CCB1	C	Source for user-interface program	SCSQC37S
CSQ4CCB2	C	Source for credit application manager	SCSQC37S
CSQ4CCB3	C	Source for checking-account program	SCSQC37S
CSQ4CCB4	C	Source for distribution program	SCSQC37S
CSQ4CCB5	C	Source for agency-query program	SCSQC37S
CSQ4CB0	C	Include file	SCSQC370
CSQ4CBMS	C	BMS screen definition source	SCSQMAPS
CSQ4VBMS	COBOL	BMS screen definition source	SCSQMAPS
CSQ4VB0	COBOL	Data definition	SCSQCOBC
CSQ4VB1	COBOL	Data definition	SCSQCOBC
CSQ4VB2	COBOL	Data definition	SCSQCOBC
CSQ4VB3	COBOL	Data definition	SCSQCOBC
CSQ4VB4	COBOL	Data definition	SCSQCOBC
CSQ4VB5	COBOL	Data definition	SCSQCOBC
CSQ4VB6	COBOL	Data definition	SCSQCOBC
CSQ4VB7	COBOL	Data definition	SCSQCOBC
CSQ4VB8	COBOL	Data definition	SCSQCOBC
CSQ4BAQ	independent	Source for VSAM data set	SCSQPROC
CSQ4FILE	independent	JCL to build VSAM data set used by CSQ4CVB3	SCSQPROC
CSQ4S100	independent	CICS system definition data set	SCSQPROC

Table 183. CICS Asynchronous Consumption and Publish/Subscribe samples		
Member name	**Description**	**Source file supplied in library**
CSQ4CVCN	Source for Simple Message Consumption program	SCSQCOBS
CSQ4CVCT	Source for Control Message Consumption program	SCSQCOBS
CSQ4CVEV	Source for Event Handler program	SCSQCOBS
CSQ4CVPT	Source for Message Put Client program	SCSQCOBS
CSQ4CVRG	Source for Registration Client program	SCSQCOBS

Table 183. CICS Asynchronous Consumption and Publish/Subscribe samples (continued)		
Member name	**Description**	**Source file supplied in library**
CSQ4S100	CICS System Definition data set	SCSQPROC

z/OS *Preparing the sample application for the IMS environment on z/OS*

Part of the Credit Check sample application can run in the IMS environment.

To prepare this part of the application to run with the CICS sample, first perform the steps described in “Preparing the sample applications for the CICS environment on z/OS” on page 1187.

Then perform the following steps:

1. Perform the same steps that you would when building any IMS IBM MQ for z/OS application. These steps are listed in “Building IMS (BMP or MPP) applications” on page 1040. The library members to use are listed in Table 184 on page 1191.
2. Identify the application program and database to IMS. Samples are provided with PSBGEN, DBDGEN, ACB definition, IMSGEN, and IMSDALOC statements to enable this.
3. Load the database CSQ4CA by tailoring and running the sample JCL provided for this purpose (CSQ4ILDB). This JCL loads the database with data from the file CSQ4BAQ. Update the IMS control region with a DD statement for the database CSQ4CA.
4. Start the checking-account program as a batch message processing (BMP) program by tailoring and running the sample JCL provided for this purpose. This JCL starts a batch-oriented BMP program. To run the program as a message-oriented BMP program, remove the comment characters from the line in the JCL that contains the IN= statement.

z/OS *Names of the sample IMS application on z/OS*

This information provides a table with the list of the sources and JCLs that are supplied for the Credit Check sample IMS application.

Table 184. Source and JCL for the Credit Check IMS sample (C only)		
Member name	**Description**	**Supplied in library**
CSQ4CVB	IBM MQ object definitions	SCSQPROC
CSQ4ICB3	Source for checking-account program	SCSQC37S
CSQ4ICBL	Source for loading the checking-account database	SCSQC37S
CSQ4CBI	Data definition	SCSQC370
CSQ4PSBL	PSBGEN JCL for database-load program	SCSQPROC
CSQ4PSB3	PSBGEN JCL for checking-account program	SCSQPROC
CSQ4DBDS	DBDGEN JCL for database CSQ4CA	SCSQPROC
CSQ4GIMS	IMSGEN macro definitions for CSQ4IVB3 and CSQ4CA	SCSQPROC
CSQ4ACBG	Application control block (ACB) definition for CSQ4IVB3	SCSQPROC
CSQ4BAQ	Source for database	SCSQPROC

Table 184. Source and JCL for the Credit Check IMS sample (C only) (continued)		
Member name	**Description**	**Supplied in library**
CSQ4ILDB	Sample run JCL for database-load job	SCSQPROC
CSQ4ICBR	Sample run JCL for checking-account program	SCSQPROC
CSQ4DYNA	IMSDALOC macro definitions for database	SCSQPROC

z/OS *The Put samples on z/OS*

The Put sample programs put messages on a queue using the MQPUT call.

The source programs are supplied in C and COBOL in the batch and CICS environments (see Table 172 on page 1184 and Table 179 on page 1188).

Design of the Put sample

The flow through the program logic is:

1. Connect to the queue manager using the MQCONN call. If this call fails, print the completion and reason codes and stop processing.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQCONN call; if you do, it returns DEF_HCONN. You can use the connection handle MQHC_DEF_HCONN for the MQI calls that follow.
2. Open the queue using the MQOPEN call with the MQOO_OUTPUT option. On input to this call, the program uses the connection handle that is returned in step “1” on page 1194. For the object descriptor structure (MQOD), it uses the default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.
3. Create a loop within the program issuing MQPUT calls until the required number of messages are put on the queue. If an MQPUT call fails, the loop is abandoned early, no further MQPUT calls are attempted, and the completion and reason codes are returned.
4. Close the queue using the MQCLOSE call with the object handle returned in step “2” on page 1194. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “1” on page 1194. If this call fails, print the completion and reason codes.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQDISC call.

z/OS *The Put samples for the batch environment on z/OS*

Use this topic when considering Put samples for the batch environment.

To run the samples, edit and run the sample JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1183.

The programs take the following parameters in an EXEC PARM, separated by spaces in C and commas in COBOL:

1. The name of the queue manager (4 characters)
2. The name of the target queue (48 characters)
3. The number of messages (up to 4 digits)
4. The padding character to write in the message (1 character)
5. The number of characters to write in the message (up to 4 digits)
6. The persistence of the message (1 character: P for persistent or N for nonpersistent)

If you enter any of the these parameters wrongly, you receive appropriate error messages.

Any messages from the samples are written to the SYSPRINT data set.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. However, these differences are minimized if you use the layout of the parameters shown in the sample run JCL, CSQ4BCJR, and CSQ4BVJR. None of the differences relate to the MQI.
- CSQ4BCK1 allows you to enter more than four digits for the number of messages sent and the length of the messages.
- For the two numeric fields, enter any digit in the range 1 through 9999. The value that you enter should be a positive number. For example, to put a single message, you can enter 1, 01, 001, or 0001 as the value. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program sends a 1-byte message, but the C program receives an error.
- For both programs, CSQ4BCK1 and CSQ4BVK1, you must enter P in the persistence parameter, ++PER++, if you want the message to be persistent. If you fail to do so, the message will be nonpersistent.

z/OS *The Put samples for the CICS environment on z/OS*

Use this topic when considering Put samples for the CICS environment.

The transactions take the following parameters separated by commas:

1. The number of messages (up to 4 digits)
2. The padding character to write in the message (1 character)
3. The number of characters to write in the message (up to 4 digits)
4. The persistence of the message (1 character: P for persistent or N for nonpersistent)
5. The name of the target queue (48 characters)

If you enter any of these parameters wrongly, you receive appropriate error messages.

For the COBOL sample, invoke the Put sample in the CICS environment by entering:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

For the C sample, invoke the Put sample in the CICS environment by entering:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Any messages from the samples are displayed on the screen.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. None of the differences relate to the MQI.
- If you enter a queue name that is longer than 48 characters, its length is truncated to the maximum of 48 characters but no error message is returned.
- Before entering the transaction, press the CLEAR key.
- For the two numeric fields, enter any number in the range 1 through 9999. The value that you enter should be a positive number. For example, to put a single message, you can enter the value 1, 01, 001, or 0001. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program sends a 1-byte message, and the C program abends with an error from malloc().
- For both programs, CSQ4CCK1 and CSQ4CVK1, enter P in the persistence parameter if you want the message to be persistent. For non-persistent messages, enter N in the persistence parameter. If you enter any other value you receive an error message.

- The messages are put in syncpoint because default values are used for all parameters except those set during program invocation.

z/OS *The Get samples on z/OS*

The Get sample programs get messages from a queue using the MQGET call.

The source programs are supplied in C and COBOL in the batch and CICS environments (see Table 172 on page 1184 and Table 179 on page 1188).

z/OS *Design of the Get sample on z/OS*

Learn about the design of the Get sample, and some usage notes to consider.

The flow through the program logic is:

1. Connect to the queue manager using the MQCONN call. If this call fails, print the completion and reason codes and stop processing.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQCONN call; if you do, it returns DEF_HCONN. You can use the connection handle MQHC_DEF_HCONN for the MQI calls that follow.
2. Open the queue using the MQOPEN call with the MQOO_INPUT_SHARED and MQOO_BROWSE options. On input to this call, the program uses the connection handle that is returned in step “1” on page 1194. For the object descriptor structure (MQOD), it uses the default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.
3. Create a loop within the program issuing MQGET calls until the required number of messages are retrieved from the queue. If an MQGET call fails, the loop is abandoned early, no further MQGET calls are attempted, and the completion and reason codes are returned. The following options are specified on the MQGET call:

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST and MQGMO_BROWSE_NEXT

   For a description of these options, see MQGET. For each message, the message number is printed followed by the length of the message and the message data.
4. Close the queue using the MQCLOSE call with the object handle returned in step “2” on page 1194. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “1” on page 1194. If this call fails, print the completion and reason codes.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQDISC call.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. However, these differences are minimized if you use the layout of the parameters shown in the sample run JCL, CSQ4BCJR, and CSQ4BVJR,. None of the differences relate to the MQI.
- CSQ4BCJ1 allows you to enter more than four digits for the number of messages retrieved.
- Messages longer than 64 KB are truncated.
- CSQ4BCJ1 can only correctly display character messages because it only displays until the first NULL (\0) character is displayed.
- For the numeric number-of-messages field, enter any digit in the range 1 through 9999. The value that you enter should be a positive number. For example, to get a single message, you can enter 1, 01, 001, or 0001 as the value. If you enter nonnumeric or negative values, you might receive an error.

For example, if you enter -1, the COBOL program retrieves one message, but the C program does not retrieve any messages.

- For both programs, CSQ4BCJ1 and CSQ4BVJ1, enter B in the get parameter, ++GET++, if you want to browse the messages.
- For both programs, CSQ4BCJ1 and CSQ4BVJ1, enter S in the syncpoint parameter, ++SYNC++, for messages to be retrieved in syncpoint.

z/OS *The Get samples for the batch environment on z/OS*

To run the samples, edit and run the sample JCL, as described in "Preparing and running sample applications for the batch environment on z/OS" on page 1183.

The programs take the following parameters in an EXEC PARM, separated by spaces in C and commas in COBOL:

1. The name of the queue manager (4 characters)
2. The name of the target queue (48 characters)
3. The number of messages to get (up to 4 digits)
4. The browse/get message option (1 character: B to browse or D to destructively get the messages)
5. The syncpoint control (1 character: S for syncpoint or N for no syncpoint)

If you enter any of these parameters incorrectly, you receive appropriate error messages.

Output from the samples is written to the SYSPRINT data set:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *The Get samples for the CICS environment on z/OS*

Special considerations for the Get samples for the CICS environment.

The transactions take the following parameters in an EXEC PARM, separated by commas:

1. The number of messages to get (up to four digits)
2. The browse/get message option (one character: B to browse or D to destructively get the messages)
3. The syncpoint control (one character: S for syncpoint or N for no syncpoint)
4. The name of the target queue (48 characters)

If you enter any of these parameters incorrectly, you receive appropriate error messages.

For the COBOL sample, invoke the Get sample in the CICS environment by entering:

```
MVGT,9999,B,S,QUEUE.NAME
```

For the C sample, invoke the Get sample in the CICS environment by entering:

```
MCGT,9999,B,S,QUEUE.NAME
```

When the messages are retrieved from the queue, they are put on a CICS temporary storage queue with the same name as the CICS transaction (for example, MCGT for the C sample).

Here is example output of the Get samples:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE *********************
```

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. None of the differences relate to the MQI.
- If you enter a queue name that is longer than 48 characters, its length is truncated to the maximum of 48 characters but no error message is returned.
- Before entering the transaction, press the CLEAR key.
- CSQ4CCJ1 can only correctly display character messages because it only displays until the first NULL (\0) character is displayed.
- For the numeric field, enter any number in the range 1 through 9999. The value that you enter should be a positive number. For example, to get a single message, you can enter the value 1, 01, 001, or 0001. If you enter a nonnumeric or negative value, you might receive an error.
- Messages longer than 24 526 bytes in C and 9 950 bytes in COBOL are truncated. This is due to the way that the CICS temporary storage queues are used.
- For both programs, CSQ4CCK1 and CSQ4CVK1, enter B in the get parameter if you want to browse the messages, otherwise enter D. This performs destructive MQGET calls. If you enter any other value you receive an error message.
- For both programs, CSQ4CCJ1 and CSQ4CVJ1, enter S in the syncpoint parameter to retrieve messages in syncpoint. If you enter N in the syncpoint parameter, the MQGET calls are issued out of syncpoint. If you enter any other value you receive an error message.

z/OS *The Browse sample on z/OS*

The Browse sample is a batch application that demonstrates how to browse messages on a queue using the MQGET call.

The application steps through all the messages in a queue, printing the first 80 bytes of each one. You could use this application to look at the messages on a queue without changing them.

Source programs and sample run JCL are supplied in the COBOL, assembler, PL/I, and C languages (see Table 173 on page 1184).

To start the application, edit and run the sample run JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1183. You can look at messages on one of your own queues by specifying the name of the queue in the run JCL.

When you run the application (and there are some messages on the queue), the output data set looks this:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
```

```
!8     9 CSQ4STOP
********** END OF REPORT **********
```

If there are no messages on the queue, the data set contains the headings and the `End of report` message only. If an error occurs with any of the MQI calls, the completion and reason codes are added to the output data set.

z/OS *Design of the Browse sample on z/OS*

The Browse sample application uses a single program module; one is provided in each of the supported programming languages.

The flow through the program logic is:

1. Open a print data set and print the title line of the report. Check that the names of the queue manager and queue have been passed from the run JCL. If both names have been passed, print the lines of the report that contain the names. If they have not, print an error message, close the print data set, and stop processing.

   The way that the program tests the parameters it is passed from the JCL depends on the language in which the program is written; for more information, see "Language-dependent design considerations on z/OS" on page 1198.
2. Connect to the queue manager using the MQCONN call. If this call is not successful, print the completion and reason codes, close the print data set, and stop processing.
3. Open the queue using the MQOPEN call with the MQOO_BROWSE option. On input to this call, the program uses the connection handle returned in step "2" on page 1197. For the object descriptor structure (MQOD), it uses the default values for all the fields except the queue name (which was passed in step "1" on page 1197). If this call is not successful, print the completion and reason codes, close the print data set, and stop processing.
4. Browse the first message on the queue, using the MQGET call. On input to this call, the program specifies:
   - The connection and queue handles from steps "2" on page 1197 and "3" on page 1197
   - An MQMD structure with all fields set to their initial values
   - Two options:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - A buffer of size 80 bytes to hold the data copied from the message

   The MQGMO_ACCEPT_TRUNCATED_MSG option allows the call to complete even if the message is longer than the 80-byte buffer specified in the call. If the message is longer than the buffer, the message is truncated to fit the buffer, and the completion and reason codes are set to show this. The sample was designed so that messages are truncated to 80 characters to make the report easy to read. The buffer size is set by a `DEFINE` statement, so you can easily change it if you want to.
5. Perform the following loop until the MQGET call fails:
   a. Print a line of the report showing:
      - The sequence number of the message (this is a count of the browse operations).
      - The true length of the message (not the truncated length). This value is returned in the `DataLength` field of the MQGET call.
      - The first 80 bytes of the message data.
   b. Reset the `MsqId` and `CorrelId` fields of the MQMD structure to nulls
   c. Browse the next message, using the MQGET call with these two options:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG

6. If the MQGET call fails, test the reason code to see if the call has failed because the browse cursor has got to the end of the queue. In this case, print the `End of report` message and go to step “7” on page 1198 ; otherwise, print the completion and reason codes, close the print data set, and stop processing.
7. Close the queue using the MQCLOSE call with the object handle returned in step “3” on page 1197.
8. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “2” on page 1197.
9. Close the print data set and stop processing.

z/OS *Language-dependent design considerations on z/OS*

Source modules are provided for the Browse sample in four programming languages.

There are two main differences between the source modules:

- When testing the parameters passed from the run JCL, the COBOL, PL/I, and assembler-language modules search for the comma character (,). If the JCL passes `PARM=(,LOCALQ1)`, the application attempts to open queue LOCALQ1 on the default queue manager. If there is no name after the comma (or no comma), the application returns an error. The C module does not search for the comma character. If the JCL passes a single parameter (for example, `PARM=('LOCALQ1')`), the C module uses this as a queue name on the default queue manager.
- To keep the assembler-language module simple, it uses the date format *yy/ddd* (for example, 05/116) when it creates the print report. The other modules use the calendar date in *mm/dd/yy* format.

z/OS *The Print Message sample on z/OS*

The Print Message sample is a batch application that demonstrates how to remove all the messages from a queue using the MQGET call.

The Print Message sample uses three parameters:

1. The name of the queue manager
2. The name of the source queue
3. An optional parameter for properties

It also prints, for each message, the fields of the message descriptor, followed by the message data. The program prints the data both in hexadecimal and as characters (if they are printable). If a character is not printable, the program replaces it with a period character (.). You can use the program when diagnosing problems with an application that is putting messages on a queue.

Permissible values for the property parameter are:

Table 185. Valores permitidos para o parâmetro da propriedade

Value	Comportamento
0	Comportamento padrão.. As propriedades que são entregues ao aplicativo dependem do atributo de fila **PropertyControl** do qual a mensagem é recuperada.

Table 185. Valores permitidos para o parâmetro da propriedade (continued)

Value	Comportamento

1	Um identificador de mensagens é criado e usado com o MQGET. As propriedades da mensagem, exceto aquelas contidas no descritor de mensagens (ou extensão), são exibidas de forma semelhante para o descritor de mensagens. Por exemplo:
	`****Message properties****`
	`  property name: property value`
	Ou se nenhuma propriedade estiver disponível:
	`****Message properties****`
	`  None`
	Valores numéricos são exibidos usando o `printf`, os valores de sequência são colocados entre aspas simples e as sequências de bytes são marcadas com X e colocadas entre aspas simples, como para o descritor de mensagens.

2	MQGMO_NO_PROPERTIES é especificado, de forma que apenas as propriedades do descritor de mensagens sejam retornadas.
3	MQGMO_PROPERTIES_FORCE_MQRFH2 é especificado, de forma que todas as propriedades sejam retornadas nos dados da mensagem.
4	MQGMO_PROPERTIES_COMPATIBILITY é especificado para que todas as propriedades possam ser retornadas, dependendo de uma propriedade IBM MQ estar incluída, caso contrário, as propriedades serão descartadas.

You can change the application so that it browses the messages, rather than removing them from the queue. To do this, compile with the option of -DBROWSE, to define the BROWSE macro, as indicated in “Design of the Print Message sample on z/OS” on page 1200. Executable code is provided for you in the SCSQLOAD library. Module CSQ4BCG0 is built with -DBROWSE; module CSQ4BCG1 destructively reads the queue.

The application has a single source program, which is written in the C language. Sample run JCL code is also supplied (see Table 174 on page 1184).

To start the application, edit and run the sample run JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1183. When you run the application (and there are some messages on the queue), the output data set looks like that in Figure 139 on page 1200.

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR  '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '  '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

Figure 139. Example of a report from the Print Message sample application

z/OS *Design of the Print Message sample on z/OS*

The Print Message sample application uses a single program written in the C language.

The flow through the program logic is:

1. Check that the names of the queue manager and queue have been passed from the run JCL. If they have not, print an error message and stop processing.
2. Connect to the queue manager using the MQCONN call. If this call is not successful, print the completion and reason codes and stop processing; otherwise print the name of the queue manager.
3. Open the queue using the MQOPEN call with the MQOO_INPUT_SHARED option.

   Note: If you want the application to browse the messages rather than remove them from the queue, compile the sample with -DBROWSE, or, add `#define BROWSE` at the beginning of the source. When you do this, the macro preprocessor adds the line in the program that selects the MQOO_BROWSE option in the compilation.

On input to this call, the program uses the connection handle returned in step "2" on page 1200. For the object descriptor structure (MQOD), it uses the default values for all the fields except the queue name (which was passed in step "1" on page 1200). If this call is not successful, print the completion and reason codes and stop processing; otherwise, print the name of the queue.

4. If you use a message handle to obtain the message properties use MQCRTMH to create such a handle for use with subsequent MQGET calls. If this call is not successful, print the completion and reason codes and stop processing.
5. Set the get message options to reflect the request action for any message properties.
6. Perform the following loop until the MQGET call fails:
   a. Initialize the buffer to blanks so that the message data does not get corrupted by any data already in the buffer.
   b. Set the `MsgId` and `CorrelId` fields of the MQMD structure to nulls so that the MQGET call selects the first message from the queue.
   c. Get a message from the queue, using the MQGET call. On input to this call, the program specifies:
      - The connection and object handles from steps "2" on page 1200 and "3" on page 1200.
      - An MQMD structure with all fields set to their initial values. (`MsgId` and `CorrelId` are reset to nulls for each MQGET call.)
      - The option MQGMO_NO_WAIT.

        Note: If you want the application to browse the messages rather than remove them from the queue, compile the sample with -DBROWSE, or, add `#define BROWSE` at the beginning of the source. When you do this, the macro preprocessor adds the line in the program that selects the MQGMO_BROWSE_NEXT option to the compilation. When this option is used on a call against a queue for which no browse cursor has previously been used with the current object handle, the browse cursor is positioned logically before the first message.
      - A buffer of size 64KB to hold the data copied from the message.
   d. Call the printMD subroutine. This prints the name of each field in the message descriptor, followed by its contents.
   e. If you created a message handle in step "4" on page 1201 call the printProperties subroutine to display any message properties.
   f. Print the length of the message, followed by the message data. Each line of message data is in this format:
      - Relative position (in hexadecimal) of this part of the data
      - 16 bytes of hexadecimal data
      - The same 16 bytes of data in character format, if it is printable (nonprintable characters are replaced by periods)
7. If the MQGET call fails, test the reason code to see if the call failed because there are no more messages on the queue. In this case, print the message: No more messages; otherwise, print the completion and reason codes. In both cases, go to step "9" on page 1201.

   Note: The MQGET call fails if it finds a message that has more than 64KB of data. To change the program to handle larger messages, you could do one of the following:
   - Add the MQGMO_ACCEPT_TRUNCATED_MSG option to the MQGET call, so that the call gets the first 64KB of data and discards the remainder
   - Make the program leave the message on the queue when it finds one with this amount of data
   - Increase the size of the buffer
8. If you created a message handle in step "4" on page 1201 call MQDLTMH to delete it.
9. Close the queue using the MQCLOSE call with the object handle returned in step "3" on page 1200.
10. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step "2" on page 1200.

z/OS *The Queue Attributes sample on z/OS*

The Queue Attributes sample is a conversational-mode CICS application that demonstrates the use of the MQINQ and MQSET calls.

It shows how to inquire about the values of the **InhibitPut** and **InhibitGet** attributes of queues, and how to change them so that programs cannot put messages on, or get messages from, a queue. You might want to *lock* a queue in this way when you are testing a program.

To prevent accidental interference with your own queues, this sample works only on a queue object that has the characters CSQ4SAMP in the first eight bytes of its name. However, the source code includes comments to show you how to remove this restriction.

Source programs are supplied in the COBOL, assembler, and C languages (see Table 180 on page 1188).

The assembler-language version of the sample uses reenterable code. To do this, you will notice that the code for each MQI call in that version of the sample includes the MF keyword; for example:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(The VL keyword means that you can use the CICS Execution Diagnostic Facility (CEDF) supplied transaction for debugging the program.) For more information about writing reenterable programs, see Coding in System/390 assembler language.

To start the application, start your CICS system and use the following CICS transactions:

- For COBOL, MVC1
- For assembler language, MAC1
- For C, MCC1

You can change the name of any of these transactions by changing the CSD data set mentioned in step 3.

Design of the sample

When you start the sample, it displays a screen map that has fields for:

- Name of the queue
- User request (valid actions are: inquire, allow, or inhibit)
- Current status of put operations for the queue
- Current status of get operations for the queue

The first two fields are for user input. The last two fields are filled by the application: they show the word `INHIBITED` or the word `ALLOWED`.

The application validates the values that you enter in the first two fields. It checks that the queue name starts with the characters CSQ4SAMP and that you entered one of the three valid requests in the Action field. The application converts all your input to uppercase, so you cannot use any queues with names that contain lowercase characters.

If you enter `inquire` in the **Action** field, the flow through the program logic is:

1. Open the queue using the MQOPEN call with the MQOO_INQUIRE option
2. Call MQINQ using the selectors MQIA_INHIBIT_GET and MQIA_INHIBIT_PUT
3. Close the queue using the MQCLOSE call
4. Analyze the attributes that are returned in the **IntAttrs** parameter of the MQINQ call and move the words INHIBITED or ALLOWED, as appropriate, to the relevant screen fields

If you enter `inhibit` in the **Action** field, the flow through the program logic is:

1. Open the queue using the MQOPEN call with the MQOO_SET option
2. Call MQSET using the selectors MQIA_INHIBIT_GET and MQIA_INHIBIT_PUT, and with the values MQQA_GET_INHIBITED and MQQA_PUT_INHIBITED in the **IntAttrs** parameter

3. Close the queue using the MQCLOSE call
4. Move the word INHIBITED to the relevant screen fields

If you enter `allow` in the **Action** field, the application performs similar processing to that for an inhibit request. The only differences are the settings of the attributes and the words displayed on the screen.

When the application opens the queue, it uses the default connection handle to the queue manager. (CICS establishes a connection to the queue manager when you start your CICS system.) The application can trap the following errors at this stage:

- The application is not connected to the queue manager
- The queue does not exist
- The user is not authorized to access the queue
- The application is not authorized to open the queue

For other MQI errors, the application displays the completion and reason codes.

z/OS *The Mail Manager sample on z/OS*

The Mail Manager sample application is a suite of programs that demonstrates sending and receiving messages, both within a single environment and across different environments. The application is a simple electronic mailing system that allows users to exchange messages, even if they use different queue managers.

The application demonstrates how to create queues using the MQOPEN call and by putting IBM MQ for z/OS commands on the system-command input queue.

Three versions of the application are provided:

- A CICS application written in COBOL
- A TSO application written in COBOL
- A TSO application written in C

z/OS *Preparing the Mail Manager sample on z/OS*

The Mail Manager is provided in versions that run in two environments. The preparation that you must carry out before you run the application depends on the environment that you want to use.

Users can access mail queues and nickname queues from both TSO and CICS so long as their sign-on user IDs are the same on each system.

Before you can send messages to another queue manager, you must set up a message channel to that queue manager. To do this, use the channel control function of IBM MQ, described in Channel control function.

Preparing the sample for the TSO environment

Follow these steps:

1. Prepare the sample as described in “Preparing sample applications for the TSO environment on z/OS” on page 1185.
2. Tailor the CLIST provided for the sample to define:
   - The location of the panels
   - The location of the message file
   - The location of the load modules
   - The name of the queue manager that you want to use with the application

   A separate CLIST is provided for each language version of the sample:
   - For the COBOL version: CSQ4RVD1
   - For the C version: CSQ4RCD1

3. Ensure that the queues used by the application are available on the queue manager. (The queues are defined in CSQ4CVD.)

Note: VS COBOL II does not support multitasking with ISPF. This means that you cannot use the Mail Manager sample application on both sides of a split screen. If you do, the results are unpredictable.

z/OS *Running the Mail Manager sample on z/OS*

To start the sample in the CICS Transaction Server for z/OS environment, run transaction MAIL. If you have not already signed on to CICS, the application prompts you to enter a user ID to which it can send your mail.

When you start the application, it opens your mail queue. If this queue does not exist, the application creates one for you. Mail queues have names of the form CSQ4SAMP.MAILMGR. *userid*, where *userid* depends on the environment:

In TSO
The user's TSO ID

In CICS
The user's CICS sign-on or the user ID entered by the user when prompted when the Mail Manager started

All parts of the queue names that the Mail Manager uses must be uppercase.

The application then presents a menu panel that has options for:

- Read incoming mail
- Send mail
- Create nickname

The menu panel also shows you how many messages are waiting on your mail queue. Each of the menu options displays a further panel:

Read incoming mail
The Mail Manager displays a list of the messages that are on your mail queue. (Only the first 99 messages on the queue are displayed.) For an example of this panel, see Figure 142 on page 1208. When you select a message from this list, the contents of the message are displayed (see Figure 143 on page 1209).

Send mail
A panel prompts you to enter:

- The name of the user to whom you want to send a message
- The name of the queue manager that owns their mail queue
- The text of your message

In the user name field, you can enter either a user ID or a nickname that you created using the Mail Manager. You can leave the queue manager name field blank if the user's mail queue is owned by the same queue manager that you are using, and you must leave it blank if you entered a nickname in the user name field:

- If you specify only a user name, the program first assumes that the name is a nickname, and sends the message to the object defined by that name. If there is no such nickname, the program attempts to send the message to a local queue of that name.
- If you specify both a user name and a queue manager name, the program sends the message to the mail queue that is defined by those two names.

For example, if you want to send a message to user JONESM on remote queue manager QM12, you could send them a message in either of two ways:

- Use both fields to specify user JONESM at queue manager QM12.
- Define a nickname (for example, MARY) for that user and send them a message by putting MARY in the user name field and nothing in the queue manager name field.

Create nickname

You can define an easy-to-remember name that you can use when you send a message to another user who you contact frequently. You are prompted to enter the user ID of the other user and the name of the queue manager that owns their mail queue.

Nicknames are queues that have names of the form CSQ4SAMP.MAILMGR. *userid.nickname*, where *userid* is your own user ID and *nickname* is the nickname that you want to use. With names structured in this way, users can each have their own set of nicknames.

The type of queue that the program creates depends on how you complete the fields of the Create Nickname panel:

- If you specify only a user name, or the queue manager name is the same as that of the queue manager to which the Mail Manager is connected, the program creates an alias queue.
- If you specify both a user name and a queue manager name (and the queue manager is not the one to which the Mail Manager is connected), the program creates a local definition of a remote queue. The program does not check the existence of the queue to which this definition resolves, or even that the remote queue manager exists.

For example, if your own user ID is SMITHK and you create a nickname called MARY for user JONESM (who uses the remote queue manager QM12), the nickname program creates a local definition of a remote queue named CSQ4SAMP.MAILMGR.SMITHK.MARY. This definition resolves to Mary's mail queue, which is CSQ4SAMP.MAILMGR.JONESM at queue manager QM12. If you are using queue manager QM12 yourself, the program instead creates an alias queue of the same name (CSQ4SAMP.MAILMGR.SMITHK.MARY).

The C version of the TSO application makes greater use of ISPF's message-handling capabilities than does the COBOL version. You might notice that different error messages are displayed by the C and COBOL versions.

z/OS *Design of the Mail Manager sample on z/OS*

The following sections describe each of the programs that make up the Mail Manager sample application.

The relationships between the programs and the panels that the application uses is shown in Figure 140 on page 1206 for the TSO version, and Figure 141 on page 1207 for the CICS Transaction Server for z/OS version.

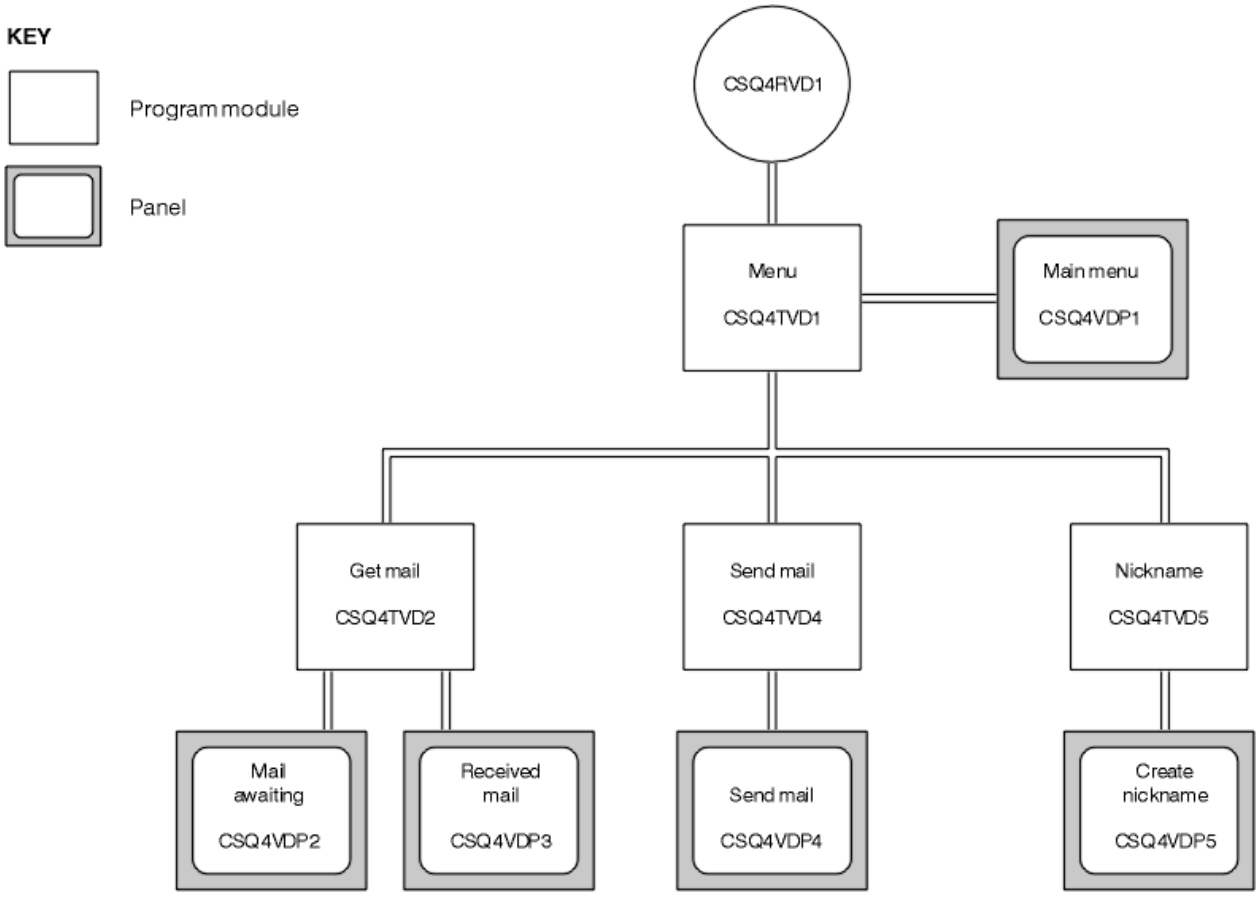


Figure 140. Programs and panels for the TSO versions of the Mail Manager

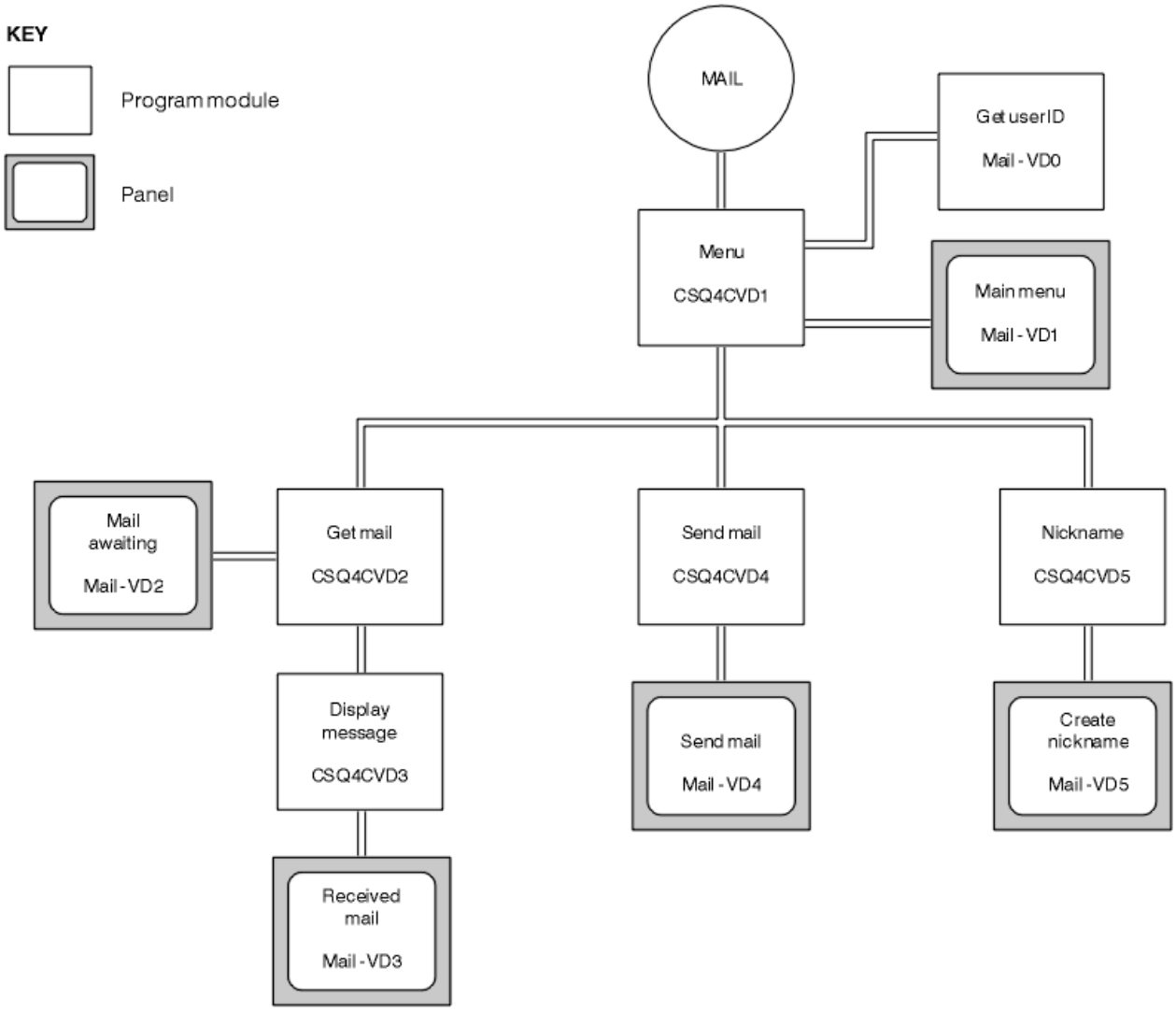


Figure 141. Programs and panels for the CICS version of the Mail Manager

z/OS *Menu program on z/OS*

In the TSO environment, the menu program is invoked by the CLIST. In the CICS environment, the program is invoked by transaction MAIL.

The menu program (CSQ4TVD1 for TSO, CSQ4CVD1 for CICS) is the initial program in the suite. It displays the menu (CSQ4VDP1 for TSO, VD1 for CICS) and invokes the other programs when they are selected from the menu.

The program first obtains the user's ID:

- In the CICS version of the program, if the user has signed on to CICS, the user ID is obtained by using the CICS command ASSIGN USERID. If the user has not signed on, the program displays the sign on panel (CSQ4VD0) to prompt the user to enter a user ID. There is no security processing within this program; the user can give any user ID.
- In the TSO version, the user's ID is obtained from TSO in the CLIST. It is passed to the menu program as a variable in the ISPF shared pool.

After the program has obtained the user ID, it checks to ensure that the user has a mail queue (CSQ4SAMP.MAILMGR. *userid*). If a mail queue does not exist, the program creates one by putting a message on the system-command input queue. The message contains the IBM MQ for z/OS command DEFINE QLOCAL. The object definition that this command uses sets the maximum depth of the queue to 9999 messages.

The program also creates a temporary dynamic queue to handle replies from the system-command input queue. To do this, the program uses the MQOPEN call, specifying the SYSTEM.DEFAULT.MODEL.QUEUE as the template for the dynamic queue. The queue manager creates the temporary dynamic queue with a name that has the prefix CSQ4SAMP; the remainder of the name is generated by the queue manager.

The program then opens the user's mail queue and finds the number of messages on the queue by inquiring about the current depth of the queue. To do this, the program uses the MQINQ call, specifying the MQIA_CURRENT_Q_DEPTH selector.

The program then performs a loop that displays the menu and processes the selection that the user makes. The loop is stopped when the user presses the PF3 key. When a valid selection is made, the appropriate program is started; otherwise an error message is displayed.

z/OS *Get-mail and display-message programs on z/OS*

In the TSO versions of the application, the get-mail and display-message functions are performed by the same program (CSQ4TVD2). In the CICS version of the application, these functions are performed by separate programs (CSQ4CVD2 and CSQ4CVD3).

The Mail Awaiting panel (CSQ4VDP2 for TSO, VD2 for CICS ; see Figure 142 on page 1208 for an example) shows all the messages that are on the user's mail queue. To create this list, the program uses the MQGET call to browse all the messages on the queue, saving information about each one. In addition to the information displayed, the program records the `MsgId` and `CorrelId` of each message.

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

Figure 142. Example of a panel showing a list of waiting messages

From the Mail Awaiting panel the user can select one message and display the contents of the message (see Figure 143 on page 1209 for an example). The program uses the MQGET call to remove this message from the queue, using the `MsgId` and `CorrelId` that the program noted when it browsed all the messages. This MQGET call is performed using the MQGMO_SYNCPOINT option. The program displays the contents of the message, then declares a syncpoint: this commits the MQGET call, so the message now no longer exists.

```
--------------------- IBM MQ for z/OS Sample Programs ---------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
----------------------------------- Message ------------------------------
| HELLO FROM JOHNJ                                |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
'-------------------------------------------------------------------------'
```

Figure 143. Example of a panel showing the contents of a message

An obvious extension to the function provided by the Mail Manager is to give the user the option to leave the message on the queue after viewing its contents. To do this, you would have to back out the MQGET call that removes the message from the queue, after displaying the message.

z/OS *Send-mail program on z/OS*

When the user has completed the Send Mail panel (CSQ4VDP4 for TSO, VD4 for CICS), the send-mail program (CSQ4TVD4 for TSO, CSQ4CVD4 for CICS) puts the message on the receiver's mail queue.

To do this, the program uses the MQPUT1 call. The destination of the message depends on how the user has filled the fields in the Send Mail panel:

- If the user has specified only a user name, the program first assumes that the name is a nickname, and sends the message to the object defined by that name. If there is no such nickname, the program attempts to send the message to a local queue of that name.
- If the user has specified both a user name and a queue manager name, the program sends the message to the mail queue that is defined by those two names.

The program does not accept blank messages, and it removes leading blanks from each line of the message text.

If the MQPUT1 call is successful, the program displays a message that shows that the user name and queue manager name to which the message was put. If the call is unsuccessful, the program checks specifically for the reason codes that indicate the queue or the queue manager do not exist; these are MQRC_UNKNOWN_OBJECT_NAME and MQRC_UNKNOWN_OBJECT_Q_MGR. The program displays its own error message for each of these errors; for other errors, the program displays the completion and reason codes returned by the call.

z/OS *Nickname program on z/OS*

When the user defines a nickname, the nickname program (CSQ4TVD5 for TSO, CSQ4CVD5 for CICS) creates a queue that has the nickname as part of its name.

The program does this by putting a message on the system-command input queue. The message contains the IBM MQ for z/OS command DEFINE QALIAS or DEFINE QREMOTE. The type of queue that the program creates depends on how the user has filled the fields of the Create Nickname panel (CSQ4VDP5 for TSO, VD5 for CICS):

- If the user has specified only a user name, or the queue manager name is the same as that of the queue manager to which the Mail Manager is connected, the program creates an alias queue.

- If the user has specified both a user name and a queue manager name (and the queue manager is not the one to which the Mail Manager is connected), the program creates a local definition of a remote queue. The program does not check the existence of the queue to which this definition resolves, or even that the remote queue manager exists.

The program also creates a temporary dynamic queue to handle replies from the system-command input queue.

If the queue manager cannot create the nickname queue for a reason that the program expects (for example, the queue already exists), the program displays its own error message. If the queue manager cannot create the queue for a reason that the program does not expect, the program displays up to two of the error messages that are returned to the program by the command server.

Note: For each nickname, the nickname program creates only an alias queue or a local definition of a remote queue. The local queues to which these queue names resolve are created only when the user ID that is contained in the nickname is used to start the Mail Manager application.

z/OS *The Credit Check sample on z/OS*

The Credit Check sample application is a suite of programs that demonstrates how to use many of the features provided by IBM MQ for z/OS. It shows how the many component programs of an application can pass messages to each other using message queuing techniques.

The sample can run as a stand-alone CICS application. However, to demonstrate how to design a message queuing application that uses the facilities provided by both the CICS and IMS environments, one module is also supplied as an IMS batch message processing program. This extension to the sample is described in “The IMS extension to the Credit Check sample on z/OS” on page 1220.

You can also run the sample on more than one queue manager, and send messages between each instance of the application. To do so, see “The Credit Check sample with multiple queue managers on z/OS” on page 1219.

The CICS programs are delivered in C and COBOL. The single IMS program is delivered only in C. The supplied data sets are shown in Table 182 on page 1189 and Table 184 on page 1191.

The application demonstrates a method of assessing the risk when bank customers ask for loans. The application shows how a bank could work in two ways to process loan requests:

- When dealing directly with a customer, bank staff want immediate access to account and credit-risk information.
- When dealing with written applications, bank staff can submit a series of requests for account and credit-risk information, and deal with the replies at a later time.

The financial and security details in the application have been kept simple so that the message queuing techniques are clear.

z/OS *Preparing and running the Credit Check sample on z/OS*

To prepare and run the Credit Check sample, perform the following steps:

1. Create the VSAM data set that holds information about some example accounts. Do this by editing and running the JCL supplied in data set CSQ4FILE.
2. Perform the steps in “Preparing the sample applications for the CICS environment on z/OS” on page 1187. (The additional steps that you must perform if you want to use the IMS extension to the sample are described in “The IMS extension to the Credit Check sample on z/OS” on page 1220.)
3. Start the CKTI trigger monitor (supplied with IBM MQ for z/OS) against queue CSQ4SAMP.INITIATION.QUEUE, using the CICS transaction CKQC.
4. To start the application, start your CICS system and use the transaction MVB1.
5. Select **Immediate** or **Batch** inquiry from the first panel.

   The immediate and batch inquiry panels are similar; Figure 144 on page 1211 shows the Immediate Inquiry panel.

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . ______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : _____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

Figure 144. Immediate Inquiry panel for the Credit Check sample application

6. Enter an account number and loan amount in the appropriate fields. See "Entering information in the inquiry panels" on page 1211 for guidance on what information to enter in these fields.

Entering information in the inquiry panels

The Credit Check sample application checks that the data you enter in the **Amount requested** field of the inquiry panels is in the form of integers.

If you enter one of the following account numbers, the application finds the appropriate account name, average account balance, and credit worthiness index in the VSAM data set CSQ4BAQ:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

You can enter any, or no, information in the other fields. The application retains any information that you enter and returns the same information in the reports that it generates.

z/OS *Design of the Credit Check sample on z/OS*

This section describes the design of each of the programs that make up the Credit Check sample application.

For more information about of some of the techniques that were considered during the design of the application, see "Design considerations for the Credit check sample on z/OS" on page 1217.

Figure 145 on page 1212 shows the programs that make up the application, and also the queues that these programs serve. In this figure, the prefix CSQ4SAMP has been omitted from all the queue names to make the figure easier to understand.

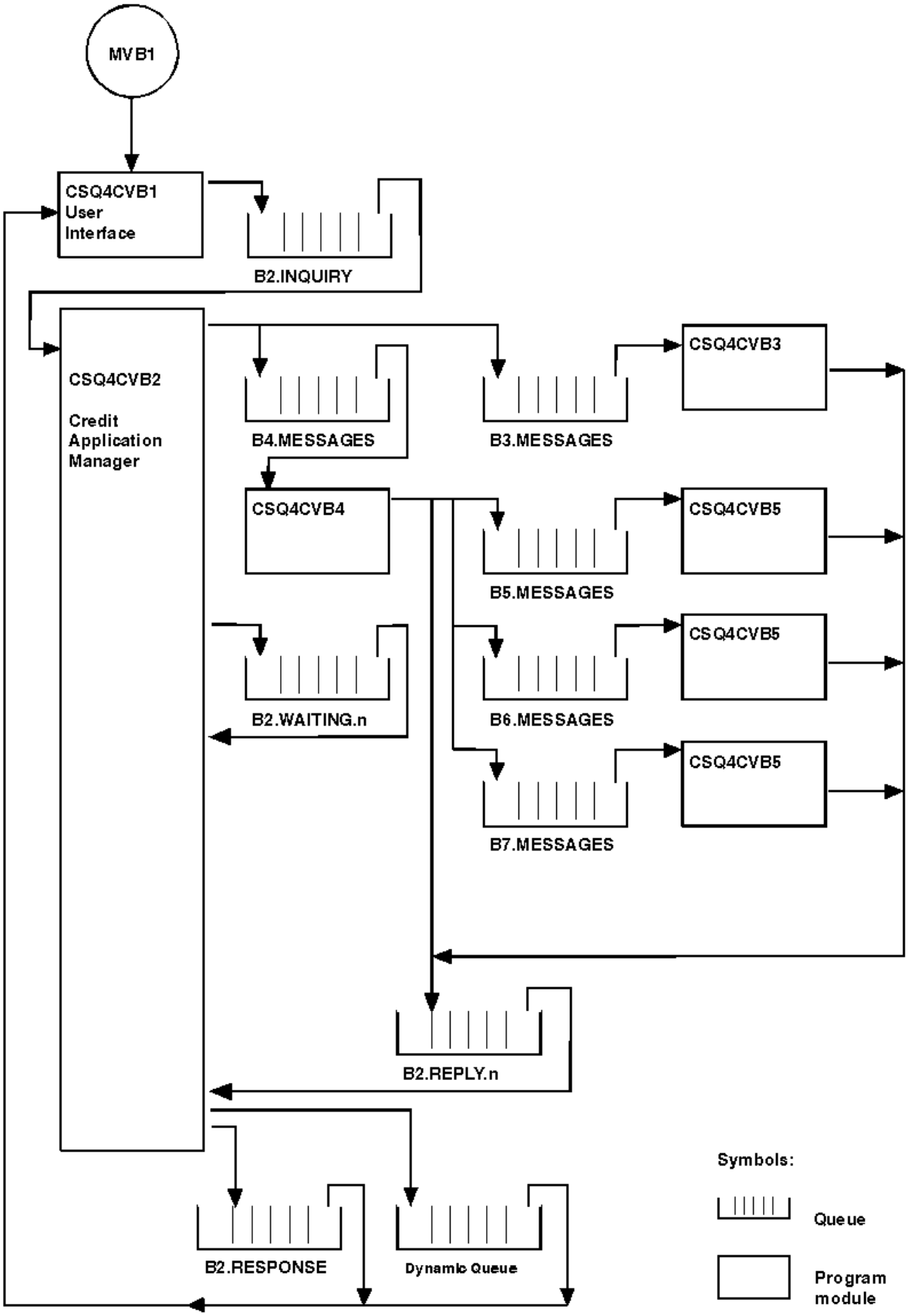


Figure 145. Programs and queues for the Credit Check sample application (COBOL programs only)

z/OS *User interface program (CSQ4CVB1) on z/OS*

When you start the conversational-mode CICS transaction MVB1, this starts the user interface program for the application.

This program puts inquiry messages on queue CSQ4SAMP.B2.INQUIRY and gets replies to those inquiries from a reply-to queue that it specifies when it makes the inquiry. From the user interface you can submit either immediate or batch inquiries:

- For immediate inquiries, the program creates a temporary dynamic queue that it uses as a reply-to queue. This means that each inquiry has its own reply-to queue.
- For batch inquiries, the user-interface program gets replies from the queue CSQ4SAMP.B2.RESPONSE. For simplicity, the program gets replies for all its inquiries from this one reply-to queue. It is easy to see that a bank might want to use a separate reply-to queue for each user of MVB1, so that they could each see replies to only those inquiries that they had initiated.

Important differences between the properties of messages used in the application when in batch and immediate mode are:

- For batch working, the messages have a low priority, so they are processed after any loan requests that are entered in immediate mode. Also, the messages are persistent, so they are recovered if the application or the queue manager has to restart.
- For immediate working, the messages have a high priority, so they are processed before any loan requests that are entered in batch mode. Also, messages are not persistent so they are discarded if the application or the queue manager has to restart.

However, in all cases, the properties of loan request messages are propagated throughout the application. So, for example, all messages that result from a high-priority request will also have a high priority.

z/OS *Credit application manager (CSQ4CVB2) on z/OS*

The Credit Application Manager (CAM) program performs most of the processing for the Credit Check application.

The CAM is started by the CKTI trigger monitor (supplied with IBM MQ for z/OS) when a trigger event occurs on either queue CSQ4SAMP.B2.INQUIRY or queue CSQ4SAMP.B2.REPLY. *n*, where *n* is an integer that identifies one of a set of reply queues. The trigger message contains data that includes the name of the queue on which the trigger event occurred.

The CAM uses queues with names of the form CSQ4SAMP.B2.WAITING.n to store information about inquiries that it is processing. The queues are named so that they are each paired with a reply-to queue; for example, queue CSQ4SAMP.B2.WAITING.3 contains the input data for a particular inquiry, and queue CSQ4SAMP.B2.REPLY.3 contains a set of reply messages (from programs that query databases) all relating to that same inquiry. To understand the reasons behind this design, see “Separate inquiry and reply queues in the CAM” on page 1217.

Startup logic

If the trigger event occurs on queue CSQ4SAMP.B2.INQUIRY, the CAM opens the queue for shared access. It then tries to open each reply queue until a free one is found. If it cannot find a free reply queue, the CAM logs the fact and terminates normally.

If the trigger event occurs on queue CSQ4SAMP.B2.REPLY.n, the CAM opens the queue for exclusive access. If the return code reports that the object is already in use, the CAM terminates normally. If any other error occurs, the CAM logs the error and terminates. The CAM opens the corresponding waiting queue and the inquiry queue, then starts getting and processing messages. From the waiting queue, the CAM recovers details of partially-completed inquiries.

For the sake of simplicity in this sample, the names of the queues used are held in the program. In a business environment, the queue names would probably be held in a file accessed by the program.

Getting a message from the enquiry queue

The CAM first attempts to get a message from the inquiry queue using the MQGET call with the MQGMO_SET_SIGNAL option. If a message is available immediately, the message is processed; if no message is available, a signal is set.

The CAM then attempts to get a message from the reply queue, again using the MQGET call with the same option. If a message is available immediately, the message is processed; otherwise a signal is set.

When both signals are set, the program waits until one of the signals is posted. If a signal is posted to indicate that a message is available, the message is retrieved and processed. If the signal expires or the queue manager is terminating, the program terminates.

Processing the message retrieved by the CAM

A message retrieved by the CAM can be one of four types:

- An inquiry message
- A reply message
- A propagation message
- An unexpected or unwanted message

The CAM processes these messages as described in "Processing the message retrieved by the CAM on z/OS" on page 1214.

Sending an answer

When the CAM has received all the replies it is expecting for an inquiry, it processes the replies and creates a single response message. It consolidates into one message all the data from all reply messages that have the same `CorrelId`. This response is put on the reply-to queue specified in the original loan request. The response message is put within the same unit of work that contains the retrieval of the final reply message. This is to simplify recovery by ensuring that there is never a completed message on queue CSQ4SAMP.B2.WAITING.n.

Recovery of partially-completed inquiries

The CAM copies onto queue CSQ4SAMP.B2.WAITING.n all the messages that it receives. It sets the fields of the message descriptor like this:

- *`Priority`* is determined by the type of message:
  - For request messages, priority = 3
  - For datagrams, priority = 2
  - For reply messages, priority = 1
- *`CorrelId`* is set to the *`MsgId`* of the loan request message
- Other MQMD fields are copied from those of the received message

When an inquiry has been completed, the messages for a specific inquiry are removed from the waiting queue during answer processing. Therefore, at any time, the waiting queue contains all messages relevant to in-progress inquiries. These messages are used to recover details of in-progress inquiries if the program has to restart. The different priorities are set so that inquiry messages are recovered before propagations or reply messages.

z/OS *Processing the message retrieved by the CAM on z/OS*

A message retrieved by the Credit Application Manager (CAM) can be one of four types. The way in which the CAM processes a message depends on its type.

A message retrieved by the CAM can be one of four types:

- An inquiry message

- A reply message
- A propagation message
- An unexpected or unwanted message

The CAM processes these messages as follows:

Inquiry message

Inquiry messages come from the user interface program. It creates an inquiry message for each loan request.

For all loan requests, the CAM requests the average balance of the customer's checking account. It does this by putting a request message on alias queue CSQ4SAMP.B2.OUTPUT.ALIAS. This queue name resolves to queue CSQ4SAMP.B3.MESSAGES, which is processed by the checking-account program, CSQ4CVB3. When the CAM puts a message on this alias queue, it specifies the appropriate CSQ4SAMP.B2.REPLY.n queue for the reply-to queue. An alias queue is used here so that program CSQ4CVB3 can easily be replaced by another program that processes a base queue of a different name. To do this, you redefine the alias queue so that its name resolves to the new queue. Also, you could assign differing access authorities to the alias queue and to the base queue.

If a user requests a loan that is larger than 10000 units, the CAM initiates checks on other databases as well. It does this by putting a request message on queue CSQ4SAMP.B4.MESSAGES, which is processed by the distribution program, CSQ4CVB4. The process serving this queue propagates the message to queues served by programs that have access to other records such as credit card history, savings accounts, and mortgage payments. The data from these programs is returned to the reply-to queue specified in the put operation. Additionally, a propagation message is sent to the reply-to queue by this program to specify how many propagation messages have been sent.

In a business environment, the distribution program would probably reformat the data provided to match the format required by each of the other types of bank account.

Any of the queues referred to can be on a remote system.

For each inquiry message, the CAM initiates an entry in the memory-resident Inquiry Record Table (IRT). This record contains:

- The `MsgId` of the inquiry message
- In the ReplyExp field, the number of responses expected (equal to the number of messages sent)
- In the ReplyRec field, the number of replies received (zero at this stage)
- In the PropsOut field, an indication of whether a propagation message is expected

The CAM copies the inquiry message onto the waiting queue with:

- `Priority` set to 3
- `CorrelId` set to the `MsgId` of the inquiry message
- The other message-descriptor fields set to those of the inquiry message

Propagation message

A propagation message contains the number of queues to which the distribution program has forwarded the inquiry. The message is processed as follows:

1. Add to the ReplyExp field of the appropriate record in the IRT the number of messages sent. This information is in the message.
2. Increment by 1 the ReplyRec field of the record in the IRT.
3. Decrement by 1 the PropsOut field of the record in the IRT.
4. Copy the message onto the waiting queue. The CAM sets the `Priority` to 2 and the other fields of the message descriptor to those of the propagation message.

Reply message

A reply message contains the response to one of the requests to the checking-account program or to one of the agency-query programs. Reply messages are processed as follows:

1. Increment by 1 the ReplyRec field of the record in the IRT.
2. Copy the message onto the waiting queue with `Priority` set to 1 and the other fields of the message descriptor set to those of the reply message.
3. If ReplyRec = ReplyExp, and PropsOut = 0, set the MsgComplete flag.

Other messages

The application does not expect other messages. However, the application might receive messages broadcast by the system, or reply messages with a unknown `CorrelIds`.

The CAM puts these messages on queue CSQ4SAMP.DEAD.QUEUE, where they can be examined. If this put operation fails, the message is lost and the program continues. For more information about the design of this part of the program, see "How the sample handles unexpected messages" on page 1218.

z/OS *Checking-account program (CSQ4CVB3) on z/OS*

The checking-account program is started by a trigger event on queue CSQ4SAMP.B3.MESSAGES. After it has opened the queue, this program gets a message from the queue using the MQGET call with the wait option, and with the wait interval set to 30 seconds.

The program searches VSAM data set CSQ4BAQ for the account number in the loan request message. It retrieves the corresponding account name, average balance, and credit worthiness index, or notes that the account number is not in the data set.

The program then puts a reply message (using the MQPUT1 call) on the reply-to queue named in the loan request message. For this reply message, the program:

- Copies the `CorrelId` of the loan request message
- Uses the MQPMO_PASS_IDENTITY_CONTEXT option

The program continues to get messages from the queue until the wait interval expires.

z/OS *Distribution program (CSQ4CVB4) on z/OS*

The distribution program is started by a trigger event on queue CSQ4SAMP.B4.MESSAGES.

To simulate the distribution of the loan request to other agencies that have access to records such as credit card history, savings accounts, and mortgage payments, the program puts a copy of the same message on all the queues in the namelist CSQ4SAMP.B4.NAMELIST. There are three of these queues, with names of the form CSQ4SAMP.B *n*.MESSAGES, where *n* is 5, 6, or 7. In a business application, the agencies could be at separate locations, so these queues could be remote queues. If you want to modify the sample application to show this, see "The Credit Check sample with multiple queue managers on z/OS" on page 1219.

The distribution program performs the following steps:

1. From the namelist, gets the names of the queues that the program is to use. The program does this by using the MQINQ call to inquire about the attributes of the namelist object.
2. Opens these queues and also CSQ4SAMP.B4.MESSAGES.
3. Performs the following loop until there are no more messages on queue CSQ4SAMP.B4.MESSAGES:
   a. Get a message using the MQGET call with the wait option, and with the wait interval set to 30 seconds.
   b. Put a message on each queue listed in the namelist, specifying the name of the appropriate CSQ4SAMP.B2.REPLY.n queue for the reply-to queue. The program copies the *`CorrelId`* of the loan request message to these copy messages, and it uses the MQPMO_PASS_IDENTITY_CONTEXT option on the MQPUT call.
   c. Send a datagram message to queue CSQ4SAMP.B2.REPLY.n to show how many messages it has successfully put.
   d. Declare a syncpoint.

z/OS *Agency-query program (CSQ4CVB5/CSQ4CCB5) on z/OS*

The agency-query program is supplied as both a COBOL program and a C program. Both programs have the same design. This shows that programs of different types can easily coexist within an IBM MQ application, and that the program modules that make up such an application can easily be replaced.

An instance of the program is started by a trigger event on any of these queues:

- For the COBOL program (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- For the C program (CSQ4CCB5), queue CSQ4SAMP.B8.MESSAGES

Note: If you want to use the C program, you must alter the definition of the namelist CSQ4SAMP.B4.NAMELIST to replace the queue CSQ4SAMP.B7.MESSAGES with CSQ4SAMP.B8.MESSAGES. To do this, you can use any one of:

- The IBM MQ for z/OS operations and control panels
- The ALTER NAMELIST command
- The CSQUTIL utility

After it has opened the appropriate queue, this program gets a message from the queue using the MQGET call with the wait option, and with the wait interval set to 30 seconds.

The program simulates the search of an agency's database by searching the VSAM data set CSQ4BAQ for the account number that was passed in the loan request message. It then builds a reply that includes the name of the queue that it is serving and a creditworthiness index. To simplify the processing, the creditworthiness index is selected at random.

When putting the reply message, the program uses the MQPUT1 call and:

- Copies the `CorrelId` of the loan request message
- Uses the MQPMO_PASS_IDENTITY_CONTEXT option

The program sends the reply message to the reply-to queue named in the loan request message. (The name of the queue manager that owns the reply-to queue is also specified in the loan request message.)

z/OS *Design considerations for the Credit check sample on z/OS*

Design considerations for the Credit Check sample.

This topic contains information about:

Separate inquiry and reply queues in the CAM

The application could use a single queue for both inquiries and replies, but it was designed to use separate queues for the following reasons:

- When the program is handling the maximum number of inquiries, further inquiries can be left on the queue. If a single queue is being used, this would have to be taken off the queue and stored elsewhere.
- Other instances of the CAM could be started automatically to service the same inquiry queue if message traffic was high enough to warrant it. But the program must track in-progress inquiries, and to do this,

it must get back all replies to inquiries it has initiated. If only one queue is used, the program would have to browse the messages to see if they were for this program or for another. This would make the operation much less efficient.

The application can support multiple CAMs and can recover in-progress inquiries effectively by using paired reply-to and waiting queues.

- The program can wait on multiple queues effectively by using signaling.

How the sample handles errors

The user interface program handles errors by reporting them directly to the user.

The other programs do not have user interfaces, so they have to handle errors in other ways. Also, in many situations (for example, if an MQGET call fails) these other programs do not know the identity of the user of the application.

The other programs put error messages on a CICS temporary storage queue called CSQ4SAMP. You can browse this queue using the CICS-supplied transaction CEBR. The programs also write error messages to the CICS CSML log.

How the sample handles unexpected messages

When you design a message-queuing application, you must decide how to handle messages that arrive on a queue unexpectedly.

The two basic choices are:

- The application does no more work until it has processed the unexpected message. This probably means that the application notifies an operator, terminates itself, and ensures that it is not restarted automatically (it can do this by setting triggering off). This choice means that all processing for the application can be halted by a single unexpected message, and the intervention of an operator is required to restart the application.
- The application removes the message from the queue it is serving, puts the message in another location, and continues processing. The best place to put this message is on the system dead-letter queue.

If you choose the second option:

- An operator, or another program, should examine the messages that are put on the dead-letter queue to find out where the messages are coming from.
- An unexpected message is lost if it cannot be put on the dead-letter queue.
- A long unexpected message is truncated if it is longer than the limit for messages on the dead-letter queue, or longer than the buffer size in the program.

To ensure that the application smoothly handles all inquiries with minimal effect from outside activities, the Credit Check sample application uses the second option. To allow you to keep the sample separate from other applications that use the same queue manager, the Credit Check sample does not use the system dead-letter queue; instead, it uses its own dead-letter queue. This queue is named CSQ4SAMP.DEAD.QUEUE. The sample truncates any messages that are longer than the buffer area provided for the sample programs. You can use the Browse sample application to browse messages on this queue, or use the Print Message sample application to print the messages together with their message descriptors.

However, if you extend the sample to run across more than one queue manager, unexpected messages, or messages that cannot be delivered, could be put on the system dead-letter queue by the queue manager.

How the sample uses syncpoints

The programs in the Credit Check sample application declare syncpoints to ensure that:

- Only one reply message is sent in response to each expected message

- Multiple copies of unexpected messages are never put on the sample's dead-letter queue
- The CAM can recover the state of all partially completed inquiries by getting persistent messages from its waiting queue

To achieve this, a single unit of work is used to cover the getting of a message, the processing of that message, and any subsequent put operations.

How the sample uses message context information

When the user interface program (CSQ4CVB1) sends messages, it uses the MQPMO_DEFAULT_CONTEXT option. This means that the queue manager generates both identity and origin context information. The queue manager gets this information from the transaction that started the program (MVB1) and from the user ID that started the transaction.

When the CAM sends inquiry messages, it uses the MQPMO_PASS_IDENTITY_CONTEXT option. This means that the identity context information of the message being put is copied from the identity context of the original inquiry message. With this option, origin context information is generated by the queue manager.

When the CAM sends reply messages, it uses the MQPMO_ALTERNATE_USER_AUTHORITY option. This causes the queue manager to use an alternate user ID for its security check when the CAM opens a reply-to queue. The CAM uses the user ID of the submitter of the original inquiry message. This means that users are allowed to see replies to only those inquiries that they have originated. The alternate user ID is obtained from the identity context information in the message descriptor of the original inquiry message.

When the query programs (CSQ4CVB3/4/5) send reply messages, they use the MQPMO_PASS_IDENTITY_CONTEXT option. This means that the identity context information of the message being put is copied from the identity context of the original inquiry message. With this option, origin context information is generated by the queue manager.

Note: The user ID associated with the MVB3/4/5 transactions requires access to the B2.REPLY.n queues. These user IDs might not be the same as those associated with the request being processed. To get around this possible security exposure, the query programs could use the MQPMO_ALTERNATE_USER_AUTHORITY option when putting their replies. This would mean that each individual user of MVB1 needs authority to open the B2.REPLY.n queues.

Use of message and correlation identifiers in the CAM

The application has to monitor the progress of all the live inquiries it is processing at any one time. To do this it uses the unique message identifier of each loan request message to associate all the information that it has about each inquiry.

The CAM copies the `MsgId` of the inquiry message into the `CorrelId` of all the request messages it sends for that inquiry. The other programs in the sample (CSQ4CVB3 - 5) copy the `CorrelId` of each message that they receive into the `CorrelId` of their reply message.

z/OS *The Credit Check sample with multiple queue managers on z/OS*

You can use the Credit Check sample application to demonstrate distributed queuing by installing the sample on two queue managers and CICS systems (with each queue manager connected to a different CICS system).

When the sample program is installed, and the trigger monitor (CKTI) is running on each system, you need to:

1. Set up the communication link between the two queue managers. For information on how to do this, see Configuring distributed queuing.
2. On one queue manager, create a local definition for each of the remote queues (on the other queue manager) that you want to use. These queues can be any of CSQ4SAMP.B *n*.MESSAGES, where *n* is 3, 5, 6, or 7. (These are the queues that are served by the checking-account program and the agency-query program.) For information on how to do this, see DEFINE QREMOTE and DEFINE queues.

3. Change the definition of the namelist (CSQ4SAMP.B4.NAMELIST) so that it contains the names of the remote queues that you want to use. For information on how to do this, see DEFINE NAMELIST.

z/OS *The IMS extension to the Credit Check sample on z/OS*

A version of the checking-account program is supplied as an IMS batch message processing (BMP) program. It is written in the C language.

The program performs the same function as the CICS version, except that to obtain the account information, the program reads an IMS database instead of a VSAM file. If you replace the CICS version of the checking-account program with the IMS version, you see no difference in the method of using the application.

To prepare and run the IMS version you must:

1. Follow the steps in "Preparing and running the Credit Check sample on z/OS" on page 1210.
2. Follow the steps in "Preparing the sample application for the IMS environment on z/OS" on page 1191.
3. Alter the definition of the alias queue CSQ4SAMP.B2.OUTPUT.ALIAS to resolve to queue CSQ4SAMP.B3.IMS.MESSAGES (instead of CSQ4SAMP.B3.MESSAGES). To do this, you can use one of:
   - The IBM MQ for z/OS operations and control panels
   - The ALTER QALIAS command .

Another way of using the IMS checking-account program is to make it serve one of the queues that receives messages from the distribution program. In the delivered form of the Credit Check sample application, there are three of these queues (B5/6/7.MESSAGES), all served by the agency-query program. This program searches a VSAM data set. To compare the use of the VSAM data set and the IMS database, you could make the IMS checking-account program serve one of these queues instead. To do this, you must alter the definition of the namelist CSQ4SAMP.B4.NAMELIST to replace one of the CSQ4SAMP.B *n*.MESSAGES queues with the CSQ4SAMP.B3.IMS.MESSAGES queue. You can use one of:

- The IBM MQ for z/OS operations and control panels
- The ALTER NAMELIST command.

You can then run the sample from CICS transaction MVB1. The user sees no difference in operation or response. The IMS BMP stops either after receiving a stop message or after being inactive for five minutes.

Design of the IMS checking-account program (CSQ4ICB3)

This program runs as a BMP. Start the program using its JCL before any IBM MQ messages are sent to it.

The program searches an IMS database for the account number in the loan request messages. It retrieves the corresponding account name, average balance, and credit worthiness index.

The program sends the results of the database search to the reply-to queue named in the IBM MQ message being processed. The message returned appends the account type and the results of the search to the message received so that the transaction building the response can confirm that the correct query is being processed. The message is in the form of three 79-character groups, as follows:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

When running as a message-oriented BMP, the program drains the IMS message queue, then reads messages from the IBM MQ for z/OS queue and processes them. No information is received from the IMS message queue. The program reconnects to the queue manager after each checkpoint because the handles have been closed.

When running in a batch-oriented BMP, the program continues to be connected to the queue manager after each checkpoint because the handles are not closed.

z/OS *The Message Handler sample on z/OS*

The Message Handler sample TSO application allows you to browse, forward, and delete messages on a queue. The sample is available in C and COBOL.

Preparing and running the sample

Follow these steps:

1. Prepare the sample as described in "Preparing sample applications for the TSO environment on z/OS" on page 1185.
2. Tailor the CLIST (CSQ4RCH1) provided for the sample to define the location of the panels, the location of the message file, and the location of the load modules.

You can use CLIST CSQ4RCH1 to run both the C and the COBOL version of the sample. The supplied version of CSQ4RCH1 runs the C version, and contains instructions on the tailoring necessary for the COBOL version.

Note:

1. There are no sample queue definitions provided with the sample.
2. VS COBOL II does not support multitasking with ISPF, so do not use the Message Handler sample application on both sides of a split screen. If you do, the results are unpredictable.

z/OS *Using the Message Handler sample on z/OS*

Having installed the sample and invoked it from the tailored CLIST CSQ4RCH1, the screen shown in Figure 146 on page 1221 is displayed.

```
----------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :

Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP     F8=DOWN    F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Figure 146. Initial screen for Message Handler sample

Enter the queue manager and queue name to be viewed (case sensitive) and the message list screen is displayed (see Figure 147 on page 1222).

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                       :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

Figure 147. Message list screen for Message Handler sample

This screen shows the first 99 messages on the queue and, for each, shows the following fields:

Msg No
Message number

Put Date MM/DD/YYYY
Date that the message was put on the queue (GMT)

Put Time HH:MM:SS
Time that the message was put on the queue (GMT)

Format Name
MQMD.Format field

User Identifier
MQMD.UserIdentifier field

Put Application Type
MQMD.PutApplType field

Put Application Name
MQMD.PutApplName field

The total number of messages on the queue is also displayed.

From this screen a message can be chosen, by number not by cursor position, and then displayed. For an example, see Figure 148 on page 1223.

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                       :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :
Forward to Q Mgr : VM03                        :
Forward to Queue : QL.TEST.ISCRES1                  :

Action : _ :  (D)elete (F)orward

Message Content :
------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount    : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                  `
ReplyToQMgr    : `VM03                       `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                        `
PutApplType    : 000000002
PutApplName    : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .............`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

Figure 148. Chosen message is displayed

Once the message has been displayed it can be deleted, left on the queue, or forwarded to another queue. The `Forward to Q Mgr` and `Forward to Queue` fields are initialized with values from the MQMD, these can be changed before forwarding the message.

The sample design allows only messages with unique `MsgId / CorrelId` combinations to be selected and displayed, because the message is retrieved using the `MsgId` and `CorrelId` as the key. If the key is not unique the sample cannot retrieve the chosen message with certainty.

Note: When you use the SCSQCLST(CSQ4RCH1) sample to browse messages, each invocation causes the backout count of the message to increase. If you want to change the behavior of this sample, copy the sample and modify the contents as necessary. You should be aware that other applications that rely on this backout count can be influenced by this increasing count.

z/OS *Design of the sample Message Handler sample on z/OS*

This topic describes the design of each of the programs that make up the Message Handler sample application.

Object validation program

This requests a valid queue and queue manager name.

If you do not specify a queue manager name, the default queue manager is used, if available. Only local queues can be used; an MQINQ is issued to check that the queue type and an error is reported if the queue is not local. If the queue is not opened successfully, or the MQGET call is inhibited on the queue, error messages are returned indicating the CompCode and Reason return code.

Message list program

This displays a list of messages on a queue with information about them such as the putdate, puttime, and the message format.

The maximum number of messages stored in the list is 99. If there are more messages on the queue than this, the current queue depth is also displayed. To choose a message for display, type the message number into the entry field (the default is 01). If your entry is not valid, you receive an appropriate error message.

Message content program

This displays message content.

The content is formatted and split into two parts:

1. Message descriptor
2. Message buffer

The message descriptor shows the contents of each field on a separate line.

The message buffer is formatted depending on its contents. If the buffer holds a dead letter header (MQDLH) or a transmission queue header (MQXQH), these are formatted and displayed before the buffer itself.

Before the buffer data is formatted, a title line shows the buffer length of the message in bytes. The maximum buffer size is 32768 bytes, and any message longer than this is truncated. The full size of the buffer is displayed along with a message indicating that only the first 32768 bytes of the message are displayed.

The buffer data is formatted in two ways:

1. After the offset into the buffer is printed, the buffer data is displayed in hexadecimal.
2. The buffer data is then displayed again as EBCDIC values. If any EBCDIC value cannot be printed, it prints a period (.) instead.

You can enter D for delete, or F for forward into the action field. If you choose to forward the message, the `forward-to queue` and `queue manager name` must be set correctly. The defaults for these fields are read from the message descriptor ReplyToQ and ReplyToQMgr fields.

If you forward a message, any header block stored in the buffer is stripped. If the message is forwarded successfully, it is removed from the original queue. If you enter invalid actions, error messages are displayed.

An example help panel called CSQ4CHP9 is also available.

z/OS *The Asynchronous Put sample on z/OS*

The Asynchronous Put sample program puts messages on a queue using the asynchronous MQPUT call. The sample also retrieves status information using the MQSTAT call.

The Asynchronous Put applications use these MQI calls:

- MQCONN
- MQOPEN

- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

The sample programs are delivered in the C programming language.

The Asynchronous Put applications run in the batch environment. See Other samples for the batch applications.

This topic also provides information about the design of the Asynchronous Consumption program, and running the CSQ4BCS2 sample.

- "Running the CSQ4BCS2 sample" on page 1225
- "Design of the Asynchronous Put sample program" on page 1225

Running the CSQ4BCS2 sample

This sample program takes up to six parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).
5. The name of the target queue manager (optional).
6. The name of the dynamic queue (optional).

If a queue manager is not specified, CSQ4BCS2 connects to the default queue manager. Message content is provided through standard input (**SYSIN DD**).

There is a sample JCL to run the program, it resides in CSQ4BCSP.

Design of the Asynchronous Put sample program

The program uses the MQOPEN call with either the output options supplied, or with the MQOO_OUTPUT and MQOO_FAIL_IF_QUIESCING options, to open the target queue for putting messages.

If the program cannot open the queue, the program outputs an error message containing the reason code returned by the MQOPEN call. To keep the program simple on this and subsequent MQI calls, default values are used for many of the options.

For each line of input, the program reads the text into a buffer and uses the MQPUT call with MQPMO_ASYNC_RESPONSE to create a datagram message containing the text of that line and asynchronously puts the message on the target queue. The program continues until it reaches the end of the input, or until the MQPUT call fails. If the program reaches the end of the input, it closes the queue using the MQCLOSE call.

The program then issues the MQSTAT call which returns an MQSTS structure, and displays messages containing the number of messages put successfully, the number of messages put with a warning, and the number of failures.

Note: To observe what happens when an MQPUT error is detected by the MQSTAT call, set MAXDEPTH on the target queue to a low value.

z/OS *The Batch Asynchronous Consumption sample on z/OS*

The CSQ4BCS1 sample program is delivered in C, it demonstrates the use of MQCB and MQCTL to consume messages from multiple queues asynchronously.

The Asynchronous Consumption samples run in the batch environment. See Other samples for the batch applications.

There is also a COBOL sample which runs in the CICS environment, see “The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” on page 1227.

The applications use these MQI calls:

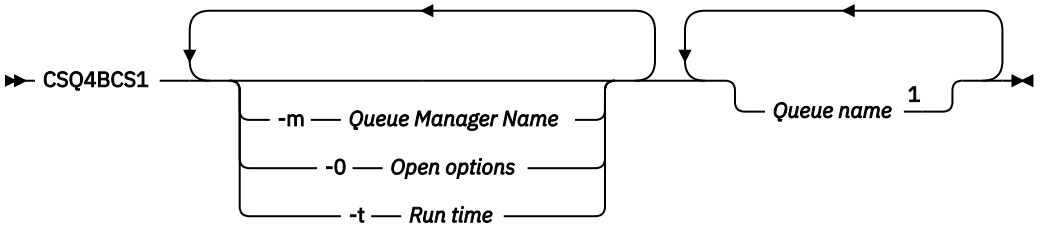
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

This topic also provided information about the following headings:

- “Running the CSQ4BCS1 sample” on page 1226
- “Design of the Batch Asynchronous Consumption sample program” on page 1226

Running the CSQ4BCS1 sample

This sample program follows the following syntax:



Notes:

[1] A maximum of 10 queues are supported by this sample. Provide more than one queue name in order to read messages from multiple queues.

There is a sample JCL to run this program, it resides in CSQ4BCSC.

Design of the Batch Asynchronous Consumption sample program

The sample shows how to read messages from multiple queues in the order of their arrival. This would require more code using synchronous MQGET. With asynchronous consumption, no polling is required, and thread and storage management is performed by IBM MQ. In the sample program, errors are written to the console.

The sample code has the following steps:

1. Define the single message consumption callback function.

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Connect to the queue manager.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Open the input queues, and associate each queue with the `MessageConsumer` callback function.

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` does not need to be set for each queue; it is an input-only field. You can associate a different callback function with each queue.

4. Start consumption of the messages.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Wait for the user to press Enter, then stop consumption of messages.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Finally, disconnect from the queue manager.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS*

The Asynchronous Consumption and Publish/Subscribe sample programs demonstrate the use of asynchronous consumption, and publish and subscribe features within CICS.

A *Registration client* program registers three Callback handlers (an event handler, and two message consumers), and starts Asynchronous Consumption. A *Messaging client* program puts messages to a queue, or publishes suitable messages from a CICS console for consumption by the two Message Consumers (CSQ4CVCN and CSQ4CVCT).

To provide runtime control over the behavior of the sample, one of the message consumers can be instructed using the messages it receives, to SUSPEND, RESUME, or DEREGISTER any of the Callback handlers. It can also be used to issue an MQCTL STOP to end Asynchronous Consumption under control. The other message consumer is registered to subscribe to a topic.

Each program issues COBOL DISPLAY statements at appropriate points to display the behavior of the sample.

The applications use these MQI calls:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

The programs are delivered in the COBOL language. See CICS Asynchronous Consumption and Publish/Subscribe samples for the CICS applications.

This topic also provides the following information:

- “Setup” on page 1228
- “Registration Client CSQ4CVRG” on page 1228
- “Event handler CSQ4CVEV” on page 1228
- “Simple Message Consumer CSQ4CVCN” on page 1228

- “Control Message Consumer CSQ4CVCT” on page 1228
- “Messaging Client CSQ4CVPT” on page 1228

Setup

The names of the Queue and Topic used by the Message Consumers are hardcoded in the Registration and Messaging Client programs.

The Queue, **SAMPLE.CONTROL.QUEUE**, should be defined to the Queue Manager associated with the CICS region before running the sample. The Topic, **News/Media/Movies**, can be defined if required, or it is created at runtime under the default Administrative Object if it does not exist.

CICS programs and transaction definitions can be installed by installing a group: CSQ4SAMP.

Registration Client CSQ4CVRG

The Registration Client program must be started under the CICS transaction MVRG. It takes no input.

When started, the Registration Client registers the following Callback handlers using MQCB:

- CSQ4CVEV as an Event Handler.
- CSQ4CVCN as a Message Consumer on a topic, **News/Media/Movies**.
- CSQ4CVCT as a Message Consumer on a Queue, **SAMPLE.CONTROL.QUEUE**.

The Registration Client passes a data structure containing the names of all three registered Callback handlers to CSQ4CVCT, together with the object handles associated with the two message consumers.

Having registered the Callback handlers, the Registration Client issues an MQCTL START_WAIT to start Asynchronous Consumption, and suspend until control is returned to it (for example, by one of the Callback handlers issuing an MQCTL STOP).

Event handler CSQ4CVEV

When driven, the Event Handler displays a message indicating the call type (for example, START). When driven for IBM MQ reason code CONNECTION_QUIESCING, the Event Handler issues an MQCTL STOP to end Asynchronous Consumption and return control to the Registration Client.

Simple Message Consumer CSQ4CVCN

When driven, this Message Consumer displays a message indicating the call type (for example, REGISTER). When driven for the MSG_REMOVED call type, the Message Consumer retrieves the inbound message and outputs it to the CICS job log.

Control Message Consumer CSQ4CVCT

When driven, this Message Consumer displays a message indicating the call type (for example, START). When driven for the MSG_REMOVED call type, the Message Consumer retrieves the inbound message and the data structure passed by the Registration Client. Based on the message content, it issues appropriate MQCB or MQCTL commands to one of the following:

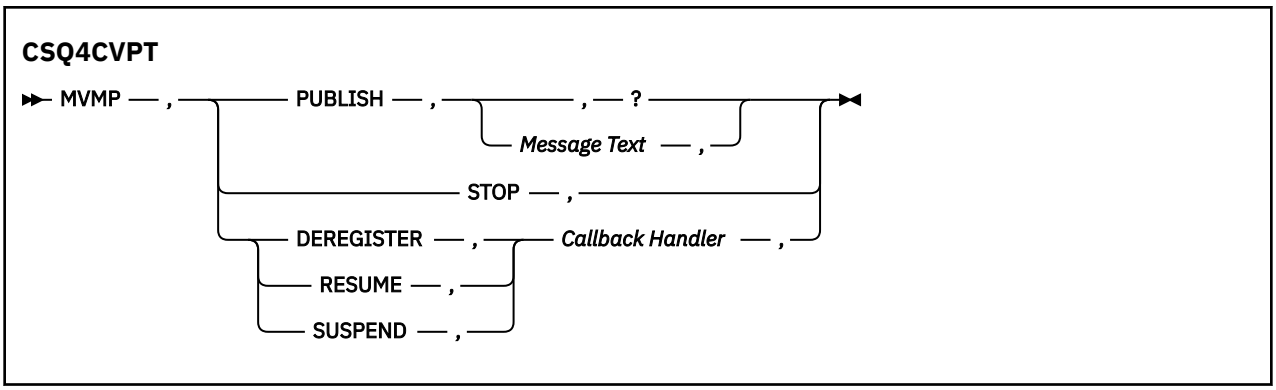
- STOP Asynchronous Consumption (returning control to the Registration Client).
- SUSPEND, RESUME, or DEREGISTER a named Callback handler (including itself).

Messaging Client CSQ4CVPT

The Messaging Client has two functions:

- It publishes a message to a topic for consumption by the Message Consumer CSQ4CVCN.
- It puts a control message to a queue for consumption by the Control Message Consumer CSQ4CVCT, resulting in a potential change in behavior of the sample.

The Messaging Client program must be started from a CICS console under a CICS transaction, and it takes command line input with the following syntax:



PUBLISH
Publish the Message Text (or a default message) as a Retained Message for consumption by the Simple Message Consumer.

STOP
Stop Asynchronous Consumption.

DEREGISTER
Deregister the named Callback handler.

RESUME
Resume the named Callback handler.

SUSPEND
Suspend the named Callback handler.

Input fields are positional, and comma-separated. Keywords and Callback Handler names are not case-sensitive.

Examples:

Table 186. Input examples

Example	Description
MVMP,PUBLISH,,	Publish a default message
MVMP,publish, `A short message`,	Publish the given text
MVMP,STOP,	Stop Asynchronous Consumption
MVMP,DEREGISTER,CSQ4CVEV,	Deregister the Event Handler
MVMP,resume,csq4cvcn,	Resume the Simple Message Consumer
MVMP,SUSPEND,CSQ4CVEV,	Suspend the Event Handler

Where MVMP is the CICS transaction associated with the Messaging Client program CSQ4CVPT.

Note:

- Suspending or deregistering all Callback handlers terminates the START_WAIT issued by the Registration Client, returning control to it, and ending the task.
- Suspending or deregistering the Control Callback Handler has deliberately not been prevented, but it removes the ability to further control the behavior of the sample.

z/OS *The Publish/Subscribe sample on z/OS*

The Publish/Subscribe sample programs demonstrate the use of the publish and subscribe features in IBM MQ.

There are four C and two COBOL programming language sample programs demonstrating how to program to the IBM MQ Publish/Subscribe interface. The programs are delivered in the C and COBOL language. The applications run in the batch environment; see Publish/Subscribe samples for the batch applications.

There are also COBOL samples that run in the CICS environment; see "The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS" on page 1227.

This topic also provides information about how to run Publish/Subscribe sample programs. These sample programs include:

Running the CSQ4BCP1 sample

This program is written in C; it publishes messages to a topic. Start one of the subscriber samples before running this program.

This program takes up to four parameters:

1. The name of the target topic string (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).

If a queue manager is not specified, CSQ4BCP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPP.

Message content is provided through standard input (**SYSIN DD**).

Running the CSQ4BCP2 sample

This program is written in C; it subscribes to a topic and prints the messages received.

This program takes up to three parameters:

1. The name of the target topic string (required).
2. The name of the queue manager (optional).
3. MQSD subscription options (optional).

If a queue manager is not specified, CSQ4BCP2 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPS.

Running the CSQ4BCP3 sample

This program is written in C; it subscribes to a topic using a user-specified destination queue and prints the messages received.

This program takes up to four parameters:

1. The name of the target topic string (required).
2. The name of the destination (required).
3. The name of the queue manager (optional).
4. MQSD subscription options (optional).

If a queue manager is not specified, CSQ4BCP3 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPD.

Running the CSQ4BCP4 sample

This program is written in C; it subscribes and gets messages from a topic allowing the use of extended options on the MQSUB call, extending those available on the simpler MQSUB sample: CSQ4BCP2. In addition to the message payload, message properties for each message are received and displayed.

This program takes a variable set of parameters:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  Important: One of **-t** or **-o**, or both, is required
- **-m** *Queue manager name* (optional).
- **-b** *Connection binding type* (optional), where *type* can have any of the following values:
  - *standard* : MQCNO_STANDARD_BINDING , which is the default value
  - *shared*: MQCNO_SHARED_BINDING
  - *fastpath*: MQCNO_FASTPATH_BINDING
  - *isolated*: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (optional).
- **-w** *Wait interval on MQGET in seconds* (optional), where *seconds* can have any of the following values:
  - unlimited: MQWI_UNLIMITED
  - none: No wait
  - *n*: Wait interval in seconds
  - No value specified: When no value is specified, the default is 30 seconds
- **-d** *Subscription name* (optional). Creates or resumes named durable subscription.
- **-k** (optional). Keeps durable subscription on MQCLOSE.

If a queue manager is not specified, CSQ4BCP4 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPE.

Running the CSQ4BVP1 sample

This program is written in COBOL, it publishes messages to a topic. Start one of the subscriber samples before running this program.

This program takes no parameters. **SYSIN DD** provides the input topic name, queue manager name, and message content.

If a queue manager is not specified, CSQ4BVP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BVPP.

Running the CSQ4BVP2 sample

This program is written in COBOL, it subscribes to a topic and prints the messages received.

This program takes no parameters. **SYSIN DD** provides the input for topic name and queue manager name.

If a queue manager is not specified, CSQ4BVP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BVPP.

z/OS *The Set and Inquire message property sample on z/OS*

The message property sample programs demonstrate the addition of user-defined properties to a message handle, and the inquisition of the properties associated with that message.

The applications use these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

The programs are delivered in the C language. The applications run in the batch environment. See Other samples for the batch applications.

The CSQ4BCM1 program is used to inquire the properties of a message handle from a message queue, and it is an example of the use of the MQINQMP API call. The sample gets one message from a queue and then prints all the message handle properties.

The CSQ4BCM2 program is used to set the properties of a message handle on a message queue, and it is an example of the use of the MQSETMP API call. The sample creates a message handle and puts it into the `MsgHandle` field of the MQGMO structure. It then puts the message to a queue.

Other examples of inquiring and printing message properties are included in the CSQ4BCG1 and CSQ4BCP4 sample programs.

This topic also provides information on running the Set and Inquire message property samples under the following headings:

- "Running the CSQ4BCM1 sample" on page 1232
- "Running the CSQ4BCM2 sample" on page 1232

Running the CSQ4BCM1 sample

This program takes up to four parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).

Running the CSQ4BCM2 sample

This program takes up to six parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).
5. The name of the target queue manager (optional).
6. The name of the dynamic queue (optional).

The property names, values, and message content are provided through the standard input (**SYSIN DD**). There is a sample JCL to run the program, it resides in CSQ4BCMP.

Desenvolvendo aplicativos para o Managed File Transfer

Especifique programas para executar com Managed File Transfer, use o Apache Ant com o Managed File Transfer, customize o Managed File Transfer com saídas de usuários e controle o Managed File Transfer ao colocar mensagens na fila de comandos do agente.

Especificando programas para executar com o MFT

É possível executar programas em um sistema no qual um Managed File Transfer Agent está em execução. Como parte de uma solicitação de transferência de arquivos, é possível especificar um programa para execução antes do início de uma transferência ou após sua conclusão. Além disso, é possível iniciar um programa que não faz parte de uma solicitação de transferência de arquivos, submetendo uma solicitação de chamada gerenciada.

Sobre esta tarefa

Há cinco cenários nos quais é possível especificar um programa para execução:

- Como parte de uma solicitação de transferência, no agente de origem, antes do início da transferência.
- Como parte de uma solicitação de transferência, no agente de destino, antes do início da transferência.
- Como parte de uma solicitação de transferência, no agente de origem, após a conclusão da transferência.
- Como parte de uma solicitação de transferência, no agente de destino, após a conclusão da transferência.
- Não como parte de uma solicitação de transferência. É possível submeter uma solicitação para um agente para executar um programa. Este cenário às vezes é referido como uma chamada gerenciada.

As saídas de usuário e chamadas de programas são chamadas na seguinte ordem:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Notas:

1. O **DestinationTransferEndExits** é executado apenas quando a transferência é concluída, com sucesso ou parcialmente com sucesso.
2. O **postDestinationCall** é executado apenas quando a transferência é concluída, com sucesso ou parcialmente com sucesso.
3. O **SourceTransferEndExits** é executado para transferências bem-sucedidas, parcialmente bem-sucedidas ou com falha.
4. O **postSourceCall** será chamado apenas se:
   - A transferência não foi cancelada.
   - Há um resultado bem-sucedido ou parcialmente bem-sucedido.
   - Quaisquer programas de transferência pós-destino foram executados com sucesso.

Procedimento

- Especifique o programa que você deseja executar usando uma das opções a seguir:

Use uma tarefa Apache Ant

Use uma das tarefas `fte:filecopy`, `fte:filemovee` `fte:call` Ant para iniciar um programa. Usando uma tarefa Ant, é possível especificar um programa em qualquer um dos cinco cenários, usando os elementos aninhados `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrc` e `fte:command`. Para obter mais informações, consulte Elementos aninhados de chamada de programa.

Editar a Mensagem de Solicitação de Transferência de Arquivos

É possível editar o XML gerado por uma solicitação de transferência. Usando esse método, é possível executar um programa em qualquer um dos cinco cenários, incluindo os elementos **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall** e **managedCall** no arquivo XML. Em seguida, use este arquivo XML modificado como a definição de transferência para uma nova solicitação de transferência de arquivos, por exemplo, com o parâmetro **fteCreateTransfer -td**. Para obter mais informações, consulte Exemplos de mensagem de solicitação de chamada do agente MFT.

Usar o comando `fteCreateTransfer`

É possível usar o comando **fteCreateTransfer** para especificar programas para iniciar. É possível usar o comando para especificar programas para execução nos primeiros quatro cenários, como parte de uma solicitação de transferência, mas não é possível iniciar uma chamada gerenciada. Para obter informações sobre os parâmetros a serem usados, consulte **fteCreateTransfer**: iniciar uma nova transferência de arquivos. Para obter exemplos de como usar esse comando, consulte Exemplos de uso de fteCreateTransfer para iniciar programas.

Referências relacionadas

Propriedade commandPath do MFT

Chamadas gerenciadas

Managed File Transfer (MFT) os agentes são tipicamente utilizados para transferir arquivos ou mensagens. Eles são conhecidos como *Transferências gerenciadas*. Os agentes também podem ser usados para executar comandos, scripts ou JCL sem a necessidade de transferência de arquivos ou mensagens. Essa capacidade é conhecida como *Chamadas gerenciadas*.

As solicitações de chamadas gerenciadas podem ser enviadas para um agente de várias formas:

- Usando a tarefa fte:callAnt.
- Configurando um monitor de recursos com um XML de tarefa que executa um comando ou um script. Para obter mais informações, consulte Configurando tarefas do monitor para iniciar comandos e scripts.
- Colocando diretamente uma mensagem XML na fila de comando do agente Consulte Formato de mensagem de solicitação de transferência de arquivo para obter mais detalhes sobre o esquema XML de Chamada gerenciada.

Para chamadas gerenciadas, o diretório que contém o comando ou script que está sendo executado deve ser especificado na propriedade do agente **commandPath**.

As chamadas gerenciadas não podem executar comandos ou scripts que estão localizados em diretórios não especificados no **commandPath**do agente. Isso é para assegurar que o agente não execute nenhum código malicioso.

Importante: Para assegurar que esse seja o caso, por padrão, ao especificar **commandPath**:

- Qualquer ambiente de simulação do agente existente é configurada pelo agente quando ele é inicializado para que todos os diretórios **commandPath** sejam incluídos automaticamente na lista de diretórios que tenham acesso negado para uma transferência
- Quaisquer ambientes de simulação do usuário existentes são atualizados quando o agente é iniciado para que todos os diretórios **commandPath** (e seus subdiretórios) sejam incluídos como elementos `<exclude>` para os elementos `<read>` e `<write>`

- Se o agente não estiver configurado para usar um ambiente de simulação do agente ou ambientes de simulação do usuário, um novo ambiente de simulação do agente será criado quando o agente for iniciado com os diretórios **commandPath** especificados como diretórios negados.

Além disso, também é possível ativar a verificação de autoridade em um agente para assegurar que apenas usuários autorizados possam enviar solicitações de chamadas gerenciadas. Para obter mais informações sobre isso, consulte Restringindo as autoridades do usuário nas MFTações do agente.

O comando, script ou JCL chamado como parte de uma chamada gerenciada funciona como um processo externo, que é monitorado pelo agente. Quando o processo sai, a chamada gerenciada é concluída e o código de retorno do processo é disponibilizado para o agente ou o script Ant que invocou a tarefa **fte:call**Ant.

Se a chamada gerenciada foi iniciada pela tarefa **fte:call** Ant, o seu script Ant pode verificar o valor do código de retorno para determinar se a chamada gerenciada foi bem-sucedida ou não.

Para todos os outros tipos de chamadas gerenciadas, é possível especificar quais valores de código de retorno devem ser usados para indicar que a chamada gerenciada foi concluída com sucesso. O agente compara o código de retorno do processo com relação a esses códigos de retorno quando o processo externo é finalizado.

Nota: Como as chamadas gerenciadas são executadas como processos externos, elas não podem ser canceladas depois que são iniciadas.

Chamadas gerenciadas e slots de transferência de origem

Um agente contém um número de slots de transferência de origem, conforme especificado pela propriedade do agente **maxSourceTransfers**, descrita em Propriedades avançadas do agente: limite de transferência.

Sempre que uma chamada gerenciada ou uma transferência gerenciada é executada, ela ocupa um slot de transferência de origem. O slot é liberado quando a chamada gerenciada ou transferência gerenciada é concluída.

Se todos os slots de transferência de origem estiverem em uso quando um agente receber uma nova chamada gerenciada ou solicitação de transferência gerenciada, a solicitação será enfileirada pelo agente até que um slot fique disponível.

Se uma chamada gerenciada iniciar uma transferência gerenciada (por exemplo, se uma chamada gerenciada executar um script Ant e esse script Ant usar a tarefa fte:filecopy ou fte:filemove para transferir um arquivo), serão necessários dois slots de transferência de origem:

- Um para a transferência gerenciada
- Um para a chamada gerenciada

Nessa situação, é importante observar que, se a transferência gerenciada levar muito tempo para ser concluída ou entrar em recuperação, os dois slots de transferência de origem serão ocupados até que a transferência gerenciada seja concluída, cancelada ou atinja o tempo limite devido a um **transferRecoveryTimeout**. Consulte Conceitos de tempo limite de recuperação de transferência para obter detalhes sobre **transferRecoveryTimeout**. Isso pode potencialmente limitar o número de outras transferências gerenciadas ou chamadas gerenciadas que o agente pode processar.

Por causa disso, você deve considerar o design de uma chamada gerenciada para assegurar que ela não ocupe slots de transferência de origem por um longo período de tempo.

Usando o REST API com chamadas gerenciadas

Os verbos HTTP GET e HTTP POST são suportados para ativar chamadas gerenciadas e funcionam apenas na Versão 3 do REST API.

Outros verbos, por exemplo, HTTP DELETE e HTTP UPDATE não são suportados e retornam o código de erro HTTP 405 se você tentar usá-los..

Atenção: Uma vez enviada, uma chamada gerenciada não pode ser cancelada usando o REST API

Usando o Apache Ant com o MFT

O Managed File Transfer fornece tarefas que podem ser usadas para integrar a função de transferência de arquivos para a ferramenta Apache Ant.

Você pode usar o comando **fteAnt** para executar tarefas Ant em um ambiente Managed File Transfer que você já configurou. É possível usar tarefas de transferência de arquivos Ant por meio de seus scripts Ant para coordenar operações complexas de transferência de arquivos por meio de uma linguagem de script interpretada.

Para obter mais informações sobre o Apache Ant, consulte a página da web do projeto Apache Ant: https://ant.apache.org/

Conceitos relacionados

“Introdução ao uso de scripts Ant com MFT” na página 1236
O uso de scripts Ant com o Managed File Transfer permite coordenar operações complexas de transferência de arquivos por meio de uma linguagem de script interpretada.

fteAnt: executar tarefas Ant no MFT

Referências relacionadas

“Tarefas Ant de amostra para o MFT” na página 1237
Há uma série de scripts de amostra Ant fornecidos com a sua instalação de Managed File Transfer. Essas amostras estão localizadas no diretório *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Cada script de amostra contém um destino `init`, edite as propriedades configuradas no destino `init` para executar esses scripts com a sua configuração.

Introdução ao uso de scripts Ant com MFT

O uso de scripts Ant com o Managed File Transfer permite coordenar operações complexas de transferência de arquivos por meio de uma linguagem de script interpretada.

Scripts Ant

Scripts Ant (ou arquivos de construção) são documentos XML que definem um ou mais destinos. Esses destinos contêm elementos de tarefa a serem executados. O Managed File Transfer fornece tarefas que podem ser usadas para integrar a função de transferência de arquivos no Apache Ant. Para aprender sobre scripts Ant, consulte a página da web do projeto Apache Ant: https://ant.apache.org/

Exemplos de scripts Ant que usam tarefas Managed File Transfer são fornecidos com a instalação do seu produto no diretório *MQ_INSTALLATION_PATH*/mqft/samples/fteant

Em agentes de ponte de protocolo, os scripts Ant são executados no sistema de agente de ponte de protocolo. Esses scripts Ant não têm acesso direto aos arquivos no servidor FTP ou SFTP.

Namespace

Um namespace é usado para diferenciar as tarefas Ant de transferência de arquivos de outras tarefas Ant que podem compartilhar o mesmo nome. Defina o namespace na tag de projeto de seu script Ant.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

O atributo `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` informa ao Ant para procurar as definições de tarefas prefixadas por `fte` na biblioteca `com.ibm.wmqfte.ant.taskdefs`.

Não é preciso usar `fte` como seu prefixo de espaço de nomes; é possível usar qualquer valor. O prefixo de namespace `fte` é usado em todos os exemplos e scripts de amostra Ant.

Executando scripts do Ant

Para executar scripts Ant que contêm as tarefas de transferência de arquivos Ant, use o comando **fteAnt**. Por exemplo:

```
fteAnt -file ant_script_location/ant_script_name
```

Para obter mais informações, consulte **fteAnt**: executar tarefas Ant no MFT.

Códigos de retorno

As tarefas Ant de transferência de arquivos retornam os mesmos códigos de retorno que os comandos Managed File Transfer. Para obter mais informações, consulte Códigos de retorno para o MFT.

Referências relacionadas

fteAnt: executar tarefas Ant no MFT

“Tarefas Ant de amostra para o MFT” na página 1237

Há uma série de scripts de amostra Ant fornecidos com a sua instalação de Managed File Transfer. Essas amostras estão localizadas no diretório *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Cada script de amostra contém um destino `init`, edite as propriedades configuradas no destino `init` para executar esses scripts com a sua configuração.

Tarefas Ant de amostra para o MFT

Há uma série de scripts de amostra Ant fornecidos com a sua instalação de Managed File Transfer. Essas amostras estão localizadas no diretório *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Cada script de amostra contém um destino `init`, edite as propriedades configuradas no destino `init` para executar esses scripts com a sua configuração.

Email

A amostra e-mail demonstra como usar as tarefas Ant para transferir um arquivo e enviar um e-mail para um endereço de e-mail especificado se a transferência falhar. O script verifica se os agentes de origem e de destino estão ativos e são capazes de processar transferências usando a tarefa Managed File Transferping . Se ambos os agentes estiverem ativos, o script usa a tarefa Managed File Transfer fte: filecopy para transferir um arquivo entre os agentes de origem e de destino, sem excluir o arquivo original. Se a transferência falhar, o script enviará um e-mail contendo informações sobre a falha usando a tarefa Ant padrão e-mail.

Hub

A amostra hub é composta por dois scripts: `hubcopy.xml` e `hubprocess.xml`. O script `hubcopy.xml` mostra como você pode usar o script Ant para construir topologias de estilo 'hub e spoke'. Nessa amostra, dois arquivos são transferidos dos agentes em execução nas máquinas spoke para um agente em execução na máquina hub. Ambos os arquivos são transferidos ao mesmo tempo e, quando as transferências são concluídas, o script `hubprocess.xml` Ant é executado na máquina hub para processar os arquivos. Se ambos os arquivos forem transferidos corretamente, o script Ant concatenará o conteúdo dos arquivos. Se os arquivos não forem transferidos corretamente, o script Ant será limpo excluindo todos os dados do arquivo que tenham sido transferidos. Para que este exemplo funcione corretamente, você deve colocar o script `hubprocess.xml` no caminho de comando do agente hub. Para obter mais informações sobre como configurar o caminho de comando de um agente, consulte Propriedade commandPath MFT.

librarytransfer (plataforma IBM i apenas)

IBM i

IBM i A amostra librarytransfer demonstra como usar as tarefas Ant para transferir uma biblioteca IBM i em um sistema IBM i para um segundo sistema IBM i.

IBM i A amostra librarytransfer usa o suporte de arquivo de salvamento nativo no IBM i com as tarefas Ant predefinidas disponíveis no Managed File Transfer para transferir objetos da biblioteca nativa entre dois sistemas IBM i. A amostra usa um elemento aninhado < presrc> em uma tarefa de arquivo Managed File Transfer para chamar um script executável `librarysave.sh` que salva a biblioteca solicitada no sistema do agente de origem em um arquivo de salvamento temporário. O arquivo de salvamento é movido pela tarefa Ant de cópia de arquivo para o sistema do agente de destino em que um elemento < postdst> aninhado é usado para chamar o script executável `libraryrestore.sh` para restaurar a biblioteca salva no arquivo de salvamento para o sistema de destino

IBM i Antes de executar esta amostra, é necessário completar alguma configuração conforme descrito no arquivo `librarytransfer.xml`. Você também deve ter um ambiente de trabalho do Managed File Transfer em duas máquinas IBM i. A configuração deve consistir em um agente de origem em execução na primeira máquina IBM i e em um agente de destino em execução na segunda máquina IBM i. Os dois agentes devem ser capazes de se comunicar.

IBM i A amostra librarytransfer consiste nos três arquivos a seguir:

- `librarytransfer.xml`
- `librarysave.sh` (<presrc>script executável)
- `libraryrestore.sh` (<postdst>script executável)

Os arquivos de amostra estão localizados no seguinte diretório: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`

IBM i Para executar esta amostra, o usuário deve completar as seguintes etapas:

1. Inicie uma sessão Qshell. Em um tipo de janela de comando IBM i : STRQSH
2. Mude o diretório para o diretório `bin` da seguinte forma:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Depois de concluir a configuração necessária, execute a amostra usando o seguinte comando:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer (plataforma IBM i apenas)

IBM i A amostra physicalfiletransfer demonstra como usar as tarefas Ant para transferir um arquivo Físico de origem ou de Banco de dados de uma biblioteca em um sistema IBM i para uma biblioteca em um segundo sistema IBM i.

IBM i A amostra physicalfiletransfer usa o suporte de arquivo de salvamento nativo no IBM i com as tarefas Ant predefinidas disponíveis no Managed File Transfer para transferir arquivos Físicos de origem e de Banco de dados completos entre dois sistemas IBM i. A amostra usa um elemento < presrc> aninhado em uma tarefa de cópia de arquivo Managed File Transfer para chamar um script executável `physicalfilesave.sh` para salvar o arquivo Físico ou de Banco de Dados de Origem solicitado de uma biblioteca no sistema do agente de origem em um arquivo de salvamento temporário. O arquivo de salvamento é movido pela tarefa ant filecopy para o sistema do agente de destino no qual um elemento < postdst> aninhado é usado para chamar o script executável `physicalfilerestore.sh` , em seguida,

restaura o objeto de arquivo dentro do arquivo de salvamento em uma biblioteca especificada no sistema de destino.

IBM i Antes de executar esta amostra, você deve concluir alguma configuração conforme descrito no arquivo `physicalfiletransfer.xml`. Você também deve ter um ambiente de trabalho do Managed File Transfer em dois sistemas IBM i. A configuração deve consistir em um agente de origem em execução no primeiro sistema IBM i e em um agente de destino em execução no segundo sistema IBM i. Os dois agentes devem ser capazes de se comunicar.

IBM i A amostra physicalfiletransfer consiste nos três arquivos a seguir:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (<presrc>script executável)
- `physicalfilerestore.sh` (<postdst>script executável)

Os arquivos de amostra estão localizados no seguinte diretório: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i Para executar esta amostra, o usuário deve completar as seguintes etapas:

1. Inicie uma sessão Qshell. Em um tipo de janela de comando IBM i : STRQSH
2. Mude o diretório para o diretório `bin` da seguinte forma:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Depois de concluir a configuração necessária, execute a amostra usando o seguinte comando:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

tempo limite

A amostra timeout demonstra como usar tarefas Ant para tentar uma transferência de arquivos e para cancelar a transferência, se ela demorar mais que um valor de tempo limite especificado. O script iniciará uma transferência de arquivos usando a tarefa Managed File Transferfte: filecopy . O resultado desta transferência é adiado. O script usa a tarefa Managed File Transfer fte: awaitoutcome Ant para aguardar um determinado número de segundos para a conclusão da transferência. Se a transferência não for concluída no tempo determinado, a tarefa Managed File Transfer fte:cancel Ant é usada para cancelar a transferência de arquivos.

vsamtransfer

z/OS

z/OS A amostra vsamtransfer demonstra como usar as tarefas Ant para transferir de um conjunto de dados VSAM para outro conjunto de dados VSAM usando o Managed File Transfer. No momento, o Managed File Transfer não suporta a transferência de conjuntos de dados VSAM. O script de amostra descarrega os registros de dados VSAM para um conjunto de dados sequenciais usando o `presrc` Elementos aninhados de chamada de programa para chamar o arquivo executável `datasetcopy.sh`. O script usa a tarefa Managed File Transferfte: filemove para transferir o conjunto de dados sequenciais a partir do agente de origem para o agente de destino. O script então usa os `postdst` Elementos aninhados de chamada de programa para chamar o script `loadvsam.jcl`. Esse script JCL carrega os registros do conjunto de dados transferido em um conjunto de dados VSAM de destino. Esta amostra usa JCL para a chamada de destino para demonstrar esta opção de linguagem. O mesmo resultado também pode ser obtido usando em vez disso um segundo shell script.

z/OS Essa amostra não requer que os conjuntos de dados de origem e destino sejam VSAM. A amostra funciona para quaisquer conjuntos de dados, se os conjuntos de dados de origem e de destino forem do mesmo tipo.

z/OS Para que esta amostra funcione corretamente, você deve colocar o script `datasetcopy.sh` no caminho de comando do agente de origem e o script `loadvsam.jcl` no caminho de comando do agente de destino. Para obter mais informações sobre como configurar o caminho de comando de um agente, consulte Propriedade commandPath MFT.

Zip

A amostra zip é formada por dois scripts: `zip.xml` e `zipfiles.xml`. A amostra demonstra como usor o `presrc` elemento aninhado dentro da tarefa Managed File Transfer fte: filemove para executar um script Ant antes de executar uma operação de movimentação de transferência de arquivos. O script `zipfiles.xml` chamado pelo elemento aninhado `presrc` no script `zip.xml` comprime o conteúdo de um diretório. O script `zip.xml` transfere o arquivo compactado. Esta amostra requer que o script `zipfiles.xml` Ant esteja presente no caminho de comando do agente de origem. Isso porque o script `zipfiles.xml` Ant contém o destino usado para comprimir os conteúdos do diretório no agente de origem. Para obter mais informações sobre como configurar o caminho de comando de um agente, consulte Propriedade commandPath MFT.

Conceitos relacionados

“Introdução ao uso de scripts Ant com MFT” na página 1236
O uso de scripts Ant com o Managed File Transfer permite coordenar operações complexas de transferência de arquivos por meio de uma linguagem de script interpretada.

Referências relacionadas

`fteAnt`: executar tarefas Ant no MFT

Customizando o MFT com saídas de usuário

É possível customizar os recursos do Managed File Transfer usando seus próprios programas conhecidos como rotinas de saída do usuário.

Importante: Qualquer código dentro de uma saída de usuário não é suportado pelo IBMe quaisquer problemas com esse código precisam ser investigados inicialmente por sua empresa ou pelo fornecedor que forneceu a saída.

O Managed File Transfer fornece pontos no código em que o Managed File Transfer pode passar o controle para um programa que foi gravado (uma rotina de saída do usuário). Esses pontos são conhecidos como pontos de saída de usuário. O Managed File Transfer pode então retomar o controle quando o programa tiver concluído seu trabalho. Não é necessário usar nenhuma das saídas de usuário, mas elas são úteis se você desejar estender e customizar a função do sistema Managed File Transfer para atender a requisitos específicos.

Há dois pontos durante o processamento de transferência de arquivos em que é possível chamar uma saída de usuário no sistema de origem e dois pontos durante o processamento de transferência de arquivos em que é possível chamar uma saída de usuário no sistema de destino. A tabela a seguir resume cada um desses pontos de saída de usuário e a interface Java que deve ser implementada para usar os pontos de saída.

Tabela 187. Resumo de pontos de saída do lado da origem e do lado do destino e as interfaces Java

Ponto de saída	**Interface Java a ser implementada**
Pontos de saída do lado de origem:	
Antes que a transferência do arquivo inteiro inicie	Interface SourceTransferStartExit.java
Depois da conclusão de uma transferência do arquivo inteiro	Interface SourceTransferEndExit.java

Tabela 187. Resumo de pontos de saída do lado da origem e do lado do destino e as interfaces Java (continuação)	
Ponto de saída	**Interface Java a ser implementada**
Pontos de saída do lado de destino:	
Antes que a transferência do arquivo inteiro inicie	Interface DestinationTransferStartExit.java
Depois da conclusão de uma transferência do arquivo inteiro	Interface DestinationTransferEndExit.java

As saídas de usuário são chamadas na seguinte ordem:

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

As alterações feitas pelas saídas SourceTransferStartExit e DestinationTransferStartExit são propagadas como entrada para as saídas subsequentes. Por exemplo, se a saída SourceTransferStartExit modificar os metadados de transferência, as alterações são refletidas no metadados de transferência de entrada para outras saídas.

As saídas de usuário e chamadas de programas são chamadas na seguinte ordem:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Notas:

1. O **DestinationTransferEndExits** é executado apenas quando a transferência é concluída, com sucesso ou parcialmente com sucesso.
2. O **postDestinationCall** é executado apenas quando a transferência é concluída, com sucesso ou parcialmente com sucesso.
3. O **SourceTransferEndExits** é executado para transferências bem-sucedidas, parcialmente bem-sucedidas ou com falha.
4. O **postSourceCall** será chamado apenas se:
   - A transferência não foi cancelada.
   - Há um resultado bem-sucedido ou parcialmente bem-sucedido.
   - Quaisquer programas de transferência pós-destino foram executados com sucesso.

Criando a Saída de Usuário

As interfaces para construir uma saída de usuário estão contidas em *MQ_INSTALL_DIRECTORY*/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar. Deve-se incluir este arquivo .jar no caminho da classe quando construir sua saída. Para executar a saída, extraia a saída como um arquivo .jar e coloque este arquivo .jar em um diretório, conforme descrito na seção a seguir.

Locais de Saída de Usuário

É possível armazenar rotinas de saída de usuário em dois locais possíveis:

- O diretório `exits`. Há um diretório exits debaixo de cada diretório de agente. Por exemplo:`var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits`
- É possível configurar a propriedade exitClassPath para especificar um local alternativo. Se houver classes de saída no diretório `exits` e no caminho de classe configurado pelo exitClassPath, as classes no diretório `exits` têm prioridade, o que significa que se houver classes em ambos os locais com o mesmo nome, as classes no diretório `exits` têm prioridade.

Configurando um Agente para Usar Saídas de Usuário

Existem quatro propriedades do agente que podem ser configuradas para especificar as saídas de usuário chamadas por um agente. Essas propriedades do agente são `sourceTransferStartExitClasses`, `sourceTransferEndExitClasses`, `destinationTransferStartExitClasses` e `destinationTransferEndExitClasses`. Para obter informações sobre como usar essas propriedades, consulte Propriedades do agente do MFT para saídas de usuário.

Executando saídas de usuário em agentes de ponte de protocolo

Quando o agente de origem chama a saída, ele passa para a saída uma lista dos itens de origem para a transferência. Para agentes normais, esta é uma lista de nomes de arquivo completos. Como os arquivos devem ser locais (ou acessíveis por meio de uma montagem), então, a saída será capaz de acessá-la e criptografá-la.

No entanto, para um Agente de ponte de protocolo, as entradas na lista têm o formato a seguir:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

Para cada entrada na lista, a saída precisa se conectar ao servidor de arquivos primeiro (usando o FTP. Protocolos FTPS ou SFTP), faça download do arquivo, criptografe-o localmente e, em seguida, faça upload do arquivo criptografado de volta para o servidor de arquivos.

Executando saídas de usuário nos agentes ponte Connect:Direct

Não é possível executar saídas de usuário em agentes ponte Connect:Direct.

Conceitos relacionados

“Saídas de usuário de origem e destino do MFT” na página 1242

Metadados para saídas de usuário do MFT

Interfaces Java para saídas de usuário do MFT

Referências relacionadas

“Ativando a depuração remota para saídas de usuário do MFT” na página 1247
Enquanto você está desenvolvendo suas saídas de usuário, pode-se querer usar um depurador para ajudar a localizar problemas em seu código.

“Amostra da saída de usuário da transferência de origem do MFT” na página 1248

“Saída do usuário da credencial de ponte do protocolo de amostra” na página 1249

Saídas de usuário do monitor de recurso do MFT

Propriedades do agente MFT para saídas de usuário

Saídas de usuário de origem e destino do MFT

Separadores de Diretório

Separadores de diretórios em especificações de arquivo de origem são sempre representados usando caracteres de barra (/), independentemente de como você especificou separadores de diretórios no comando **`fteCreateTransfer`** ou no IBM MQ Explorer. Deve-se levar isso em conta quando você escrever uma saída. Por exemplo, se desejar verificar se o arquivo de origem a seguir existe: `c:\a\b.txt` e tiver especificado esse arquivo de origem usando o comando **`fteCreateTransfer`** ou o IBM MQ

Explorer, observe que o nome do arquivo está, na verdade, armazenado como: `c:/a/b.txt`. Portanto, se você procurar pela sequência original de `c:\a\b.txt`, não localizará uma correspondência.

Pontos de Saída do Lado da Origem

Antes que a transferência do arquivo inteiro inicie

Essa saída é chamada pelo agente de origem quando um pedido de transferência é a próxima na lista de transferências pendentes e a transferência está prestes a iniciar.

Os usos de exemplo deste ponto de saída são para enviar arquivos em estágios para um diretório ao qual o agente possui acesso de leitura/gravação usando um comando externo ou para renomear os arquivos no sistema de destino.

Transfira os seguintes argumentos para essa saída:

- Nome do agente de origem
- Nome do agente de destino
- Metadados do Ambiente
- Metadados de Transferência
- Especificações de arquivo (incluindo os metadados de arquivos)

Os dados retornados dessa saída são os que seguem:

- Metadados de transferência atualizados. As entradas podem ser incluídas, modificadas e excluídas.
- A lista atualizada de especificações de arquivo, composta dos pares nome de arquivo de origem e nome do arquivo de destino. As entradas podem ser incluídas, modificadas e excluídas.
- Indicador que especifica se a transferência continuará
- Cadeia a ser inserida no Log de Transferência.

Implemente a interface SourceTransferStartExit.java para chamar o código de saída do usuário nesse ponto de saída.

Depois da conclusão de uma transferência do arquivo inteiro

Essa saída é chamada pelo agente de origem depois da conclusão da transferência do arquivo inteiro.

Um exemplo de uso desse ponto de saída é executar algumas tarefas de conclusão, como enviar e-mail ou uma mensagem do IBM MQ para sinalizar que a transferência foi concluída.

Transfira os seguintes argumentos para essa saída:

- Resultado da saída da transferência
- Nome do agente de origem
- Nome do agente de destino
- Metadados do Ambiente
- Metadados de Transferência
- Resultados do arquivo

Os dados retornados dessa saída são os que seguem:

- Cadeias atualizadas a serem inseridas no Log de Transferência.

Implemente a interface SourceTransferEndExit.java para chamar o código de saída do usuário nesse ponto de saída.

Pontos de Saída do Lado de Destino

Antes que a transferência do arquivo inteiro inicie

Um exemplo de uso desse ponto de saída é validar as permissões no destino.

Transfira os seguintes argumentos para essa saída:

- Nome do agente de origem
- Nome do agente de destino
- Metadados do Ambiente
- Metadados de Transferência
- Especificações de arquivos

Os dados retornados dessa saída são os que seguem:

- Conjunto de nomes de arquivos de destino atualizados. As entradas podem ser modificadas, mas não incluídas ou excluídas.
- Indicador que especifica se a transferência continuará
- Cadeia a ser inserida no Log de Transferência.

Implemente a interface DestinationTransferStartExit.java para chamar o código de saída do usuário nesse ponto de saída.

Depois da conclusão de uma transferência do arquivo inteiro

Um exemplo do uso dessa saída de usuário é iniciar um processo em lote que use os arquivos transferidos ou envie um email se a transferência falhar.

Transfira os seguintes argumentos para essa saída:

- Resultado da saída da transferência
- Nome do agente de origem
- Nome do agente de destino
- Metadados do Ambiente
- Metadados de Transferência
- Resultados do arquivo

Os dados retornados dessa saída são os que seguem:

- Cadeias atualizadas a serem inseridas no Log de Transferência.

Implemente a interface DestinationTransferEndExit.java para chamar o código de saída de usuário nesse ponto de saída.

Conceitos relacionados

Interfaces Java para saídas de usuário do MFT

Referências relacionadas

“Ativando a depuração remota para saídas de usuário do MFT” na página 1247
Enquanto você está desenvolvendo suas saídas de usuário, pode-se querer usar um depurador para ajudar a localizar problemas em seu código.

“Amostra da saída de usuário da transferência de origem do MFT” na página 1248

Saídas de usuário do monitor de recurso do MFT

Usando as saídas de usuário de E/S de transferência do MFT

É possível usar saídas de usuário de E/S de transferência do Managed File Transfer para configurar código customizado para executar o trabalho de E/S do sistema de arquivos subjacente para transferências do Managed File Transfer.

Geralmente, para transferências do MFT, um agente seleciona entre um dos provedores de E/S integrados para interagir com os sistemas de arquivos apropriados para a transferência. Os provedores de E/S integrados suportam as seguintes etapas do sistema de arquivos:

- Sistemas de arquivos regulares do tipo UNIX e do tipo Windows
- z/OS Conjuntos de dados sequenciais e particionados do z/OS (no z/OS somente)

- IBM i Arquivos salvos nativos do IBM i (no IBM i somente)
- Filas do IBM MQ
- Servidores de protocolo remotos FTP e SFTP (apenas para agentes de ponte de protocolo)
- Nós remotos do Connect:Direct (para agentes de ponte do Connect:Direct somente)

Para sistemas de arquivos que não são suportados, ou onde você precisa de comportamento de E/S, é possível gravar uma saída de usuário de E/S de transferência.

As saídas de usuário de E/S de transferência usam a infraestrutura existente para saídas de usuário. No entanto, estas saídas de usuário de E/S de transferência se diferem de outras saídas de usuário porque sua função é acessada diversas vezes durante a transferência de cada arquivo.

Use a propriedade do agente IOExitClasses (no arquivo `agent.properties`) para especificar quais classes de saída de E/S a carregar. Separe cada classe de saída com uma vírgula, por exemplo:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

As interfaces Java para as saídas de usuário de E/S de transferência são conforme a seguir:

IOExit
O ponto de entrada principal usado para determinar se a saída de E/S é usada. Esta instância é responsável por criar instâncias IOExitPath.

É necessário especificar apenas a interface de saída de E/S IOExit para a propriedade do agente IOExitClasses.

IOExitPath
Representa uma interface abstrata; por exemplo, um contêiner de dados ou curinga que representa um conjunto de contêineres de dados. Não é possível criar uma instância de classe que implementa esta interface. A interface permite que o caminho seja examinado e caminhos derivados sejam listados. As interfaces IOExitResourcePath e IOExitWildcardPath estendem IOExitPath.

IOExitChannel
Permite que dados sejam lidos ou gravados em um recurso IOExitPath.

IOExitRecordChannel
Estende a interface IOExitChannel para recursos IOExitPath orientados a registros, que permite que dados sejam lidos ou gravados em um recurso IOExitPath em diversos registros.

IOExitLock
Representa um bloqueio em um recurso IOExitPath para acesso compartilhado ou exclusivo.

z/OS **IOExitRecordResourcePath**
Estende a interface IOExitResourcePath para representar um contêiner de dados para um arquivo orientado a registros; por exemplo, um conjunto de dados do z/OS. É possível usar a interface para localizar dados e criar instâncias IOExitRecordChannel para operações de leitura ou gravação.

IOExitResourcePath
Estende a interface IOExitPath para representar um contêiner de dados; por exemplo, um arquivo ou diretório. É possível usar a interface para localizar dados. Se a interface representar um diretório, será possível usar o método listPaths para retornar uma lista de caminhos.

IOExitWildcardPath
Estende a interface IOExitPath para representar um caminho que indica um curinga. É possível usar esta interface para corresponder a diversos IOExitResourcePaths.

IOExitProperties
Especifica propriedades que determinam como Managed File Transfer manipula IOExitPath para certos aspectos de E/S. Por exemplo, se deve usar arquivos intermediários ou se deve reler um recurso desde o início se uma transferência for reiniciada.

Conceitos relacionados

"Customizando o MFT com saídas de usuário" na página 1240
É possível customizar os recursos do Managed File Transfer usando seus próprios programas conhecidos como rotinas de saída do usuário.

Referências relacionadas

Interface IOExit.java
Interface IOExitChannel.java
Interface IOExitLock.java
Interface IOExitPath.java
Interface IOExitProperties.java
Interface IOExitRecordChannel.java
z/OS Interface IOExitRecordResourcePath.java
Interface IOExitResourcePath.java
Interface IOExitWildcardPath.java
O arquivo MFT `agent.properties`

IBM i Saídas de usuário de amostra do MFT no IBM i

O Managed File Transfer fornece saídas de usuário de amostra específicas do IBM i com a instalação. As amostras estão nos diretórios *MQMFT_install_dir*`/samples/ioexit-IBMi` e *MQMFT_install_dir*`/samples/userexit-IBMi`.

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit
A saída de usuário de amostra com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit transfere arquivos no sistema de arquivos QDLS no IBM i. Após a instalação da saída, quaisquer transferências para arquivos que começam com `/QDLS` utilizam automaticamente a saída.

Para instalar essa saída, conclua as seguintes etapas:

1. Copie o arquivo `com.ibm.wmqfte.samples.ibmi.ioexits.jar` do diretório *WMQFTE_install_dir*`/samples/ioexit-IBMi` para o diretório `exits` do agente.
2. Inclua com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit na propriedade IOExitClasses.
3. Reinicie o agente.

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit
A saída de usuário de amostra com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit comporta-se como um monitor de arquivo do MFT e transfere automaticamente membros de arquivo físico de uma biblioteca do IBM i.

Para executar essa saída, especifique um valor para o campo de metadados "library.qsys.monitor" (usando o parâmetro **-md**, por exemplo). Esse parâmetro leva um caminho do estilo IFS até um membro do arquivo e pode conter curingas de arquivo e de membro. Por exemplo, /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.FILE/*.MBR.

Essa saída de amostra também contém um campo de metadados opcional "naming.scheme.qsys.monitor", que pode ser usado para determinar o esquema de nomenclatura usado durante a transferência. Por padrão, este campo é configurado como "unix,", o que faz com que o arquivo de destino seja chamado `F00.MBR`. Também é possível especificar o valor "ibmi" para usar o FILE IBM i FTP FILE.MEMBER , por exemplo, /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR é transferido como BAR.BAZ.

Para instalar essa saída, conclua as seguintes etapas:

1. Copie o arquivo `com.ibm.wmqfte.samples.ibmi.userexits.jar` do diretório *WMQFTE_install_dir*`/samples/userexit-IBMi` para o diretório `exits` do agente.
2. Inclua com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit na propriedade sourceTransferStartExitClasses no arquivo `agent.properties`.
3. Reinicie o agente.

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

A saída de usuário de amostra com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit exclui um objeto de arquivo vazio quando o membro do arquivo de origem é excluído como parte da transferência. Como os objetos de arquivo do IBM i potencialmente podem reter muitos membros, os objetos de arquivo são tratados como diretórios pelo MFT. Portanto, não é possível executar uma operação de movimentação em um objeto de arquivo usando o MFT; as operações de movimentação são suportadas somente no nível do membro. Consequentemente, ao executar uma operação de movimentação em um membro, o arquivo agora vazio é deixado de lado. Use essa saída de amostra se desejar excluir esses arquivos vazios como parte da solicitação de transferência.

Se você especificar "true" para os metadados "empty.file.delete" e transferir um FTEFileMember, a saída de amostra excluirá o arquivo-pai, caso o arquivo esteja vazio.

Para instalar essa saída, conclua as seguintes etapas:

1. Copie o arquivo com.ibm.wmqfte.samples.ibmi.userexits.jar de *WMQFTE_install_dir*`/samples/userexit-IBMi` para o diretório `exits` do agente.
2. Inclua com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit na propriedade sourceTransferStartExitClasses no arquivo `agent.properties`.
3. Reinicie o agente.

Referências relacionadas

“Usando as saídas de usuário de E/S de transferência do MFT” na página 1244
É possível usar saídas de usuário de E/S de transferência do Managed File Transfer para configurar código customizado para executar o trabalho de E/S do sistema de arquivos subjacente para transferências do Managed File Transfer.

Propriedades do agente MFT para saídas de usuário

Ativando a depuração remota para saídas de usuário do MFT

Enquanto você está desenvolvendo suas saídas de usuário, pode-se querer usar um depurador para ajudar a localizar problemas em seu código.

Como as saídas são executadas dentro da Java virtual machine que executa o agente, não é possível usar o suporte de depuração direta que geralmente está incluído em um ambiente de desenvolvimento integrado. No entanto, é possível ativar a depuração remota da JVM e, em seguida, conectar um depurador remoto apropriado.

Para ativar a depuração remota, use os parâmetros padrão da JVM **-Xdebug** e **-Xrunjdwp**.
Essas propriedades são passadas para a JVM que executa o agente pela variável de ambiente **BFG_JVM_PROPERTIES**. Por exemplo, no AIX and Linux os comandos a seguir iniciam o agente e fazem com que a JVM receba conexões do depurador na porta TCP 8765.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

O agente não inicia até que o depurador seja conectado. Use o comando **set** no Windows em vez do comando **export**.

Você também pode usar outros métodos de comunicação entre o depurador e o JVM. Por exemplo, o JVM pode abrir a conexão com o depurador ao invés do contrário, ou você pode usar memória compartilhada ao invés de TCP. Veja a documentação do Java Platform Debugger Architecture para obter detalhes adicionais.

Você deve usar o parâmetro **-F** (primeiro plano) quando iniciar o agente no modo de depuração remoto.

Usando o depurador Eclipse

As etapas a seguir se aplicam ao recurso de depuração remota no ambiente de desenvolvimento Eclipse. É possível também usar outros depuradores remotos que sejam compatíveis com JPDA.

1. Clique em **Executar** > **Abrir Diálogo de Depuração** (ou **Executar** > **Depurar Configurações** ou **Executar** > **Depurar Diálogo** dependendo de sua versão do Eclipse).
2. Clique duas vezes em **Aplicativo Java Remoto** na lista de tipos de configuração para criar uma configuração de depuração.
3. Complete os campos de configuração e salve a configuração de depuração. Se você já tiver iniciado o agente JVM em modo de depuração, é possível se conectar ao JVM agora.

Amostra da saída de usuário da transferência de origem do MFT

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {
```

```
            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

Saída do usuário da credencial de ponte do protocolo de amostra

Para obter informações sobre como usar essa saída de usuário de amostra, consulte Mapeando credenciais para um servidor de arquivos usando classes de saída.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {
```

```
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
                 inputStream = new FileInputStream(propertiesFile);

                 // Load the properties from the file
                 mappingProperties.load(inputStream);
             }
             catch (FileNotFoundException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                 initialisationResult = false;
             }
             catch (IOException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                 initialisationResult = false;
             }
             finally {
                 // Close the inputStream
                 if (inputStream != null) {
                     try {
                         inputStream.close();
                     }
                     catch (IOException ex) {
                         System.err.println("Error initializing SampleCredentialExit.");
                          System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                         initialisationResult = false;
                     }
                 }
             }

             if (initialisationResult) {
                 // Populate the map of mqUserId to server credentials from the properties
                 final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                  while ( propertyNames.hasMoreElements()) {
                     final Object name = propertyNames.nextElement();
                     if (name instanceof String ) {
                         final String mqUserId = ((String)name).trim();
                          // Get the value and split into serverUserId and serverPassword
                        final String value = mappingProperties.getProperty(mqUserId);
                        final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                         String serverUserId = "";
                         String serverPassword = "";
                         if (valueTokenizer.hasMoreTokens()) {
                             serverUserId = valueTokenizer.nextToken().trim();
                         }
                         if (valueTokenizer.hasMoreTokens()) {
                             serverPassword = valueTokenizer.nextToken().trim();
                         }
                         // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                         // Insert the credentials into the map
                         credentialsMap.put(mqUserId, credentials);
```

```
                    }
                }

            }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

Saída de Usuário de Propriedades da Ponte de Protocolo de Amostra

Para obter informações sobre como usar essa saída de usuário de amostra, consulte ProtocolBridgePropertiesExit2: consultando as propriedades do servidor de arquivos de protocolo

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
```

```
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
```

```
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }
```

```
    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

Controlando o MFT Colocando Mensagens na Fila de Comandos do Agente

É possível gravar um aplicativo que controla o Managed File Transfer colocando mensagens em filas de comandos do agente.

É possível colocar uma mensagem na fila de comandos de um agente para solicitar que o agente execute uma das seguintes ações:

- Criar uma transferência de arquivos
- Criar uma transferência de arquivos planejada
- Cancelar uma transferência de arquivos
- Cancelar uma transferência de arquivos planejada
- Chamar um comando
- Criar um monitor
- Excluir um monitor
- Retornar um ping para indicar que o agente está ativo

Para solicitar que o agente execute uma destas ações, a mensagem deve estar em um formato XML que esteja em conformidade com um dos seguintes esquemas:

FileTransfer.xsd
: As mensagens neste formato podem ser usadas para criar uma transferência de arquivos ou uma transferência de arquivos planejada, chamar um comando ou cancelar uma transferência de arquivos ou uma transferência de arquivos planejada. Para obter mais informações, consulte Formato da mensagem de solicitação de transferência de arquivos.

Monitor.xsd
: Mensagens neste formato podem ser usadas para criar ou excluir um monitor de recurso. Para obter mais informações, consulte Formatos de mensagens de solicitação de monitor do MFT.

PingAgent.xsd
: Mensagens neste formato podem ser usadas para executar ping em um agente para verificar se ele está ativo. Para obter mais informações, consulte Formato da mensagem de solicitação para executar ping do agente do MFT.

O agente retorna uma resposta às mensagens de solicitação. A mensagem de resposta é colocada em uma fila de resposta que está definida na mensagem de solicitação. A mensagem de resposta está em um formato XML definido pelo seguinte esquema:

Reply.xsd
: Para obter mais informações, consulte Formato da mensagem de resposta do agente do MFT.

Desenvolvendo aplicativos para o MQ Telemetry

aplicativos de telemetria integram dispositivos de sentido e controle a outras fontes de informações disponíveis na Internet e nas empresas.

Desenvolva aplicativos para o MQ Telemetry usando padrões, exemplos trabalhados, programas de amostra, conceitos de programação e informações de referência.

Conceitos relacionados

MQ Telemetry

Casos de Uso de Telemetria

Tarefas relacionadas

Instalando MQ Telemetry

Administrando MQ Telemetry

Resolução de problemas do MQ Telemetry ..

Referências relacionadas

Referência do MQ Telemetry

Programas de amostra doIBM MQ Telemetry Transport

Os scripts de amostra são fornecidos que funcionam com uma amostra IBM MQ Telemetry Transport aplicativo cliente v3 (`mqttv3app.jar`). Para IBM MQ 8.0.0 e posterior, o aplicativo cliente de amostra não é mais incluído no MQ Telemetry. Ele fazia parte do IBM Messaging Telemetry Clients SupportPac (não está mais disponível). Os aplicativos de amostra semelhantes continuam disponíveis gratuitamente no Eclipse Paho e MQTT.org.

Para obter as informações e os downloads mais recentes, consulte os recursos a seguir:

- O projeto Eclipse Paho e o MQTT.org têm downloads, sem custo, dos clientes de telemetria e amostras mais recentes para uma gama de linguagens de programação. Use esses sites para ajudar você a desenvolver programas de amostra para publicação e assinatura do IBM MQ Telemetry Transport e para inclusão de recursos de segurança.
- O IBM Messaging Telemetry Clients SupportPac não está mais disponível para download. Se você tiver uma cópia transferida por download anteriormente, ela terá os conteúdos a seguir:
  - A versão MA9B do IBM Messaging Telemetry Clients SupportPac incluía um aplicativo de amostra compilado (`mqttv3app.jar`) e a biblioteca do cliente associada (`mqttv3.jar`). Eles eram fornecidos nos diretórios a seguir:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - Na versão MA9C desse SupportPac, o diretório e os conteúdos do `/SDK/` foram removidos:
    - Somente a origem para o aplicativo de amostra (`mqttv3app.jar`) foi fornecida. Ela estava neste diretório:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - A biblioteca do cliente compilada ainda era fornecida. Ela estava neste diretório:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Se você ainda tiver uma cópia do IBM Messaging Telemetry Clients SupportPac (não está mais disponível), informações sobre como instalar e executar o aplicativo de amostra serão fornecidas em Verificando a instalação do MQ Telemetry usando a linha de comandos.

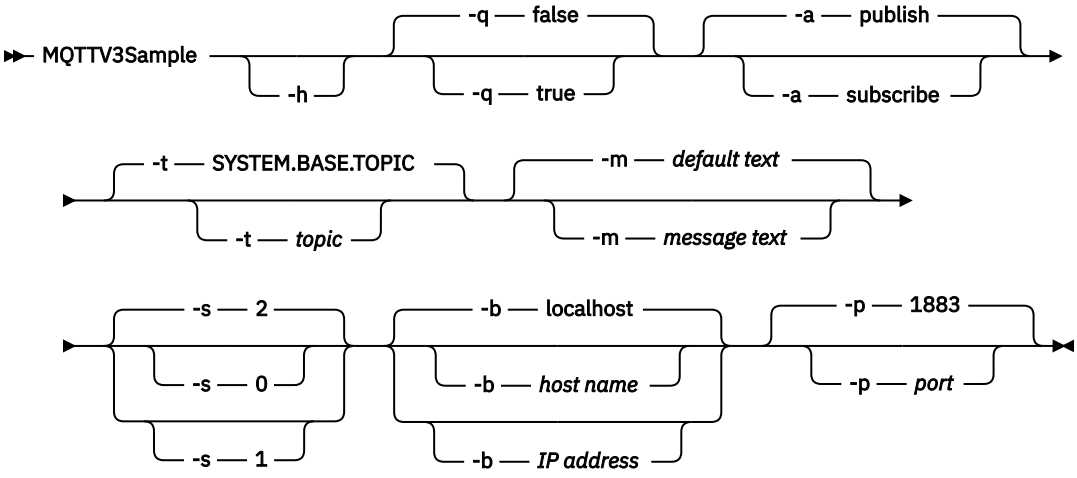
Programa MQTTV3Sample

Informações de referência sobre a sintaxe de amostra e os parâmetros para o programa `MQTTV3Sample`.

Propósito

O programa `MQTTV3Sample` pode ser usado para publicar uma mensagem e assinar um tópico. Para obter informações sobre como obter esse programa de amostra, consulte “Programas de amostra doIBM MQ Telemetry Transport” na página 1255.

MQTTV3Sample syntax



Parâmetros

-h
Imprimir este texto de ajuda e sair

-q
Configure o modo silencioso em vez de usar o modo false padrão.

-a
Configure publicar ou assinar em vez de assumir a ação padrão de publicação.

-t
Publique ou assine o tópico em vez de publicar ou assinar o tópico padrão

-m
Publicar o texto da mensagem em vez de enviar o texto de publicação padrão, "Hello from an MQTT v3 application".

-s
Configure o QoS em vez de usar o QoS padrão, 2.

-b
Conectar a esse nome do host ou endereço IP em vez de conectar ao nome do host padrão, host local.

-p
Use essa porta em vez de usar o padrão, 1883.

Execute o programa MQTTV3Sample

Para assinar um tópico em Windows, use o comando:

```
run MQTTV3Sample -a subscribe
```

Para publicar uma mensagem no Windows, use o comando:

```
run MQTTV3Sample
```

Conceitos de programação do clienteMQTT

Os conceitos descritos nesta seção o ajudam a entender as bibliotecas do cliente para o MQTT protocol. Os conceitos complementam a documentação da API que acompanha as bibliotecas do cliente.

Para obter as informações e os downloads mais recentes, consulte os recursos a seguir:

- O projeto Eclipse Paho e o MQTT.org têm downloads, sem custo, dos clientes de telemetria e amostras mais recentes para uma gama de linguagens de programação. Use esses sites para ajudar você a desenvolver programas de amostra para publicação e assinatura do IBM MQ Telemetry Transport e para inclusão de recursos de segurança.
- O IBM Messaging Telemetry Clients SupportPac não está mais disponível para download. Se você tiver uma cópia transferida por download anteriormente, ela terá os conteúdos a seguir:
  - A versão MA9B do IBM Messaging Telemetry Clients SupportPac incluía um aplicativo de amostra compilado (`mqttv3app.jar`) e a biblioteca do cliente associada (`mqttv3.jar`). Eles eram fornecidos nos diretórios a seguir:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - Na versão MA9C desse SupportPac, o diretório e os conteúdos do `/SDK/` foram removidos:
    - Somente a origem para o aplicativo de amostra (`mqttv3app.jar`) foi fornecida. Ela estava neste diretório:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - A biblioteca do cliente compilada ainda era fornecida. Ela estava neste diretório:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Para desenvolver e executar um cliente MQTT, é necessário copiar ou instalar esses recursos no dispositivo do cliente. Não é necessário instalar um tempo de execução de cliente separado.

As condições de licenciamento para clientes são associadas ao servidor ao qual você está conectando os clientes.

As bibliotecas do cliente do MQTT são implementações de referência do MQTT protocol. É possível implementar seus próprios clientes em diferentes idiomas apropriados para diferentes plataformas de dispositivo. Consulte IBM MQ Telemetry Transport formato e protocolo.

A documentação da API não faz suposições sobre qual servidor MQTT o cliente está conectado. O comportamento do cliente poderá ser um pouco diferente quando conectado a diferentes servidores. As descrições que seguem descrevem o comportamento do cliente quando conectado ao serviço de telemetria do IBM MQ.

Retornos de chamadas e sincronização em aplicativos do cliente MQTT

O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Retornos de chamada

Nota: Consulte o website do Eclipse Paho para obter as mudanças mais recentes para `MqttCallback` Por exemplo `MqttCallback` é definido como uma Interface na versão Paho do cliente e métodos assíncronos são fornecidos pela classe Paho `MqttAsyncClient`.

A interface `MqttCallback` tem três métodos de retorno de chamada:

connectionLost(java.lang.Throwable cause)

`connectionLost` será chamado quando um erro de comunicação levar ao descarte da conexão. Ele também será chamado, se o servidor descartar a conexão como resultado de um erro no servidor após a conexão ter sido estabelecida. Os erros do servidor são registrados no log de erros do gerenciador de filas. O servidor descarta a conexão com o cliente e o cliente chama o `MqttCallback.connectionLost`.

Os únicos erros remotos lançados como exceções no mesmo encadeamento que o aplicativo cliente são exceções a partir de `MqttClient.connect`. Os erros detectados pelo servidor após a conexão ser estabelecida serão relatados de volta ao método de retorno de chamada `MqttCallback.connectionLost` como `throwables`.

Os erros típicos do servidor que resultam em `connectionLost` são erros de autorização. Por exemplo, o servidor de telemetria tenta publicar em um tópico em nome de um cliente que não está autorizado a publicar no tópico. Qualquer coisa que resulte no retorno de um código de condição `MQCC_FAIL` ao servidor de telemetria pode resultar no descarte da conexão.

deliveryComplete(IMqttDeliveryToken token)

`deliveryComplete` é chamado pelo cliente MQTT para transmitir um token de entrega de volta ao aplicativo cliente; consulte "Tokens de entrega" na página 1264. Usando o token de entrega, o retorno de chamada pode acessar a mensagem publicada com o método `token.getMessage`.

Quando o retorno de chamada do aplicativo retornar o controle para o cliente MQTT após ser chamado pelo método `deliveryComplete`, a entrega será concluída. Até que a entrega seja concluída, mensagens com QoS 1 ou 2 serão retidas pela classe de persistência.

A chamada para `deliveryComplete` é um ponto de sincronização entre o aplicativo e a classe de persistência. O método `deliveryComplete` nunca é chamado duas vezes para a mesma mensagem.

Quando o retorno de chamada do aplicativo retorna de `deliveryComplete` para o cliente MQTT, o cliente chama `MqttClientPersistence.remove` para mensagens com QoS 1 ou 2. `MqttClientPersistence.remove` exclui a cópia armazenada localmente da mensagem publicada.

De uma perspectiva de processamento de transações, a chamada para `deliveryComplete` é uma transação de fase única que confirma a entrega. Se o processamento falhar durante o retorno de chamada, na reinicialização do cliente, `MqttClientPersistence.remove` será chamado novamente para excluir a cópia local da mensagem publicada. O retorno de chamada não é chamado novamente. Se você estiver usando o retorno de chamada para armazenar um log de mensagens entregues, não será possível sincronizar o log com o cliente MQTT. Se você deseja armazenar um log confiavelmente, então atualize o log na classe `MqttClientPersistence`.

O token de entrega e a mensagem são referenciados pelo encadeamento de aplicativos principal e o cliente MQTT. O cliente MQTT desreferenciará do objeto `MqttMessage` quando a entrega for concluída e o objeto do token de entrega quando o cliente for desconectado. O objeto `MqttMessage` poderá ser lixo coletado após a entrega ser concluída se o aplicativo cliente desreferenciá-lo. O token de entrega poderá ser lixo coletado após a sessão ser desconectada.

É possível obter atributos `IMqttDeliveryToken` e `MqttMessage` após uma mensagem ter sido publicada. Se você tentar configurar quaisquer atributos `MqttMessage` após a mensagem ter sido publicada, o resultado será indefinido.

O cliente MQTT continuará a processar as confirmações de entrega, se o cliente se reconectar à sessão anterior com o mesmo `ClientIdentifier`; consulte "Sessões limpas" na página 1261. O aplicativo cliente MQTT deve configurar `MqttClient.CleanSession` como `false` para a sessão anterior e configurá-lo como `false` na nova sessão. O cliente MQTT cria novos tokens de entrega e objetos de mensagem na nova sessão para as entregas pendentes. Ele recupera os objetos usando a classe `MqttClientPersistence`. Se o aplicativo cliente ainda tiver referências ao antigo tokens de entrega e mensagens, os desreferenciem. O retorno de chamada do aplicativo é chamado na nova sessão para quaisquer entregas iniciada na sessão anterior e concluídas nessa sessão.

O retorno de chamada do aplicativo será chamado depois que o aplicativo cliente se conectar, quando uma entrega pendente for concluída. Antes de o aplicativo cliente se conectar, ele poderá recuperar entregas pendentes usando o método `MqttClient.getPendingDeliveryTokens`.

Observe que o aplicativo cliente criou originalmente o objeto de mensagem publicado e sua matriz de bytes de carga útil. O cliente MQTT referencia esses objetos. O objeto de mensagem retornado pelo token de entrega no método `token.getMessage` não é necessariamente o mesmo objeto de mensagem criado pelo cliente. Se uma nova instância do cliente MQTT recria o token de entrega, a classe `MqttClientPersistence` recria o objeto `MqttMessage`. Para consistência, o `token.getMessage` retornará `null` se o `token.isCompleted` for `true`, independentemente se o objeto de mensagem foi criado pelo aplicativo cliente ou pela classe `MqttClientPersistence`.

`messageArrived(String topic, MqttMessage message)`

`messageArrived` é chamado quando uma publicação chega para o cliente que corresponde a um tópico de subscrição `topic` é o tópico da publicação, não o filtro de assinatura. Os dois poderão ser diferentes se o filtro contiver caracteres curingas.

Se o tópico corresponder a diversas assinaturas criadas pelo cliente, o cliente receberá diversas cópias da publicação. Se um cliente publicar em um tópico que também assina, ele receberá uma cópia da sua própria publicação.

Se uma mensagem for enviada com um QoS de 1 ou 2, a mensagem será armazenada pelo `MqttClientPersistence` de classe antes de o cliente MQTT chamar o `messageArrived`. `messageArrived` se comporta como `deliveryComplete`: é chamado apenas uma vez para uma publicação e a cópia local da publicação será removida por `MqttClientPersistence.remove` quando `messageArrived` retornar ao cliente MQTT. O cliente MQTT descartará suas referências para o tópico e a mensagem quando `messageArrived` retornar ao cliente MQTT. Os objetos de tópicos e mensagens serão o lixo coletado, se o aplicativo cliente não tiver retido em uma referência aos objetos.

Retornos de chamadas, encadeamento e sincronização do aplicativo cliente

O cliente MQTT chama um método de retorno de chamada em um encadeamento separado para o encadeamento de aplicativo principal. O aplicativo cliente não cria um encadeamento para o retorno de chamada, é criado pelo cliente MQTT.

O cliente MQTT sincroniza os métodos de retorno de chamada. Apenas uma instância do método de retorno de chamada é executada por vez. A sincronização torna mais fácil a atualização de um objeto que registra a contagem total que as publicações foram entregues. Uma instância do `MqttCallback.deliveryComplete` é executada por vez, e, portanto, é seguro para atualizar a contagem total sem sincronização adicional. Nesse caso, somente uma publicação chega por vez. Seu código no método `messageArrived` pode atualizar um objeto sem sincronizá-lo. Se você estiver se referindo à contagem total ou ao objeto que está sendo atualizado, em outro encadeamento, sincronize a contagem total ou o objeto.

O token de entrega fornece um mecanismo de sincronização entre o encadeamento principal do aplicativo e a entrega de uma publicação. O método `token.waitForCompletion` aguarda até que a entrega de uma publicação específica seja concluída ou até que um tempo limite opcional expire. É possível usar `token.waitForCompletion` no caminho a seguir para processar uma publicação por vez.

Para sincronizar com o método `MqttCallback.deliveryComplete`. Somente quando o `MqttCallback.deliveryComplete` retornar para o cliente MQTT, `token.waitForCompletion` continuará. Usando esse mecanismo será possível sincronizar a execução de código em `MqttCallback.deliveryComplete` antes de o código ser executado no encadeamento de aplicativos principal.

E se você desejava publicar sem aguardar que cada publicação seja entregue, mas deseja confirmação quando todas as publicações foram entregues? Se você publicar em um único encadeamento, a última publicação a ser enviada será também a última a ser entregue.

Sincronização de solicitações enviadas ao servidor

Tabela 188 na página 1260 descreve os métodos no cliente MQTT Java que envia uma solicitação ao servidor. A menos que o aplicativo cliente configure um tempo limite indefinido, o cliente nunca aguardará

indefinidamente pelo servidor. Se o cliente for interrompido, será um problema de programação de aplicativos ou um defeito no cliente MQTT.

Tabela 188. Comportamento de sincronização de métodos que resultam em solicitações para o servidor

Método	Sincronização	Intervalo de tempo limite
`MqttClient.Connect`	Aguarda para que uma conexão seja estabelecida com o servidor.	Padronizado como 30 segundos ou como configurado por um parâmetro, em seguida, lança uma exceção.
`MqttClient.Disconnect`	Aguarda o cliente MQTT concluir qualquer trabalho que deve ser realizado e a sessão TCP/IP para se desconectar.	
`MqttClient.Subscribe`	Espera pela conclusão do método Subscribe ou UnSubscribe.	
`MqttClient.UnSubscribe`		
`MqttClient.Publish`	Retorna imediatamente para o encadeamento de aplicativos após transmitir a solicitação ao cliente MQTT.	Nenhum.
`IMqttDeliveryToken.waitForCompletion`	Aguarda o token de entrega a ser retornado.	Indefinido ou configurado como um parâmetro.

Conceitos relacionados

Sessões limpas

O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente

O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament

Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT

As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações

Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Sessões limpas

O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Ao conectar um aplicativo cliente MQTT usando o método `MqttClient.connect`, o cliente identifica a conexão usando o identificador de cliente e o endereço do servidor. O servidor verifica se informações da sessão foram salvas de uma conexão anterior no servidor. Se uma sessão anterior ainda existir e `cleanSession=true`, então, as informações da sessão anterior no cliente e no servidor serão limpas. Se `cleanSession=false`, a sessão anterior será continuada. Se não existir nenhuma sessão anterior, uma nova sessão será iniciada.

Nota: O administrador do IBM MQ pode fechar de forma forçada uma sessão aberta e excluir todas as informações de sessão. Se o cliente reabrir a sessão com `cleanSession=false`, uma nova sessão será iniciada.

Publicações

Se você usar o padrão `MqttConnectOptions` ou configurar `MqttConnectOptions.cleanSession` para `true` antes de se conectar ao cliente, todas as entregas de publicação pendentes para o cliente serão removidas quando o cliente se conectar.

A configuração de sessão limpa não tem efeito sobre as publicações enviadas com `QoS=0`. Para `QoS=1` e `QoS=2`, o uso de `cleanSession=true` pode resultar na perda de uma publicação.

Assinaturas

Se você usar o `MqttConnectOptions`padrão ou configurar `MqttConnectOptions.cleanSession` como `true` antes de conectar o cliente, quaisquer assinaturas antigas para o cliente serão removidas quando o cliente se conectar.. Quaisquer novas assinaturas feitas pelo cliente durante a sessão serão removidas quando ele se desconectar.

Se você configurar `MqttConnectOptions.cleanSession` como `false` antes de se conectar, quaisquer assinaturas criadas pelo cliente serão incluídas em todas as assinaturas que existiam para o cliente antes de ele se conectar. Todas as assinaturas permanecem ativas quando o cliente se desconecta.

Outra maneira de entender a maneira como o atributo `cleanSession` afeta assinaturas é pensar nele como um atributo modal. Em seu modo padrão, `cleanSession=true`, o cliente cria assinaturas e recebe publicações apenas dentro do escopo da sessão. No modo alternativo, `cleanSession=false`, assinaturas são duráveis. O cliente pode se conectar e se desconectar e suas assinaturas permanecem ativas. Ao se reconectar, o cliente recebe todas as publicações não entregues. Enquanto estiver conectado, ele pode modificar o conjunto de assinaturas que estão ativas em seu nome.

Deve-se configurar o modo `cleanSession` antes de conectar; o modo dura toda a sessão. Para alterar essa configuração, você deve desconectar e reconectar o cliente. Se você mudar os modos de usar `cleanSession=false` para `cleanSession=true`, todas as assinaturas anteriores para o cliente e quaisquer publicações que não foram recebidas serão descartadas

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT

As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Identificador de Cliente

O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

O identificador do cliente é usado na administração de um sistema MQTT. Com centenas de milhares de clientes em potencial para administrar, é necessário estar apto para identificar rapidamente um cliente específico. Por exemplo, suponha que um dispositivo tenha um mau funcionamento e você seja notificado, talvez por um cliente entrando em contato com a help desk. O cliente precisa ser capaz de identificar o dispositivo e você precisa ser capaz de correlacionar essa identificação com o servidor que está normalmente conectado ao cliente.

Ao navegar pelas conexões do cliente MQTT, cada conexão será rotulada com o identificador do cliente. Para ajudar a decidir a melhor forma de mapear este identificador ao dispositivo e ao servidor, faça a si mesmo as seguintes perguntas:

- Seria conveniente manter e utilizar um banco de dados que mapeia cada dispositivo para um identificador de cliente e para um servidor?
- O nome do dispositivo poderia identificar o servidor ao qual ele está conectado?
- Uma tabela de consulta que mapeia um identificador de cliente a um dispositivo físico é necessária?
- O identificador do cliente identifica um determinado dispositivo, usuário ou aplicativo em execução no cliente?
- Se um cliente substitui um dispositivo defeituoso por um novo, o novo dispositivo tem o mesmo identificador que o dispositivo antigo, ou você aloca um novo identificador? (Se você alterar um dispositivo físico e manter o mesmo identificador, as publicações pendentes e as assinaturas ativas são automaticamente transferidas para o novo dispositivo.)

Também é necessário um sistema para garantir que os identificadores de clientes sejam exclusivos e deve-se ter um processo confiável para configurar o identificador no cliente. Se o dispositivo do cliente for uma "caixa preta", sem interface com o usuário, você poderia fabricar o dispositivo com um identificador de cliente ou poderia ter um processo de instalação e configuração de software que configura o dispositivo antes de ele ser ativado.

Para manter o identificador curto e exclusivo, poderia-se criar um identificador de cliente a partir do endereço MAC do dispositivo de 48 bits. Se o tamanho da transmissão não for uma questão crítica, poderia-se então utilizar os 17 bytes restantes para tornar o endereço mais fácil de administrar.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Tokens de entrega

Quando um cliente publica em um tópico, um novo token de entrega é criado. Use o token de entrega para monitorar a entrega de uma publicação ou para bloquear o aplicativo cliente até que a entrega seja concluída.

O token é um objeto `MqttDeliveryToken`. Ele é criado chamando o método MqttTopic.publish() e é retido pelo cliente MQTT até que a sessão do cliente seja desconectada e a entrega seja concluída.

O uso normal do token é para verificar se a entrega foi concluída. Bloqueie o aplicativo cliente até que a entrega seja concluída usando o token retornado para chamar `token.waitForCompletion`. Como alternativa, forneça um identificador `MqttCallBack`. Quando o cliente MQTT tiver recebido todas as confirmações esperadas como parte da entrega da publicação, ele chamará `MqttCallBack.deliveryComplete` transmitindo o token de entrega como um parâmetro.

Até a entrega ser concluída, será possível inspecionar a publicação usando o token de entrega retornado chamando `token.getMessage`.

Entregas concluídas
A conclusão de entregas é assíncrona e depende da qualidade de serviço associada à publicação.

No máximo uma vez

`QoS=0`

A entrega é concluída imediatamente no retorno de `MqttTopic.publish`. `MqttCallback.deliveryComplete` é chamado imediatamente.

Pelo menos uma vez

QoS=1
A entrega será concluída quando uma confirmação da publicação tiver sido recebida do gerenciador de filas. `MqttCallback.deliveryComplete` será chamado quando a confirmação for recebida. A mensagem poderá ser entregue mais de uma vez antes de `MqttCallback.deliveryComplete` ser chamado, se as comunicações forem lentas ou não confiáveis.

Exatamente uma vez

QoS=2
A entrega será concluída quando o cliente receber uma mensagem de conclusão de que a publicação foi publicada para os assinantes. `MqttCallback.deliveryComplete` será chamado assim que a mensagem de publicação for recebida. Ele não espera a mensagem de conclusão.

Em raras circunstâncias, o aplicativo cliente pode não retornar normalmente para o cliente MQTT a partir do `MqttCallback.deliveryComplete`. Você sabe que a entrega foi concluída porque o `MqttCallback.deliveryComplete` foi chamado. Se o cliente reiniciar a mesma sessão, `MqttCallback.deliveryComplete` não será chamado novamente.

Entregas incompletas

Se a entrega não for concluída após a sessão do cliente ser desconectada, será possível conectar o cliente novamente e concluir a entrega. Só será possível concluir a entrega de uma mensagem, se a mensagem foi publicada em uma sessão com o atributo `MqttConnectionOptions` configurado como `false`.

Crie o cliente usando o mesmo identificador de cliente e endereço do servidor e, em seguida, se conecte configurando o atributo `cleanSession` `MqttConnectionOptions` como `false` novamente. Se você configurar `cleanSession` como `true`, os tokens de entrega pendentes serão descartados.

Será possível verificar se há alguma entrega pendente chamando `MqttClient.getPendingDeliveryTokens`. Será possível chamar `MqttClient.getPendingDeliveryTokens` antes de se conectar ao cliente.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Publicação last will and testament

Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Publicação last will and testament

Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Crie um tópico para a publicação last will and testament. Você pode criar um tópico como `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Configure uma "last will and testament" usando o método `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Considere criar um registro de data e hora na mensagem `lastWillPayload`. Inclua outras informações do cliente que auxiliem na identificação do cliente e nas circunstâncias da conexão. Transmita o objeto `MqttConnectionOptions` para o construtor `MqttClient`.

Configure `lastWillQos` como 1 ou 2 para transformar a mensagem em persistente no IBM MQ e garantir a entrega. Para reter as informações da última conexão perdida, configure `lastWillRetained` como `true`.

A publicação "last will and testament" será enviada para os assinantes, se a conexão terminar inesperadamente. Ela será enviada se a conexão terminar sem que o cliente chame o método `MqttClient.disconnect`.

Para monitorar conexões, complemente a publicação "last will and testament" com outras publicações para registrar conexões e desconexões programadas.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Persistência de mensagem em clientes MQTT

As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o

cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

No MQTT, a persistência de mensagens tem dois aspectos, como a mensagem é transferida e se ela é enfileirada no IBM MQ como uma mensagem persistente.

1. O cliente MQTT acopla a persistência da mensagem com a qualidade de serviço. Dependendo da qualidade de serviço escolhida para uma mensagem, a mensagem se torna persistente. A persistência de mensagem é necessária para implementar a qualidade de serviço necessária.

   Se você especificar "no máximo uma vez", QoS=0, o cliente descartará a mensagem assim que ela for publicada. Se houver alguma falha no processamento de envio de dados da mensagem, ela não será enviada novamente. Mesmo se o cliente permanecer ativo, a mensagem não será enviada novamente. O comportamento das mensagens QoS=0 é o mesmo que as mensagens não persistentes rápidas do IBM MQ.

   Se uma mensagem for publicada por um cliente com QoS de 1 ou 2, ela se tornará persistente. A mensagem será armazenada localmente e só será descartada a partir do cliente quando não for mais necessária para garantir a entrega "pelo menos uma vez", QoS=1 ou "exatamente uma vez" QoS=2.
2. Se uma mensagem for marcada como QoS 1 ou 2, ela será enfileirada no IBM MQ como uma mensagem persistente. Se ela for marcada como QoS=0, ela será enfileirada no IBM MQ como uma mensagem não persistente. Nas mensagens não persistentes do IBM MQ são transferidas entre gerenciadores de filas "exatamente uma vez", a menos que o canal de mensagens tenha o atributo NPMSPEED configurado como FAST.

Uma publicação persistente é armazenada no cliente até ser recebida por um aplicativo cliente. Para QoS=2, a publicação será descartada a partir do cliente quando o retorno de chamada do aplicativo retornar ao controle. Para QoS=1, o aplicativo poderá receber a publicação novamente, se ocorrer uma falha. Para QoS=0, o retorno de chamada não recebe a publicação mais de uma vez. Ele poderá não receber a publicação se houver uma falha ou se o cliente estiver desconectado no momento da publicação.

Quando você se subscrever para um tópico, será possível reduzir a QoS com a qual o assinante recebe as mensagens para corresponder a suas capacidades de persistência. As publicações criadas em uma QoS `maior` superior são enviadas com a QoS mais alta que o assinante solicitou.

Armazenando de mensagens

A implementação do armazenamento de dados em pequenos dispositivos varia bastante. O modelo de mensagens persistentes de salvamento temporariamente em armazenamento gerenciado pelo cliente MQTT, pode ser muito lento ou demandar muito armazenamento. Em dispositivos móveis, o sistema operacional móvel poderá fornecer um serviço de armazenamento ideal para mensagens do MQTT.

Para fornecer flexibilidade para atender às restrições de pequenos dispositivos, o cliente MQTT tem duas interfaces de persistência. As interfaces definem as operações envolvidas no armazenamento de mensagens persistentes. As interfaces são descritas na documentação da API para o MQTT client for Java. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT.. É possível implementar as interfaces para se adequar a um dispositivo. O cliente MQTT executado no Java SE tem uma implementação padrão das interfaces que armazenam mensagens persistentes no sistema de arquivos. Usa o pacote `java.io`.

Classes de persistência

`MqttClientPersistence`
: Transmite uma instância de sua implementação de `MqttClientPersistence` para o cliente MQTT como um parâmetro do construtor `MqttClient`. Se você omitir o parâmetro `MqttClientPersistence` do construtor `MqttClient`, o cliente MQTT armazenará as mensagens persistentes usando a classe `MqttDefaultFilePersistence`.

MqttPersistable

`MqttClientPersistence` obtém e coloca objetos `MqttPersistable` usando uma chave de armazenamento. Deve-se fornecer uma implementação de `MqttPersistable`, bem como a implementação de `MqttClientPersistence` se você não estiver usando o `MqttDefaultFilePersistence`.

MqttDefaultFilePersistence

O cliente MQTT fornece a classe `MqttDefaultFilePersistence`. Se você instanciar `MqttDefaultFilePersistence` em seu aplicativo cliente, será possível fornecer o diretório para armazenar mensagens persistentes como um parâmetro do construtor `MqttDefaultFilePersistence`.

Em alternativa, o cliente MQTT pode instanciar `MqttDefaultFilePersistence` e colocar arquivos no diretório padrão a seguir:

client identifier -tcp *hostname portnumber*

Os caracteres a seguir são removidos da sequência de caracteres do nome do diretório:

"\", "\\", "/", ":" e " "

O caminho para o diretório é o valor da propriedade do sistema `rcp.data`; Se `rcp.data` não for configurado, o caminho é o valor da propriedade do sistema `usr.data`, em que

- `rcp.data` é uma propriedade associada à instalação de um OSGi ou Eclipse Rich Client Platform (RCP).
- `usr.data` é o diretório no qual o comando Java que iniciou o aplicativo foi ativado.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT

O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas

O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente

O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament

Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Publicações

Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT

Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente

MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Publicações

Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Um `MqttMessage` tem uma matriz de bytes como sua carga útil. Tente manter as mensagens o menor possível. O comprimento máximo de mensagem permitido pelo MQTT protocol é de 250 MB.

Geralmente, um programa cliente do MQTT usa `java.lang.String` ou `java.lang.StringBuffer` para manipular o conteúdo da mensagem. Por conveniência, a classe `MqttMessage` tem um método `toString` para converter sua carga útil para uma sequência. Para criar a carga útil da matriz de bytes a partir de um `java.lang.String` ou `java.lang.StringBuffer`, use o método `getBytes`.

O método `getBytes` converte uma sequência para o conjunto de caracteres padrão para a plataforma. O conjunto de caracteres padrão geralmente é `UTF-8`. As publicações do MQTT que contêm apenas texto são normalmente codificadas em `UTF-8`. Use o método `getBytes("UTF8")` para substituir o conjunto de caracteres padrão.

No IBM MQ, uma publicação do MQTT é recebida como uma mensagem `jms-bytes`. A mensagem inclui uma pasta `MQRFH2` que contém um `<mqtt>` e uma pasta `<mqps>`. A pasta `<mqtt>` contém o `clientId`, `msgId` e `qos`, mas esse conteúdo pode mudar no futuro.

Um `MqttMessage` tem três atributos adicionais: qualidade de serviço, se está retida e se é uma duplicata. O sinalizador duplicado será configurado somente se a qualidade de serviço for "pelo menos uma vez" ou "exatamente uma vez". Se a mensagem foi enviada anteriormente e não for confirmada rápido suficiente pelo cliente MQTT, a mensagem será enviada novamente, com o atributo duplicado configurado como `true`.

Publicação

Para criar uma publicação em um aplicativo cliente MQTT, crie um `MqttMessage`. Configure sua carga útil, qualidade de serviço e se ela está retida, e chame o método `MqttTopic.publish(MqttMessage message)`; `MqttDeliveryToken` é retornado e a conclusão da publicação é assíncrona.

Como alternativa, o cliente MQTT poderá criar um objeto de mensagem temporário para você a partir dos parâmetros no método `MqttTopic.publish(byte [] payload, int qos, boolean retained)`, ao criar uma publicação.

Se a publicação tiver uma qualidade de serviço "pelo menos uma vez" ou "exatamente uma vez", `QoS=1` ou `QoS=2`, o cliente MQTT chamará a interface `MqttClientPersistence`. Ele chamará `MqttClientPersistence` para armazenar a mensagem antes de retornar um token de entrega para o aplicativo.

O aplicativo pode escolher bloquear até a mensagem ser entregue ao servidor usando o método `MqttDeliveryToken.waitForCompletion`. Como alternativa, o aplicativo pode continuar sem

bloqueio. Se você deseja verificar se as publicações foram entregues, sem bloqueio, registre uma instância de uma classe de retorno de chamada que implementa `MqttCallback` com o cliente MQTT. O cliente MQTT chama o método `MqttCallback.deliveryComplete` assim que a publicação foi entregue. Dependendo da qualidade de serviço, a entrega pode ser quase imediatamente para `QoS=0` ou ela pode levar algum tempo para `QoS=2`.

Use o método `MqttDeliveryToken.isComplete` para pesquisar se a entrega está concluída. Enquanto o valor de `MqttDeliveryToken.isComplete` for `false`, será possível chamar `MqttDeliveryToken.getMessage` para obter o conteúdo da mensagem. Se o resultado da chamada `MqttDeliveryToken.isComplete` for `true`, a mensagem foi descartada e a chamada de `MqttDeliveryToken.getMessage` lançaria uma exceção nula de ponteiro. Não há nenhuma sincronização integrada entre o `MqttDeliveryToken.getMessage` e `MqttDeliveryToken.isComplete`.

Se o cliente se desconectar antes de receber todos os tokens de entrega pendentes, uma nova instância do cliente poderá consultar os tokens de entrega pendentes antes de se conectar. Até que o cliente se conecte, nenhuma nova entrega será concluída e segura para chamar `MqttDeliveryToken.getMessage`. Use o método `MqttDeliveryToken.getMessage` para descobrir quais publicações não foram entregues. Os tokens de entrega pendentes serão descartados se você se conectar com `MqttConnectOptions.cleanSession` configurado em seu valor padrão, `true`.

Assinando

Um gerenciador de filas é responsável por criar publicações para enviar a um assinante do MQTT. O gerenciador de filas verificará se o filtro de tópicos em uma assinatura criada por um cliente MQTT corresponderá à sequência de tópicos em uma publicação. A correspondência pode ser uma correspondência exata ou a correspondência pode incluir caracteres curingas. Antes de a publicação ser encaminhada para o assinante pelo gerenciador de filas, o gerenciador de filas verificará os atributos do tópico associada à publicação. Ele segue o procedimento de procura descrito em Assinatura usando uma sequência de tópicos que contém caracteres curingas para identificar se um objeto de tópico administrativo concederá a autoridade do usuário para assinatura.

Quando o cliente MQTT receber uma publicação com a qualidade de serviço "pelo menos uma vez", ele chamará o método `MqttCallback.messageArrived` para processar a publicação. Se a qualidade de serviço da publicação for "exatamente uma vez", `QoS=2`, o cliente MQTT chamará a interface `MqttClientPersistence` para armazenar a mensagem quando ela for recebida. Ela então chama `MqttCallback.messageArrived`.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Qualidades de serviço fornecidas por um cliente MQTT

Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

A qualidade de serviço de uma publicação é um atributo de `MqttMessage`. Ela é configurada pelo método `MqttMessage.setQos`.

O método `MqttClient.subscribe` pode reduzir a qualidade de serviço aplicada às publicações enviadas para um cliente em um tópico. A qualidade de serviço de uma publicação encaminhada para um assinante pode ser diferente da qualidade de serviço da publicação. O menor dos dois valores é usado para encaminhar uma publicação.

No máximo uma vez
`QoS=0`

A mensagem é entregue no máximo uma vez ou não é entregue de modo algum. Sua entrega na rede não é reconhecida.

A mensagem não é armazenada. A mensagem poderá ser perdida se o cliente for desconectado ou se o servidor falhar.

`QoS=0` é o modo de transferência mais rápido. Ele é, às vezes, chamado de "fire and forget".

O MQTT protocol não requer que servidores encaminhem publicações no `QoS=0` para um cliente. Se o cliente estiver desconectado no momento em que o servidor receber a publicação, a publicação poderá ser descartada, dependendo do servidor. O serviço de telemetria (MQXR) não descarta mensagens enviadas com `QoS=0`. Elas são armazenadas como mensagens não persistentes e só serão descartadas, se o gerenciador de filas for interrompido.

Pelo menos uma vez

QoS=1

QoS=1 é o modo de transferência padrão.

A mensagem é sempre entregue pelo menos uma vez. Se o emissor não receber uma confirmação, a mensagem será enviada novamente com o sinalizador DUP configurado até que uma confirmação seja recebida. Como resultado, a mesma mensagem pode ser enviada ao destinatário múltiplas vezes e ele pode processá-la múltiplas vezes.

A mensagem deve ser armazenada localmente no emissor e no receptor até ser processada.

A mensagem será excluída do receptor após ter processado a mensagem. Se o receptor for um broker, a mensagem será publicada para seus assinantes. Se o receptor for um cliente, a mensagem será entregue para o aplicativo de assinante. Após a mensagem ser excluída, o receptor enviará uma confirmação para o emissor.

A mensagem será excluída do emissor após ter recebido uma confirmação do receptor.

Exatamente uma vez

QoS=2

A mensagem é sempre entregue exatamente uma vez.

A mensagem deve ser armazenada localmente no emissor e no receptor até ser processada.

QoS=2 é o mais seguro e o modo de transferência mais lento. Ele levará pelo menos dois pares de transmissões entre o emissor e o receptor antes de a mensagem ser excluída do emissor. A mensagem poderá ser processada no receptor após a primeira transmissão.

No primeiro par de transmissões, o emissor transmite a mensagem e recebe a confirmação do receptor de que ele armazenou a mensagem. Se o emissor não receber uma confirmação, a mensagem será enviada novamente com o sinalizador DUP configurado até que uma confirmação seja recebida.

No segundo par de transmissões, o emissor informa ao receptor que ele pode concluir o processamento da mensagem, "PUBREL". Se o emissor não receber uma confirmação da mensagem "PUBREL", a mensagem "PUBREL" será enviada novamente até que uma confirmação seja recebida. O emissor excluirá a mensagem que salvou ao receber a confirmação para a mensagem "PUBREL"

O receptor poderá processar a mensagem na primeira ou na segunda fase, contanto que não reprocesse a mensagem. Se o receptor for um broker, ele publicará a mensagem para os assinantes. Se o receptor for um cliente, ele entregará a mensagem para o aplicativo de assinante. O receptor envia uma mensagem de conclusão de volta para o emissor que concluiu o processamento da mensagem.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT

O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas

O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente

O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Publicações retidas e clientes MQTT

Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Use o método `MqttMessage.setRetained` para especificar se uma publicação em um tópico está retida.

Ao criar ou atualizar uma publicação retida, envie a publicação com um QoS de 1 ou 2. Se você enviar com um QoS de 0, o IBM MQ cria uma publicação retida não persistente. A publicação não será retida se o gerenciador de filas for interrompido.

Se você publicar uma publicação não retida para um tópico que tenha uma publicação retida, a publicação retida não será afetada. Os assinantes atuais recebem a nova publicação. Novos assinantes recebem a publicação retida por primeiro, em seguida, recebem quaisquer novas publicações.

É possível usar uma publicação retida para registrar o valor mais recente de uma medida. Novos assinantes de um tópico recebem imediatamente o valor mais recente da medida. Se nenhuma nova medida for adquirida desde que o assinante se inscreveu por último no tópico de publicação e se o assinante se inscrever novamente, o assinante receberá a publicação retida mais recente no tópico novamente.

Para excluir uma publicação retida, você tem duas opções:

- Execute o comando MQSC **CLEAR TOPICSTR** .
- Criar uma publicação retida de comprimento zero. Conforme estabelecido na especificação MQTT 3.1.1, se uma mensagem retida de comprimento zero for publicada em um tópico, qualquer mensagem retida desse tópico será limpa.

Conceitos relacionados
Retornos de chamadas e sincronização em aplicativos do cliente MQTT

O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Assinaturas

Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Crie assinaturas usando os métodos `MqttClient.subscribe`, passando um ou mais filtros de tópicos e parâmetros de qualidade de serviço. O parâmetro de qualidade de serviço configura a qualidade de serviço máxima que o assinante está preparado para usar para receber uma mensagem. Mensagens enviadas para esse cliente não podem ser entregues com qualidade de serviço superior. A qualidade de serviço é configurada para o valor original mais baixo quando a mensagem foi publicada e para o nível especificado para a assinatura. A qualidade de serviço padrão para o recebimento de mensagens é `QoS=1`, pelo menos uma vez.

A solicitação de assinatura em si é enviada com `QoS=1`.

Publicações são recebidas por um assinante quando o cliente MQTT chama o método `MqttCallback.messageArrived`. O método `messageArrived` também passa a sequência de tópicos com a qual a mensagem foi publicada para o assinante.

É possível remover uma assinatura ou um conjunto ou assinaturas usando os métodos `MqttClient.unsubscribe`.

Um comando IBM MQ pode remover uma assinatura. Listar assinaturas usandoIBM MQ Explorer , ou usando**runmqsc** ou comandos PCF. Todas as assinaturas do cliente MQTT são nomeadas. Eles recebem um nome no formato:*`ClientIdentifier`*:*`Topic name`*

Se você usar o `MqttConnectOptions`padrão ou configurar `MqttConnectOptions.cleanSession` como `true` antes de conectar o cliente, quaisquer assinaturas antigas para o cliente serão removidas quando o cliente se conectar.. Quaisquer novas assinaturas feitas pelo cliente durante a sessão serão removidas quando ele se desconectar.

Se você configurar `MqttConnectOptions.cleanSession` como `false` antes de se conectar, quaisquer assinaturas criadas pelo cliente serão incluídas em todas as assinaturas que existiam para o cliente antes de ele se conectar. Todas as assinaturas permanecem ativas quando o cliente se desconecta.

Outra maneira de entender a maneira como o atributo `cleanSession` afeta assinaturas é pensar nele como um atributo modal. Em seu modo padrão, `cleanSession=true`, o cliente cria assinaturas e recebe publicações apenas dentro do escopo da sessão. No modo alternativo, `cleanSession=false`, assinaturas são duráveis. O cliente pode se conectar e se desconectar e suas assinaturas permanecem ativas. Ao se reconectar, o cliente recebe todas as publicações não entregues. Enquanto estiver conectado, ele pode modificar o conjunto de assinaturas que estão ativas em seu nome.

Deve-se configurar o modo `cleanSession` antes de conectar; o modo dura toda a sessão. Para alterar essa configuração, você deve desconectar e reconectar o cliente. Se você mudar os modos de usar `cleanSession=false` para `cleanSession=true`, todas as assinaturas anteriores para o cliente e quaisquer publicações que não foram recebidas serão descartadas

Publicações que correspondem a assinaturas ativas são enviadas para o cliente assim que são publicadas. Se o cliente estiver desconectado, elas serão enviadas para o cliente se reconectar ao mesmo servidor com o mesmo identificador de cliente e se `MqttConnectOptions.cleanSession` estiver configurado para `false`.

Assinaturas para um determinado cliente são identificadas pelo identificador de cliente. É possível reconectar o cliente de um dispositivo de cliente diferente ao mesmo servidor, continuar com as mesmas assinaturas e receber publicações não entregues.

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui

assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações
Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Sequências e filtros de tópicos em clientes MQTT
As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

Sequências e filtros de tópicos em clientes MQTT

As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM MQ.

As sequências de tópicos são usadas para enviar publicações para os assinantes. Crie uma sequência de tópicos usando o método `MqttClient.getTopic(java.lang.String topicString)`.

Os filtros de tópicos são usados para assinar tópicos e receber publicações. Os filtros de tópicos podem conter caracteres curingas. Com caracteres curinga, é possível assinar vários tópicos. Crie um filtro de tópicos usando um método de assinatura; por exemplo, `MqttClient.subscribe(java.lang.String topicFilter)`.

Sequências de tópicos

A sintaxe de uma sequência de tópico IBM MQ é descrita em Sequências de tópicos. A sintaxe de sequência de tópicos MQTT é descrita na classe `MqttClient` na documentação da API para o MQTT client for Java. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

A sintaxe de cada tipo de sequência de tópicos é quase idêntica. Há quatro pequenas diferenças:

1. As sequências de tópicos enviadas para IBM MQ por clientes MQTT devem seguir a convenção para nomes do gerenciador de filas.
2. Os comprimentos máximos são diferentes. As sequências de tópicos do IBM MQ são limitadas a 10.240 caracteres. Um cliente MQTT pode criar sequências de tópicos de até 65.535 bytes.
3. Uma sequência de tópicos criada por um cliente MQTT não pode conter um caractere nulo.
4. Em IBM Integration Bus, um nível de tópico nulo, `'...//...'` é inválido. Os níveis de tópicos nulos são suportados pelo IBM MQ.

Ao contrário do publicar/assinar IBM MQ, o protocolo `mqttv3` não tem um conceito de objeto de tópico administrativo. Não é possível construir uma sequência de tópicos a partir de um objeto do tópico e uma sequência de tópicos. No entanto, uma sequência de tópicos é mapeada para um tópico administrativo no IBM MQ. O controle de acesso associado ao tópico administrativo determina se uma publicação é publicada para o tópico ou descartada. Os atributos aplicados a uma publicação quando for redirecionada para os assinantes serão influenciados pelos atributos do tópico administrativo.

Filtros de tópico

A sintaxe de um filtro tópico IBM MQ é descrita em Esquema curinga baseado em tópicos. A sintaxe dos filtros de tópicos que você pode construir com um cliente MQTT são descritas na classe `MqttClient` na documentação da API para o MQTT client for Java. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

Conceitos relacionados

Retornos de chamadas e sincronização em aplicativos do cliente MQTT
O modelo de programação do cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT, o máximo que puderem, a partir dos atrasos na transmissão de mensagens de e para servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Sessões limpas
O cliente MQTT e o serviço de telemetria (MQXR) mantêm informações do estado da sessão. As informações de estado são usadas para assegurar entrega "pelo menos uma vez" e "exatamente uma vez" e recebimento "exatamente uma vez" de publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Identificador de Cliente
O identificador do cliente é uma sequência de caracteres de 23 bytes que identifica um cliente MQTT. Cada identificador deve ser exclusivo para apenas um cliente conectado de cada vez. O identificador deve conter apenas caracteres válidos em um nome de gerenciador de filas. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

Tokens de entrega

Publicação last will and testament
Se uma conexão do cliente MQTT terminar inesperadamente, é possível configurar o MQ Telemetry para enviar uma publicação "last will and testament". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Persistência de mensagem em clientes MQTT
As mensagens de publicação serão transformadas em persistentes se enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações enviadas para o cliente ou por meio dele.

Publicações

Publicações são instâncias de `MqttMessage` associadas a uma cadeia de tópicos Os clientes do MQTT podem criar publicações para enviar ao IBM MQ e assinar tópicos no IBM MQ para receber publicações.

Qualidades de serviço fornecidas por um cliente MQTT
Um cliente MQTT fornece três qualidades de serviço para entrega de publicações para o IBM MQ e para o cliente MQTT: "no máximo uma vez", "no mínimo uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

Publicações retidas e clientes MQTT
Um tópico pode ter uma, e apenas uma, publicação retida. Se você criar uma assinatura de um tópico que tenha uma publicação retida, a publicação será imediatamente encaminhada para você.

Assinaturas
Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Desenvolvendo aplicativos Microsoft Windows Communication Foundation com o IBM MQ

O canal customizado do Microsoft Windows Communication Foundation (WCF) para IBM MQ envia e recebe mensagens entre clientes e serviços WCF.

Conceitos relacionados

“Introdução ao canal customizado do IBM MQ para o WCF com o .NET” na página 1279
O canal customizado para o IBM MQ é um canal de transporte usando o modelo de programação unificado do Microsoft Windows Communication Foundation (WCF).

“Usando os canais customizados do IBM MQ para WCF” na página 1284
Visão Geral das informações disponíveis para programadores que usam canais customizados do IBM MQ para Windows Communication Foundation (WCF).

“Usando as Amostras do WCF” na página 1303
As amostras do Windows Communication Foundation (WCF) fornecem alguns exemplos simples de como o canal customizado do IBM MQ pode ser usado.

FFST: Tecnologia de Suporte de Primeira Falha do WCF XMS

Tarefas relacionadas

Rastreando o canal customizado do WCF para o IBM MQ

Resolução de problemas do canal customizado WCF para problemas do IBM MQ

Introdução ao canal customizado do IBM MQ para o WCF com o .NET

O canal customizado para o IBM MQ é um canal de transporte usando o modelo de programação unificado do Microsoft Windows Communication Foundation (WCF).

A estrutura do Microsoft Windows Communication Foundation, introduzida no Microsoft.NET 3, permite que aplicativos e serviços .NET sejam desenvolvidos independentemente do transporte e dos protocolos usados para conectá-los, possibilitando que transportes ou configurações alternativos sejam usados de acordo com o ambiente em que o serviço ou o aplicativo está implementado.

As conexões são gerenciadas no tempo de execução pelo WCF construindo uma pilha de canais que contém a combinação necessária de:

- Elementos de protocolo: Um conjunto opcional de elementos em que nenhum, um ou mais podem ser incluídos para suportar protocolos como os padrões WS-*.
- Codificador de mensagem: Um elemento obrigatório na pilha que controla a serialização da mensagem em seu formato de ligação.

- Canal de transporte: Um elemento obrigatório na pilha responsável por transportar a mensagem serializada para seu terminal.

O canal customizado para o IBM MQ é um canal de transporte e, como tal, deve ser unido a um codificador de mensagem e protocolos opcionais conforme requerido pelo aplicativo usando uma ligação customizada do WCF. Dessa maneira, aplicativos que foram desenvolvidos para usar o WCF podem usar o canal customizado para o IBM MQ para enviar e receber dados da mesma maneira que usam os transportes integrados fornecidos por Microsoft, possibilitando a integração simples com funções do sistema de mensagens assíncronas, escaláveis e confiáveis do IBM MQ. Para obter uma lista completa das funções suportadas, consulte: "Recursos de Capacidades do Canal Customizado WCF" na página 1284.

Quando e por que uso o canal customizado do IBM MQ para WCF?

É possível usar o canal customizado do IBM MQ para enviar e receber mensagens entre clientes e serviços do WCF da mesma maneira que os transportes integrados fornecidos pela Microsoft, possibilitando aos aplicativos acessar os recursos do IBM MQ dentro do modelo de programação unificado do WCF.

Cenários típicos de padrão de uso para o canal personalizado IBM MQ para WCF é como uma interface não SOAP para transmissão de mensagens nativas IBM MQ.

As mensagens transportadas usando o formato de mensagens não SOAP/não JMS (Pure MQMessage)

Ao usar o canal customizado do IBM MQ para WCF como uma interface não SOAP para a transmissão de mensagens nativas do IBM MQ, as mensagens serão transportadas usando o formato de mensagens não SOAP/não JMS (Pure MQMessage) de IBM MQ.

Os usuários do WCF são capazes de iniciar o serviço ou, em outras palavras, os usuários de serviços são aptos a enviar uma mensagem para uma fila do IBM MQ usando MQMessages. Os aplicativos podem obter e configurar os campos MQMD e carga útil. Quando a mensagem estiver disponível em filas do IBM MQ , essa mensagem poderá ser processada por qualquer serviço WCF ou aplicativos não WCF, como aplicativos C ou Java que estejam em execução no AIX, Linux, Windowsou z/OS

Requisitos de software para o canal customizado do IBM MQ para WCF

Este tópico descreve os requisitos de software para o canal customizado do IBM MQ para WCF. O canal customizado do IBM MQ para WCF pode conectar-se apenas aos gerenciadores de filas do IBM WebSphere MQ 7.0 ou superior.

Requisitos do Ambiente de Tempo

- Microsoft.NET Framework v4.7.2 ou superior deve ser instalado na máquina host.
- O *Java and .NET Messaging and Web Services* é instalado por padrão como parte do instalador do IBM MQ. Esse componente instala os conjuntos do .NET necessários para o canal customizado no Cache de conjunto global.

Nota: Se o Microsoft .NET Framework V4.7.2 ou superior não for instalado antes da instalação do IBM MQ, a instalação do produto IBM MQ continuará sem erro, mas o IBM MQ classes for .NET não estará disponível Se o .NET Framework for instalado após instalar o IBM MQ, então os conjuntos IBM MQ.NET devem ser registrados executando o script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd`, em que *WMQInstallDir* é o diretório em que o IBM MQ está instalado. Este script instala as montagens necessárias no Global Assembly Cache (GAC). Um conjunto de arquivos do `amqi*.log` que registram as ações que são tomadas é criado no diretório %TEMP%. Não será necessário executar novamente o script `amqiRegisterdotNet.cmd` se .NET for atualizado para V4.7.2 ou superior a partir de uma versão anterior, por exemplo, de .NET V3.5.

Requisitos do Ambiente de Desenvolvimento

- Microsoft Visual Studio 2015 ou Windows Software Development Kit para .NET 4.7.2 ou mais recente.
- A Microsoft.NET Estrutura V4.7.2 ou superior deve ser instalada na máquina host para construir os arquivos da solução de amostra

Canal customizado doIBM MQ para WCF: o que está instalado?

O canal customizado para o IBM MQ é um canal de transporte usando o modelo de programação unificado do Microsoft Windows Communication Foundation (WCF). O canal customizado é instalado por padrão como parte da instalação.

Canal customizado do IBM MQ para o WCF

O canal customizado e suas dependências estão contidos dentro do componente `Java and .NET Messaging and Web Services`, que é instalado por padrão. Para fazer upgrade do IBM MQ de uma versão anterior à IBM MQ 8.0, a atualização instalará o canal customizado do IBM MQ para WCF por padrão se o componente `Java and .NET Messaging and Web Services` tiver sido instalado previamente em uma instalação anterior.

O componente `.NET Messaging and Web Services` contém o arquivo `IBM.XMS.WCF.dll` e o arquivo `IBM.WMQ.WCF.dll`. Esses arquivos são o principal conjunto de canais customizados que contém as classes de interface do WCF. Esses arquivos estão instalados no Global Assembly Cache (GAC) e também estão disponíveis no diretório a seguir: *MQ_INSTALLATION_PATH* `\bin`, em que *MQ_INSTALLATION_PATH* é o diretório no qual o IBM MQ está instalado.

A tabela a seguir resume as classes de chave que são necessárias para usar o canal customizado.

Tabela 189. As classes de chave necessárias para usar o canal customizado

	Interface SOAP/JMS (Existente)	**Interface não SOAP/não JMS (a partir de IBM MQ 8.0)**
Montagem de Canal Customizado	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Nome de Ligação de Transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Transport Binding Importer:	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Configuração de Ligação de Transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Samples(Oneway)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client,MQMessaging_OneWay_Service
Samples(RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	MQMessaging_RequestReply_Client,MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll suporta interfaces SOAP/JMS e não SOAP/não JMS. Novos aplicativos desenvolvidos são recomendados para usar a montagem IBM.WMQ.WCF, pois ela suporta ambas as interfaces.

Enviando mensagens formatadas MQSTR

Se a mensagem de solicitação for do tipo MQSTR, será possível selecionar para enviar a mensagem de resposta no formato MQSTR.

Deve-se usar um parâmetro de URI adicional **`replyMessageFormat`** para mudar o formato da mensagem de resposta. Os valores suportados são:

""

" " é o valor padrão.

A mensagem de resposta está em formato de byte (MQMFT_NONE). Por exemplo:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR

A mensagem de resposta está em formato MQSTR (MQMFT_STRING). Por exemplo:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

Notas:

1. O valor para **replyMessageFormat** não faz distinção entre maiúsculas e minúsculas.
2. O uso de qualquer valor diferente de " " ou *MQSTR* causa uma exceção de valor de parâmetro inválido.

Amostras do canal customizado doIBM MQ

As amostras fornecem alguns exemplos simples de como o canal customizado do IBM MQ para WCF pode ser usado. As amostras e seus arquivos associados estão localizados no diretório *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para o IBM MQ. Para obter mais informações sobre as amostras de canal customizado do IBM MQ, consulte "Usando as Amostras do WCF" na página 1303.

svcutil.exe.config

O `svcutil.exe.config` é um exemplo das definições de configuração necessárias para ativar a ferramenta de geração de proxy do cliente do Microsoft WCF `svcutil` para reconhecer o canal customizado. O arquivo `svcutil.exe.config` está localizado no diretório *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para o IBM MQ. Para obter mais informações sobre o uso do `svcutil.exe.config`, consulte "Gerando um Proxy de Cliente do WCF e Arquivos de Configuração do Aplicativo Usando a Ferramenta svcutil com Metadados de um Serviço em Execução" na página 1301.

Arquitetura do WCF

O canal customizado do IBM MQ para o WCF é integrado a parte superior da API `IBM Message Service Client for .NET (XMS .NET)`.

Interface SOAP/JMS

A arquitetura do WCF é conforme mostrada no diagrama a seguir:

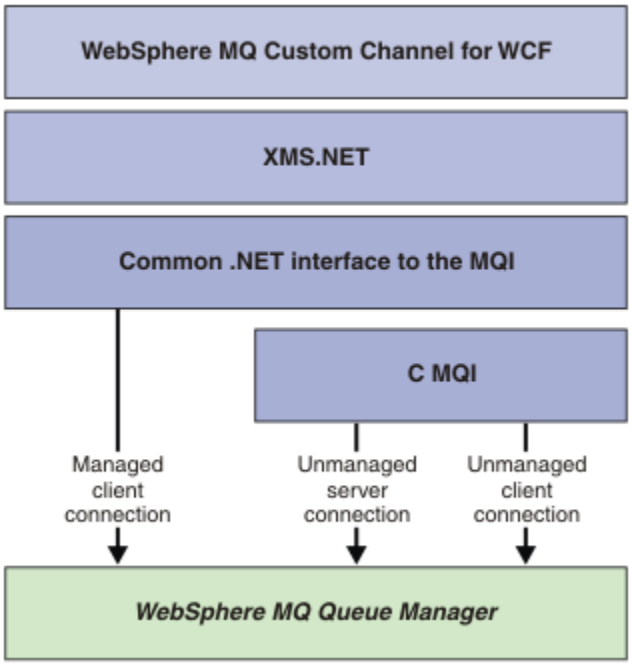


Figura 149. Arquitetura do WCF para a interface SOAP/JMS

Todos os componentes necessários são instalados por padrão com a instalação do produto.

A três conexões são:

- Conexões do Cliente Gerenciadas
- Conexões do servidor não gerenciadas
- Conexões do cliente não gerenciadas

Para obter mais informações sobre essas conexões, consulte “Opções de Conexão do WCF” na página 1290.

Interface não SOAP/não JMS

O canal customizado do IBM MQ para WCF suporta ambas as interfaces SOAP/JMS (disponível no IBM WebSphere MQ 7.0.1) e não SOAP/não JMS.

A arquitetura do WCF é conforme mostrada no diagrama a seguir:

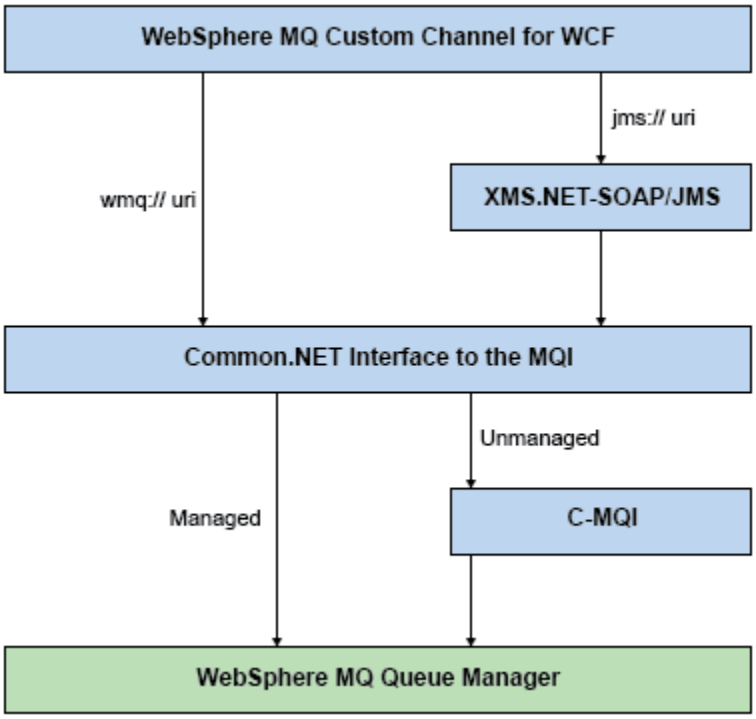


Figura 150. Arquitetura do WCF para a interface não SOAP/não JMS

Usando os canais customizados do IBM MQ para WCF

Visão Geral das informações disponíveis para programadores que usam canais customizados do IBM MQ para Windows Communication Foundation (WCF).

O Microsoft Windows Communication Foundation sustenta os serviços da web e o suporte ao sistema de mensagens no Microsoft.NET Framework 3 O IBM MQ pode ser usado como um canal customizado no WCF no .NET Framework 3 da mesma forma que os canais integrados oferecidos pelo Microsoft.

As mensagens transportadas pelo canal customizado são formatadas de acordo com a implementação SOAP sobre JMS de IBM MQ. Os aplicativos podem então se comunicar com serviços hospedados pelo WCF ou pela infraestrutura de serviço WebSphere SOAP over JMS.

Recursos de Capacidades do Canal Customizado WCF

Use os seguintes tópicos para obter informações relativas aos recursos e capacidades do canal customizado do WCF.

Formas de Canal Customizado WCF

Visão geral das formas de canal customizadas com as quais o IBM MQ pode ser usado dentro do canais customizados do Microsoft Windows Communication Foundation (WCF).

O canal customizado do IBM MQ para WCF suporta duas formas de canais:

- Unidirecional
- Pedido-resposta

O WCF automaticamente seleciona a forma de canal de acordo com o contrato de serviço que está sendo hospedado.

Os contratos que incluem métodos que usam apenas o parâmetro **IsOneWay** são servidos pela forma de canal unidirecional, por exemplo:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

Os contratos que incluem uma mistura dos métodos unidirecional e solicitação e resposta ou todos os métodos de solicitação e resposta são atendidos pela forma de canal solicitação e resposta. Por exemplo:

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

Nota: Ao misturar os métodos unidirecional e pedido-resposta no mesmo contrato, deve-se assegurar que o comportamento seja o pretendido, especialmente ao trabalho em um ambiente misto porque métodos unidirecionais esperarão até que recebam uma resposta nula do serviço.

Canal unidirecional

O canal customizado unidirecional do IBM MQ para o WCF é usado, por exemplo, para enviar mensagens de um cliente WCF usando um formato de canal unidirecional. O canal pode enviar mensagens apenas em uma direção, por exemplo, de um gerenciador de filas do cliente para uma fila em um serviço do WCF.

Canal Pedido-Resposta

O canal customizado de solicitação e resposta do IBM MQ para WCF é usado, por exemplo, para enviar mensagens em duas direções assincronamente. A mesma instância do cliente deve ser usada para o sistema de mensagens assíncrono. O canal pode enviar mensagens em uma direção, por exemplo, de um gerenciador de filas do cliente para uma fila em um serviço do WCF e, então, enviar uma mensagem de resposta do WCF para uma fila no gerenciador de filas do cliente.

Nomes e Valores de Parâmetros da URI do WCF

Nomes e valores de parâmetros do URI para a interface SOAP/JMS e interface Não SOAP/Não JMS.

Interface SOAP/JMS

connectionFactory
O parâmetro connectionFactory é necessário.

initialContextFactory
O parâmetro initialContextFactory é necessário e deve ser configurado como "com.ibm.mq.jms.Nojndi" para compatibilidade com o WebSphere Application Server e com outros produtos.

Interface não SOAP/Não JMS

O formato de URI é o mesmo das especificações de MA93. Consulte SupportPac - MA93 para obter detalhes adicionais das especificações IRI do IBM MQ.

Sintaxe de URI do IBM MQ

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
```

```
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

Exemplo de IRI IBM MQ

O IRI de exemplo a seguir informa um solicitante de serviço que ele pode usar uma conexão de ligação de cliente TCP de IBM MQ para uma máquina chamada example.com na porta 1414 e colocar mensagens de solicitação persistentes em uma fila chamada SampleQ no gerenciador de filas QM1. O IRI especifica que o provedor de serviços colocará respostas em uma fila chamada SampleReplyQ.

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

Para conexões ativadas para TLS

Para fazer conexões Asseguradas (TLS) usando o Cliente/Serviço WCF, configure as propriedades a seguir com valores apropriados no URI. Todas as propriedades que são prefixadas com "*" são obrigatórias para estabelecer uma conexão segura.

- **sslKeyRepository**: *SYSTEM ou *USER
- * **sslCipherSpec**: um CipherSpec válido, por exemplo TLS_RSA_WITH_AES_128_CBC_SHA256.
- **sslCertRevocationCheck**: true ou false.
- **sslKeyResetCount**: um valor maior que 32kb.
- **sslPeerName:** o nome distinto do certificado do servidor

Por exemplo:

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

Entrega Assegurada do Canal Customizado WCF

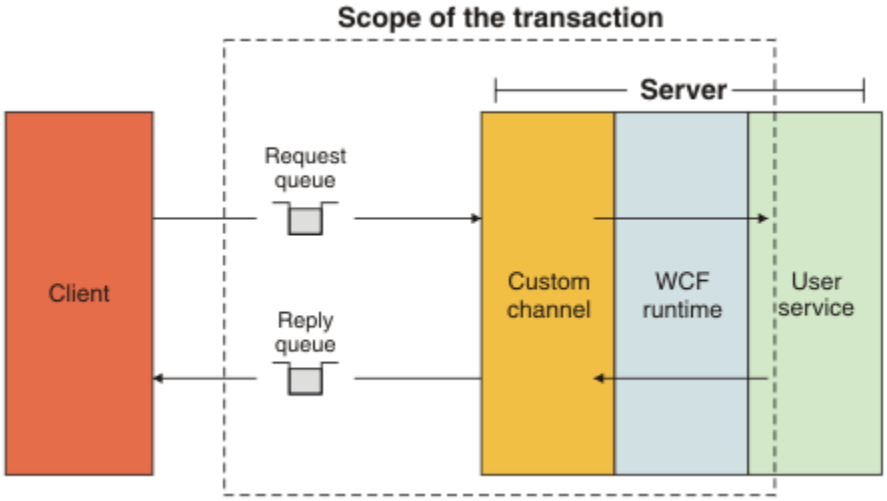
A Entrega Assegurada garante que um pedido ou resposta de serviço é acionada e não perdida.

Uma mensagem de pedido é recebida e qualquer mensagem de resposta é enviada em um ponto de sincronização da transação local, que pode ser retrocedido no caso de falha de tempo de execução. Exemplos dessas falhas são: Uma exceção não manipulada lançada pelo serviço, falha para executar o dispatch da mensagem para o serviço ou falha para entregar a mensagem de resposta.

`AssuredDelivery` é o atributo de entrega assegurada que pode ser especificado em um contrato de serviço para garantir que as mensagens de pedido recebidas por um serviço e a mensagem de resposta enviada de um serviço não sejam perdida no caso de uma falha de tempo de execução.

Para assegurar que as mensagens também sejam preservadas no caso de falha do sistema ou queda de energia, elas devem ser enviadas como persistentes. Para usar as mensagens persistentes, o aplicativo cliente deve ter esta opção especificada no URI do terminal.

As transações distribuídas não são suportadas e o escopo da transação não se estende além do processamento da mensagem de solicitação e resposta executado pelo IBM MQ. Qualquer trabalho executado dentro do serviço pode ser executado novamente como resultado de uma falha que faz com que a mensagem seja recebida outra vez. O diagrama a seguir mostra o escopo da transação:



Entrega assegurada é ativada aplicando o atributo `AssuredDelivery` à classe de serviço conforme mostrado no exemplo a seguir:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

Ao usar o atributo `AssuredDelivery`, você deve estar ciente dos seguintes pontos:

- Quando um canal determina que uma falha provavelmente ocorrerá outra vez se uma mensagem tiver sido revertida e recebida novamente, a mensagem será tratada como uma mensagem suspeita e não é retornada à fila de pedidos para reprocessamento. Por exemplo: Se a mensagem recebida não está formatada corretamente ou não pode ter o dispatch executado para um serviço. As exceções não manipuladas lançadas de uma operação de serviço são sempre reenviadas até que a mensagem tenha sido entregue novamente o número máximo de vezes especificado pela propriedade de limite de restauração da fila de solicitações. Para obter informações adicionais, consulte: “Mensagens Suspeitas do Canal Customizado WCF” na página 1288
- O canal executa a leitura, o processamento e a resposta de cada mensagem de pedido como uma operação atômica usando um único encadeamento de execução para reforçar a integridade transacional. Para possibilitar que as operações de serviço sejam executadas simultaneamente, o canal permite que o WCF crie várias instâncias do canal. O número de instâncias do canal disponíveis para pedidos de processamento é controlado pela propriedade de ligação `MaxConcurrentCalls`. Para obter informações adicionais, consulte: “Opções de Configuração de Ligação do WCF” na página 1296
- A função de entrega assegurada usa os pontos de extensibilidade do WCF IOperationInvoker e IErrorHandler. Se esses pontos de extensibilidade forem usados externamente por um aplicativo, o aplicativo deverá assegurar que qualquer ponto de extensibilidade registrado anteriormente seja chamado. A falha em fazer isso para IErrorHandler pode resultar em erros não serem relatados. A falha em fazer isso para IOperationInvoker pode fazer com que o WCF pare de responder.

Segurança do Canal Customizado WCF

O canal customizado do IBM MQ para o WCF suporta o uso do TLS somente para conexões do cliente não gerenciadas para o gerenciador de filas.

Especifique o TLS usando uma entrada na tabela de definição de canal de cliente (CCDT). Para obter mais informações sobre CCDTs, consulte Tabela de definição de canal do cliente.

Client Channel Definition Tables (CCDT) do WCF

O canal customizado do IBM MQ para WCF suporta o uso de tabelas de definição de canal do cliente (CCDT) para configurar as informações de conexão para conexões do cliente.

As CCDTs são controladas através dessas duas variáveis de ambiente:

- *MQCHLLIB* especifica o diretório no qual a tabela está localizada.
- *MQCHLTAB* especifica o nome do arquivo da tabela.

Se essas variáveis de ambiente estão definidas, elas terão prioridade sobre qualquer detalhe de conexão do cliente especificado no URI.

Para obter mais informações sobre tabelas de definição de canal do cliente, veja: Tabela de definição de canal do cliente.

Mensagens Suspeitas do Canal Customizado WCF

Quando um serviço falha em processar uma mensagem de pedido ou falha em entregar uma mensagem de resposta para uma fila de resposta, a mensagem é tratada como uma mensagem suspeita.

Mensagens de Pedido Suspeitas

Se uma mensagem de pedido não puder ser processada, ela será tratada como uma mensagem suspeita. Esta ação evita que o serviço receba a mesma mensagem não processável novamente. Para uma mensagem de pedido não processável ser tratada como uma mensagem suspeita, uma das seguintes situações deve ser verdadeira:

- A contagem de restaurações de mensagens excedeu o limite de restauração especificado na fila de pedidos, o que só ocorre se entrega garantida tiver sido especificada para o serviço. Para obter mais informações sobre a entrega garantida, veja: “Entrega Assegurada do Canal Customizado WCF” na página 1286
- A mensagem não foi formatada corretamente e não pôde ser interpretada como uma mensagem SOAP sobre JMS.

Mensagens de Resposta Suspeitas

Se um serviço falhar em entregar uma mensagem de resposta para a fila de resposta, a mensagem de resposta será tratada como uma mensagem suspeita. Para mensagens de resposta, esta ação possibilita que as mensagens de resposta sejam recuperadas posteriormente para auxiliar na determinação de problema.

Manipulação de Mensagens Suspeitas

A ação tomada para uma mensagem suspeita depende da configuração do gerenciador de filas e dos valores configurados nas opções de relatório da mensagem. Para SOAP sobre JMS, as seguintes opções de relatório são configuradas nas mensagens de solicitação por padrão e não são configuráveis:

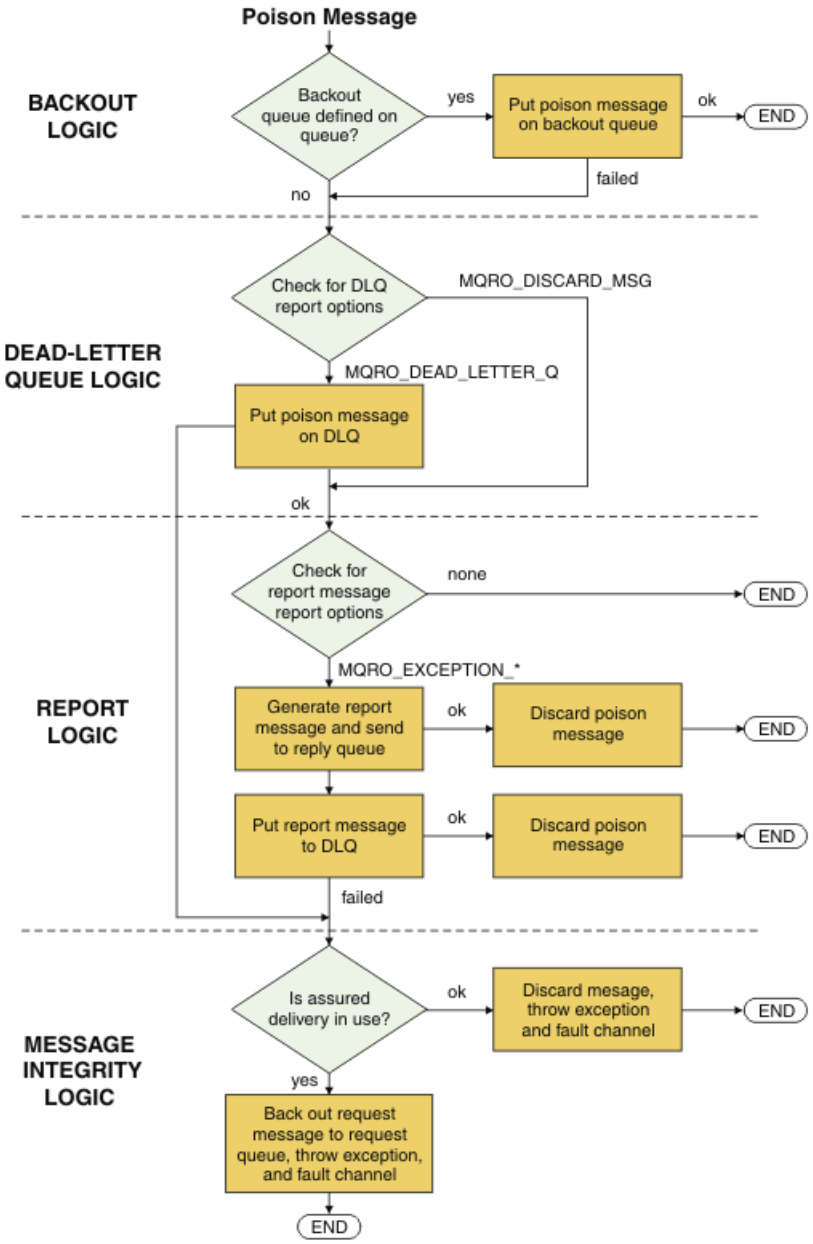
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Para SOAP sobre JMS, a seguinte opção de relatório é configurada em mensagens de resposta por padrão e não é configurável:

- MQRO_DEAD_LETTER_Q

Se as mensagens vierem de uma origem não WCF, consulte a documentação para essa origem.

O diagrama a seguir mostra as ações possíveis e as etapas executadas se a manipulação de mensagens suspeitas falhar:



Recursos de mensagens doIBM MQ para aplicativos WCF

Recursos de mensagens não SOAP/não JMS (ou seja, IBM MQ) para aplicativos WCF.

Para a interface não SOAP/não JMS, os recursos de mensagens do IBM MQ para aplicativos WCF são os seguintes:

- Os aplicativos WCF podem enviar e receber mensagens base do IBM MQ que podem ser processadas por qualquer aplicativo IBM MQ.
- Os aplicativos WCF têm controle total para atualizar o MQMD e a carga útil.
- O cliente WCF pode enviar mensagens do IBM MQ que podem ser consumidas por qualquer um dos clientes do IBM MQ, por exemplo C, clientes Java, JMS e .NET.

O WCF para interface não SOAP/não JMS deve usar as seguintes classes para configurar a carga útil da mensagem e MQMD para a mensagem:

- WmqStringMessage para uma carga útil do tipo String
- WmqBytesMessage para uma carga útil do tipo Bytes
- WmqXmlMessage para uma carga útil do tipo XML

Para configurar a carga útil da mensagem, use a propriedade **Data** para a classe WmqStringMessage, WmqBytesMessage ou WmqXmlMessage, dependendo do tipo de carga útil. Por exemplo, use o seguinte código para configurar uma carga útil do tipo String:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

Opções de Conexão do WCF

Há três modos de conectar um canal customizado do IBM MQ para WCF a um gerenciador de filas. Considere qual tipo de conexão melhor se adequa aos seus requisitos.

Para obter informações adicionais sobre opções de conexão, consulte: “Diferenças de Conexão” na página 589

Para obter informações adicionais sobre arquitetura WCF, veja: “Arquitetura do WCF” na página 1282

Conexão do Cliente Não Gerenciada

Uma conexão feita neste modo se conecta como um cliente IBM MQ a um servidor IBM MQ em execução na máquina local ou em uma máquina remota.

Para usar o canal customizado do IBM MQ para WCF como um cliente IBM MQ, é possível instalá-lo, com o IBM MQ MQI client, no servidor do IBM MQ ou em uma máquina separada.

Conexão do Servidor Não Gerenciada

Quando usado no modo de ligações do servidor, o canal customizado do IBM MQ para WCF usa a API do gerenciador de filas, em vez de se comunicar através de uma rede. Usar conexões de ligações fornece melhor desempenho para aplicativos IBM MQ do que usar conexões de rede.

Para usar a conexão de ligações, deve-se instalar o canal customizado do IBM MQ para WCF no servidor IBM MQ.

Conexão do Cliente Gerenciada

Uma conexão feita neste modo se conecta como um cliente IBM MQ a um servidor IBM MQ em execução na máquina local ou em uma máquina remota.

As classes de canal customizadas IBM MQ para o .NET 3 que se conecta deste modo permanecem no código gerenciado pelo .NET e não fazem chamadas para serviços nativos. Para obter informações adicionais sobre código gerenciado, veja a documentação do Microsoft.

Existem várias limitações no uso do cliente gerenciado. Para obter mais informações sobre essas limitações, consulte “Conexões do Cliente Gerenciadas” na página 589.

Criando e configurando o canal customizado de IBM MQ para WCF

Os canais customizados do IBM MQ para WCF funcionam da mesma maneira que os canais WCF de transporte oferecidos pelo Microsoft. O canal customizado do IBM MQ para WCF pode ser criado de uma de duas maneiras.

Sobre esta tarefa

O canal customizado do IBM MQ é integrado com o WCF como um canal de transporte WCF e, como tal, deve ser unido a um codificador de mensagem e canais de protocolo opcionais para que possa criar uma pilha completa de canais que possa ser usada por um aplicativo. Dois elementos são necessários para que uma pilha completa de canais para seja criada com êxito:

1. Uma definição de ligação: Especifica quais elementos são necessários para construir a pilha de canais de aplicativos, incluindo canal de transporte, codificador de mensagens e quaisquer protocolos além das definições gerais de configuração. Para o canal customizado, a definição de ligação deve ser criada na forma de uma ligação customizada WCF.
2. Uma definição de terminal: Vincula o contrato de serviço à definição de ligação e também fornece o URI de conexão real que descreve onde o aplicativo pode se conectar. Para o canal customizado, o URI está no formato de um SOAP através do URI JMS.

Estas definições podem ser criadas de duas maneiras diferentes:

- Administrativamente: as definições são criadas fornecendo os detalhes em um arquivo de configuração do aplicativo (por exemplo: `app.config`).
- Programaticamente: As definições são criadas diretamente do código do aplicativo.

A decisão sobre qual método usar para criar as definições deve ser baseada nos requisitos do aplicativo conforme segue:

- O método administrativo para configuração fornece a flexibilidade para alterar os detalhes do serviço e pós-implementação de cliente sem reconstruir o aplicativo.
- O método programático para configuração fornece maior proteção contra erros de configuração e a capacidade de gerar dinamicamente uma configuração no tempo de execução.

Criando um Canal Customizado do WCF Administrativamente Fornecendo Informações de Ligação e Terminal em um Arquivo de Configuração do Aplicativo

O canal customizado do IBM MQ para WCF é um canal WCF no nível de transporte. Um terminal e ligação devem ser definidos para usar o canal customizado e estas definições podem ser feitas fornecendo as informações de ligação e terminal em um arquivo de configuração do aplicativo.

Para configurar e usar o canal customizado do IBM MQ para WCF, que é um canal WCF de nível de transporte, uma ligação e uma definição de terminal devem ser definidas. A ligação contém as informações de configuração para o canal e a definição de terminal contém os detalhes da conexão. Estas definições podem ser criados de duas formas:

- Programaticamente diretamente a partir do código do aplicativo, conforme descrito aqui: “Criando um Canal Customizado do WCF Fornecendo Informações de Ligação e Terminal Programaticamente” na página 1293
- Administrativamente, fornecendo os detalhes em um arquivo de configuração do aplicativo, conforme descrito no procedimento a seguir.

O arquivo de configuração do aplicativo de serviço ou cliente normalmente é denominado *yourappname*`.exe.config`, em que *yourappname* é o nome da aplicação. O arquivo de configuração de aplicativo é modificado mais facilmente usando a ferramenta de editor de configuração de serviço Microsoft chamada `SvcConfigEditor.exe` da seguinte maneira:

- Inicie a ferramenta de editor de configuração `SvcConfigEditor.exe`. O local de instalação padrão para a ferramenta é: *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe`, em que *Drive:* é o nome da unidade de instalação.

Etapa 1: Incluir uma Extensão de Elemento de Ligação para Ativar o WCF para Localizar o Canal Customizado

1. Clique com o botão direito em **Avançado** > **Extensão** > **elemento de ligação** para abrir o menu e selecione **Novo**
2. Preencha os campos conforme mostrado nesta tabela:

Tabela 190. Novos Campos do Elemento de Ligação	
Campo	**Value**
Nome	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`
Tipo	Navegue para `IBM.XMS.WCF.dll` no Global Assembly Cache (GAC) e selecione `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig`

Etapa 2: Criar uma Definição de Ligação Customizada que Une o Canal Customizado a um Codificador de Mensagem do WCF

1. Clique com o botão direito em **Ligações** para abrir o menu e selecione **Nova configuração de ligação**
2. Preencha os campos conforme mostrado nesta tabela:

Tabela 191. Novos Campo de Configuração de Ligação	
Campo	**Value**
Nome	`CustomBinding_WMQ`
BindingElement 1	`textMessageEncoding (MessageVersion: Soap11)`
BindingElement 2	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Etapa 3: Especificar as Propriedades de Ligação

1. Selecione a ligação de transporte de *IBM.XMS.WCF.SoapJmsIbmTransportChannel* a partir da ligação que você criou em: "Etapa 2: Criar uma Definição de Ligação Customizada que Une o Canal Customizado a um Codificador de Mensagem do WCF" na página 1292
2. Faça qualquer alteração necessária nos valores-padrão das propriedades conforme descrito em: "Opções de Configuração de Ligação do WCF" na página 1296

Step 4: Criar uma Definição de Terminal

Crie uma definição de terminal que faça referência à ligação customizada criada em "Etapa 2: Criar uma Definição de Ligação Customizada que Une o Canal Customizado a um Codificador de Mensagem do WCF" na página 1292 e forneça os detalhes da conexão do serviço. A maneira como estas informações são especificadas é dependente de se a definição é para um aplicativo cliente ou um aplicativo de serviço.

Para um aplicativo cliente, inclua uma definição de terminal na seção do cliente conforme a seguir:

1. Clique com o botão direito em **Cliente** > **Terminais** para abrir o menu e selecione **Novo terminal do cliente**
2. Preencha os campos conforme mostrado nesta tabela:

Tabela 192. Novos Campos de Terminal do Cliente	
Campo	**Value**
Nome	`Endpoint_WMQ`
Endereço	*O URI SOAP/JMS que descreve os detalhes de conexão do WMQ necessários para acessar o serviço.* Para obter detalhes adicionais, consulte: "Canal customizado do IBM MQ para o formato de endereço de URI do terminal WCF" na página 1295
Ligação	`customBinding`

Tabela 192. Novos Campos de Terminal do Cliente (continuação)	
Campo	**Value**
BindingConfiguration	`CustomBinding_WMQ`
Contrato	*O nome de sua interface de contrato do serviço*

Para um aplicativo de serviço, inclua uma definição de serviço na seção de serviços conforme a seguir:

1. Clique com o botão direito em **Serviços** para abrir o menu e selecione **Novo Serviço** e, em seguida, selecione a classe de serviço a ser hospedada.
2. Inclua uma definição de terminal à seção **Terminais** ao seu novo serviço e preencha os campos conforme mostrado nesta tabela:

Tabela 193. Novos Campos do Terminal em Serviço	
Campo	**Value**
Nome	`Endpoint_WMQ`
Endereço	*O URI SOAP/JMS que descreve os detalhes de conexão do WMQ necessários para acessar o serviço.* Para obter detalhes adicionais, consulte: "Canal customizado do IBM MQ para o formato de endereço de URI do terminal WCF" na página 1295
Ligação	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contrato	*O nome de sua classe de implementação de serviço*

Criando um Canal Customizado do WCF Fornecendo Informações de Ligação e Terminal Programaticamente

O canal customizado do IBM MQ para WCF é um canal WCF no nível de transporte. Um terminal e uma ligação devem ser definidos para usar o canal customizado e estas definições podem ser feitas programaticamente diretamente a partir do código do aplicativo.

Para configurar e usar o canal customizado do IBM MQ para WCF, que é um canal WCF de nível de transporte, uma ligação e uma definição de terminal devem ser definidas. A ligação contém as informações de configuração para o canal e a definição de terminal contém os detalhes da conexão. Para obter informações adicionais, consulte "Usando as Amostras do WCF" na página 1303.

Estas definições podem ser criados de duas formas:

- Administrativamente, fornecendo os detalhes em um arquivo de configuração do aplicativo, conforme descrito em "Criando um Canal Customizado do WCF Administrativamente Fornecendo Informações de Ligação e Terminal em um Arquivo de Configuração do Aplicativo" na página 1291.
- Programaticamente diretamente do código do aplicativo, conforme descrito nos seguintes subtópicos.

Definindo informações de ligação e terminal programaticamente: interface SOAP/JMS

Para a interface SOAP/JMS, é possível definir um terminal e uma ligação programaticamente direto no código do aplicativo.

Sobre esta tarefa

Para fornecer informações de ligação e terminal programaticamente, inclua o código necessário no aplicativo concluindo as etapas a seguir.

Procedimento

1. Criar uma instância do elemento de ligação de transporte do canal incluindo o seguinte código no aplicativo:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. Configure todas as propriedades de ligação necessárias, por exemplo, incluindo o seguinte código em seu aplicativo para configurar `ClientConnectionMode`:

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. Crie uma ligação customizada cria um par do canal de transporte com um codificador de mensagem incluindo o código a seguir para seu aplicativo:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. Crie o URI SOAP/JMS.

   O URI SOAP/JMS que descreve os detalhes da conexão IBM MQ necessários para acessar o serviço deve ser fornecido como o endereço do terminal. O endereço que você especifica depende de se o canal está sendo usado para um aplicativo de serviço ou um aplicativo cliente.

   - Para aplicativos clientes, o URI SOAP/JMS deve ser criado como um `EndpointAddress` conforme a seguir:

```
EndpointAddress address = new EndpointAddress("jms:/queue?
destination=SampleQ@QM1&connectionFactory
=connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

   - Para aplicativos de serviços, o URI SOAP/JMS deve ser criado como um URI conforme a seguir:

```
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

   Para obter mais informações sobre endereços de terminal, veja “Canal customizado do IBM MQ para o formato de endereço de URI do terminal WCF” na página 1295.

Definindo informações de ligação e terminal programaticamente: interface não SOAP/não JMS
Para interface não SOAP/não JMS, é possível definir um terminal e uma ligação programaticamente de forma direta a partir do código do aplicativo.

Sobre esta tarefa

Para fornecer informações de ligação e terminal programaticamente, inclua o código necessário no aplicativo concluindo as etapas a seguir.

Procedimento

1. Crie um WmqBinding incluindo o seguinte código no aplicativo:

```
WmqBinding binding = new WmqBinding();
```

   Esse código cria uma ligação que une o WmqMsgEncodingElement e WmqIbmTransportBindingElement necessário para a interface não SOAP/não JMS.

2. Forneça o URI wmq:// que descreve os detalhes de conexão do IBM MQ necessários para acessar o serviço.

A maneira na qual você fornece o URI wmq:// depende se o canal está sendo usado para um aplicativo de serviço ou um aplicativo cliente.

- Para aplicativos clientes, o URI wmq:// deve ser criado como um EndpointAddress, conforme a seguir:

```
EndpointAddress address = new EndpointAddress
("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

- Para aplicativos de serviços, o URI wmq:// deve ser criado como um URI conforme a seguir:

```
Uri sampleAddress = new Uri(
"wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

Canal customizado do IBM MQ para o formato de endereço de URI do terminal WCF

Um serviço da web é especificado usando um Universal Resource Identifier (URI) que fornece detalhes de local e de conexão. O formato do URI depende de se você está usando a interface SOAP/JMS ou a interface não SOAP/nãoJMS.

Interface SOAP/JMS

O formato do URI suportado no transporte do IBM MQ para SOAP permite um grau abrangente de controle sobre parâmetros e opções específicos de SOAP/IBM MQ ao acessar serviços de destino. Esse formato é compatível com WebSphere Application Server e com CICS, facilitando a integração de IBM MQ com os dois produtos.

A sintaxe de URI é a seguinte:

```
jms:/queue? name=value&name=value...
```

em que `name` é um nome de parâmetro, *value* é um valor apropriado e o elemento `name` = *value* pode ser repetido qualquer número de vezes com a segunda ocorrência e as subsequentes sendo precedidas por um e comercial (símbolo &).

Os nomes de parâmetros fazem distinção entre maiúsculas e minúsculas, pois são nomes de objetos do IBM MQ. Se qualquer parâmetro for especificado mais de uma vez, a ocorrência final do parâmetro entrará em vigor, o que significa que os aplicativos clientes podem substituir os valores de parâmetro anexando no URI. Se qualquer parâmetro não reconhecido adicional for incluído, ele será ignorado.

Se você armazenar um URI em uma sequência de XML, deverá representar o caractere de e comercial como "&". Da mesma forma, se um URI for codificado em um script, tome cuidado para escapar caracteres como **&** que, de outra forma, seriam interpretados pelo shell.

Esse é um exemplo de um URI simples para um serviço do Axis:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Aqui está um exemplo de um URI simples para um serviço do .NET:

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Somente os parâmetros necessários são fornecidos (`targetService` é necessário para os serviços apenas .NET) e `connectionFactory` não recebe opções.

Neste exemplo de Axis, `connectionFactory` contém várias opções:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
```

```
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Neste exemplo de Axis, a opção `sslPeerName` de `connectionFactory` também foi especificada. O valor de sslPeerName em si contém pares de nome-valor e espaços em branco integrados significativos:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Interface não SOAP/não JMS

O formato do URI para a interface NÃO SOAP/não JMS permite um grau abrangente de controle sobre opções e parâmetros específicos do IBM MQ ao acessar serviços de destino.

A sintaxe de URI é a seguinte:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

Esse IRI informa um solicitante de serviço que pode usar uma conexão de ligação de cliente TCP do IBM MQ com uma máquina chamada example.com na porta 1415 e colocar mensagens de solicitação persistentes em uma fila chamada INS.QUOTE.REQUEST no gerenciador de filas MOTOR.INS. O IRI especifica que o provedor de serviços enviará as respostas para uma fila chamada INS.QUOTE.REPLY no BRANCH452 do gerenciador de filas. Formato do URI é conforme especificado para SupportPac As MA93. Veja SupportPac MA93: IBM MQ – definição de serviço para obter mais detalhes sobre as especificações do IRI de IBM MQ.

Opções de Configuração de Ligação do WCF

Existem duas maneiras de aplicar as opções de configuração para as informações de ligação de canais customizados. Você também configurar as propriedades administrativamente ou configurá-las programaticamente.

As opções de configuração de ligação podem ser configuradas de uma de duas maneiras diferentes:

1. Administrativamente: as configurações de propriedade de ligação devem ser especificadas na seção de transporte da definição de ligação customizada no arquivo de configuração dos aplicativos, por exemplo: `app.config`.
2. Programaticamente: O código do aplicativo deve ser modificado para especificar a propriedade durante a inicialização da ligação customizada.

Configurando as Propriedades de Ligação Administrativamente

As configurações de propriedade de ligação podem ser especificadas no arquivo de configuração do aplicativo, por exemplo: `app.config`. O arquivo de configuração é gerado pelo **`svcutil`**, conforme mostrado nos seguintes exemplos.

Interface SOAP/JMS

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

Interface não SOAP/não JMS

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
```

```
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

Configurando as Propriedades de Ligação Programaticamente

Para incluir uma propriedade de ligação de WCF para especificar o modo de conexão do cliente, você deve modificar o código de serviço para especificar a propriedade durante a inicialização da ligação customizada.

Use o exemplo a seguir para especificar o modo de conexão do cliente não gerenciado:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

Propriedades de Ligação do WCF

Tabela 194. Os valores das propriedades de ligação ao configurar programaticamente ou administrativamente

Nome da Propriedade	Aplicativo Cliente ou de Serviço	Valor Administrativo	Valor Programático	Descrição
maxBufferPoolSize	Ambos	0 para 64 bit número inteiro assinado	0 para 64 bit número inteiro assinado	Especifica o tamanho máximo da memória que pode ser utilizada para armazenar os buffers de mensagem do WCF para uma instância do canal.
maxMessageSize	Ambos	1 para 32 bit número inteiro assinado	1 para 32 bit número inteiro assinado	Especifica a memória máxima que pode ser utilizada para uma mensagem do WCF individual.
clientConnectionMode	Ambos	0 (valor padrão) 1	AS_URI (valor padrão) CLIENT_UNMANAGED	Especifica o modo de conexão do cliente do canal de transporte. 0 significa que o modo de conexão do cliente é conforme especificado no URI. Usado somente se a conexão do cliente for usada. Especifica que o modo de conexão do cliente é conforme especificado no URI. 0 é o valor padrão se nenhum modo de conexão do cliente estiver configurado. 1 significa que o modo de conexão do cliente é um cliente não gerenciado. Usado somente se a conexão do cliente for usada.

Tabela 194. Os valores das propriedades de ligação ao configurar programaticamente ou administrativamente (continuação)

Nome da Propriedade	Aplicativo Cliente ou de Serviço	Valor Administrativo	Valor Programático	Descrição
MaxConcurrentCalls	Client	O intervalo é de 0–2 147 483 647 16 é o valor padrão	O intervalo é de 0–2 147 483 647 16 é o valor padrão	Essa propriedade define o número máximo de operações simultâneas que podem ocorrer em um proxy de cliente individual a qualquer momento. Se mais operações forem iniciadas, elas serão enfileiradas até que uma operação em andamento seja concluída ou expire. Esta configuração pode ser usada para controlar o máximo de encadeamentos e recursos que podem ser consumidos por um proxy individual. `0` remove este limite, possibilitando que todas as operações sejam tentadas simultaneamente.
MaxConcurrentCalls	Serviço	O intervalo é de 1–2 147 483 647 16 é o valor padrão	O intervalo é de 1–2 147 483 647 16 é o valor padrão	Esta propriedade é usada apenas se o recurso de entrega assegurada for ativado (para obter mais informações sobre a entrega assegurada, veja "Entrega Assegurada do Canal Customizado WCF" na página 1286). Ela especifica o número máximo de operações simultâneas que podem estar em andamento ao mesmo tempo para o terminal determinado. É necessário tomar cuidado ao alterar esta configuração. Cada operação simultânea requer recursos adicionais, em particular uma nova instância do canal customizado e os encadeamentos associados do conjunto de encadeamentos para acionar os pedidos. A alocação em excesso pode ser contraproducente e afetar o desempenho gravemente. A configuração apropriada do conjunto de encadeamentos deve ser feita para suportar esta propriedade.

Construindo e Hospedando Serviços para WCF

Visão geral dos serviços do Microsoft Windows Communication Foundation (WCF) explicando como criar e configurar serviços do WCF.

O canal customizado do IBM MQ para WCF e serviços do WCF que o usam pode ser hospedado pelos seguintes métodos:

- Auto-hosting
- Serviço Windows

O canal customizado do IBM MQ para o WCF não pode ser hospedado no Windows Process Activation Service.

Os tópicos a seguir fornecem alguns exemplos de auto-hosting simples para demonstrar as etapas envolvidas. A documentação on-line do WCF do Microsoft, que contém informações adicionais e os detalhes mais recentes, pode ser localizada no website MSDN do Microsoft em https://msdn.microsoft.com.

Construindo Aplicativos de Serviço WCF Usando o Método 1: Auto-hospedando Administrativamente Usando um Arquivo de Configuração do Aplicativo

Depois de criar um arquivo de configuração do aplicativo, abra uma instância do serviço e incluir o código especificado para seu aplicativo.

Antes de começar

Crie ou edite um arquivo de configuração do aplicativo para o serviço, conforme descrito em: "Criando um Canal Customizado do WCF Administrativamente Fornecendo Informações de Ligação e Terminal em um Arquivo de Configuração do Aplicativo" na página 1291

Sobre esta tarefa

1. Instancie e abra uma instância do serviço no host do serviço. O tipo de serviço deve ser o mesmo que o tipo de serviço especificado no arquivo de configuração do serviço.
2. Inclua o seguinte código em seu aplicativo:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

Construindo Aplicativos de Serviço do WCF Usando o Método 2: Auto-hospedando Programaticamente Diretamente a partir do Aplicativo

Inclua as propriedades de ligação, crie o host de serviço com uma instância da classe de serviço necessária e abrir o serviço.

Antes de começar

1. Inclua uma referência no arquivo `IBM.XMS.WCF.dll` do canal customizado para o projeto. O `IBM.XMS.WCF.dll` está no *`WMQInstallDir`*`\bin`, em que *WMQInstallDir* é o diretório no qual o IBM MQ está instalado.
2. Inclua uma instrução *using* no namespace `IBM.XMS.WCF`, por exemplo: `using IBM.XMS.WCF`
3. Crie uma instância do elemento de ligação de canais e terminal conforme descrito em: "Criando um Canal Customizado do WCF Fornecendo Informações de Ligação e Terminal Programaticamente" na página 1293

Sobre esta tarefa

Se as mudanças para as propriedades de ligação do canal forem necessárias, conclua as seguintes etapas:

1. Inclua as propriedades de ligação em `transportBindingElement` conforme mostrado no exemplo a seguir:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Crie o host de serviço com uma instância da classe de serviço necessária:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. Abra o serviço:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

Expondo Metadados Usando um Terminal HTTP

Instruções para expor os metadados de um serviço configurado para usar o canal customizado do IBM MQ para WCF.

Sobre esta tarefa

Se os metadados de serviços tiverem que ser expostos (para que ferramentas como `svcutil` possam acessá-los diretamente do serviço em execução em vez de por meio de um arquivo WSDL off-line, por exemplo), isso deverá ser feito expondo os metadados de serviços com um terminal HTTP. As etapas a seguir podem ser usadas para incluir este terminal adicional.

1. Inclua o endereço de base de onde os metadados devem ser expostos no ServiceHost, por exemplo:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Inclua o seguinte código no ServiceHost antes do serviço ser aberto:

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

Resultados

Agora os metadados estão disponíveis no endereço a seguir: `http://localhost:8000/MyService`

Construindo Aplicativos Clientes para WCF

Visão geral da geração e construção de aplicativos clientes do Microsoft Windows Communication Foundation (WCF).

Um aplicativo cliente pode ser criado para um serviço WCF; os aplicativos clientes são, geralmente, gerados usando o Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) para criar os arquivos de configuração e de proxy necessários que podem ser usados diretamente pelo aplicativo.

Gerando um Proxy de Cliente do WCF e Arquivos de Configuração do Aplicativo Usando a Ferramenta svcutil com Metadados de um Serviço em Execução

Instruções para usar a ferramenta svcutil.exe da Microsoft para gerar um cliente para um serviço configurado para usar o canal customizado do IBM MQ para WCF.

Antes de começar

Há três pré-requisitos para usar a ferramenta svcutil para criar arquivos de configuração e proxy necessários que podem ser usados diretamente pelo aplicativo:

- O serviço do WCF deve estar em execução antes da ferramenta svcutil ser iniciada.
- O serviço do WCF deve expor seus metadados usando uma porta HTTP além das referências de terminal de canal customizado IBM MQ para gerar um cliente diretamente de um serviço em execução.
- O canal customizado deve ser registrado nos dados de configuração para svcutil.

Sobre esta tarefa

As etapas a seguir explicam como gerar um cliente para um serviço configurado para usar o canal customizado do IBM MQ, mas também expõe seus metadados no tempo de execução através de uma porta HTTP separada:

1. Inicie o serviço do WCF (O serviço deve estar em execução antes da ferramenta svcutil ser iniciada).
2. Inclua os detalhes do arquivo de configuração `svcutil.exe` da raiz da instalação, no arquivo de configuração svcutil ativo, normalmente o `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config`, para que o svcutil reconheça o canal customizado IBM MQ.
3. Execute svcutil a partir de um prompt de comandos, por exemplo:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. Copie os arquivos `app.config` e `YourService.cs` gerados para o projeto cliente do Visual Studio Microsoft.

Como proceder a seguir

Se os metadados de serviços não puderem ser recuperados diretamente, svcutil poderá ser usada para gerar os arquivos de cliente a partir de wsdl. Para obter informações adicionais, consulte: “Gerando um Proxy de Cliente do WCF e Arquivos de Configuração de Aplicativo Usando a Ferramenta svcutil com WSDL” na página 1301

Gerando um Proxy de Cliente do WCF e Arquivos de Configuração de Aplicativo Usando a Ferramenta svcutil com WSDL

Instruções para gerar clientes WCF a partir do WSDL se os metadados do serviço está indisponível.

Se não for possível recuperar os metadados do serviço diretamente para gerar um cliente a partir dos metadados de um serviço em execução, svcutil poderá ser usado para gerar os arquivos de cliente a partir do WSDL. As modificações a seguir devem ser feitas no WSDL para especificar que o canal customizado do IBM MQ deve ser usado:

1. Inclua as seguintes definições de espaço de nomes e informações de política:

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
```

```
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. Modifique a seção de ligações para fazer referência à nova seção de política e remova qualquer definição de `transport` do elemento de ligação subjacente:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Execute svcutil a partir de um prompt de comandos, por exemplo:

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

Construindo Aplicativos Clientes WCF Usando um Proxy de Cliente com um Arquivo de Configuração de Aplicativo

Antes de começar

Crie ou edite um arquivo de configuração de aplicativo para o cliente, conforme descrito em: “Criando um Canal Customizado do WCF Administrativamente Fornecendo Informações de Ligação e Terminal em um Arquivo de Configuração do Aplicativo” na página 1291

Sobre esta tarefa

Instancie e abra uma instância do proxy de cliente. O parâmetro passado para o proxy gerado deve ser o mesmo que o nome de terminal especificado no arquivo de configuração de cliente, por exemplo `Endpoint_WMQ`:

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

Construindo Aplicativos Cliente WCF Usando um Proxy de Cliente com Configuração Programática

Antes de começar

1. Inclua uma referência no arquivo `IBM.XMS.WCF.dll` do canal customizado para o projeto. O `IBM.XMS.WCF.dll` está no diretório *`WMQInstallDir`*`\bin`, em que *WMQInstallDir* é o diretório no qual o IBM MQ está instalado.

2. Inclua uma instrução *using* no namespace `IBM.XMS.WCF`, por exemplo: `using IBM.XMS.WCF`
3. Crie uma instância do elemento de ligação th' e terminal do canal conforme descrito em: "Criando um Canal Customizado do WCF Fornecendo Informações de Ligação e Terminal Programaticamente" na página 1293

Sobre esta tarefa

Se as mudanças para as propriedades de ligação do canal forem necessárias, conclua as etapas a seguir.

1. Inclua as propriedades de ligação em `transportBindingElement` conforme mostrado na figura a seguir:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Crie o proxy de cliente conforme mostrado na figura a seguir, em que *binding* e *endpoint address* são a ligação e o endereço de terminal configurados na etapa 1 e passados em:

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
           try {
                  myClient.myMethod("HelloWorld!");
                  myClient.Close();
           }
           catch (TimeoutException e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
           catch (CommunicationException e) {
               Console.Out.WriteLine(e);
                              myClient.Abort();
           }
           catch (Exception e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
```

Usando as Amostras do WCF

As amostras do Windows Communication Foundation (WCF) fornecem alguns exemplos simples de como o canal customizado do IBM MQ pode ser usado.

Para construir os projetos de amostra, o Microsoft.NET 3.5 SDK ou o Microsoft Visual Studio 2008 é necessário.

Amostra do WCF de Cliente e Servidor Unidirecional Simples

Essa amostra demonstra o canal customizado do IBM MQ que está sendo usado para iniciar um serviço do Windows Communication Foundation (WCF) a partir de um cliente WCF que está usando um formato de canal unidirecional.

Sobre esta tarefa

O serviço implementa um único método que envia uma sequência para o console. O cliente foi gerado usando a ferramenta `svcutil` para recuperar os metadados de serviço de um terminal HTTP exposto separadamente, conforme descrito em "Gerando um Proxy de Cliente do WCF e Arquivos de Configuração do Aplicativo Usando a Ferramenta svcutil com Metadados de um Serviço em Execução" na página 1301

A amostra foi configurada com nomes de recurso específicos conforme descrito no procedimento a seguir. Se for necessário mudar os nomes de recursos, você também deverá mudar o valor correspondente no aplicativo cliente no arquivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` e no

aplicativo de serviço no arquivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ. Para obter mais informações sobre a formatação do URI do terminal JMS, consulte *IBM MQ Transporte para SOAP* na documentação do produto IBM MQ. Se você precisar modificar a solução de amostra e a origem, será necessário um IDE, por exemplo, Microsoft Visual Studio 8 ou superior.

Procedimento

1. Crie um gerenciador de filas chamado *QM1*
2. Crie um destino de fila chamado *SampleQ*
3. Inicie o serviço para que o listener fique esperando por mensagens: execute o arquivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ.
4. Execute o cliente uma vez: execute o arquivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ.

   O aplicativo cliente faz loop cinco vezes enviando cinco mensagens para *SampleQ*

Resultados

O aplicativo de serviço obtém as mensagens de *SampleQ* e exibe `Hello World` na tela cinco vezes.

Como proceder a seguir

Amostra do WCF do Cliente e Servidor de Pedido-Resposta Simples

Essa amostra demonstra o canal customizado do IBM MQ sendo usado para iniciar um serviço do Windows Communication Foundation (WCF) a partir de um cliente WCF usando um formato de canal de solicitação-resposta.

Sobre esta tarefa

Este serviço fonece alguns métodos calculadores simples para adicionar e subtrair dois números e, em seguida, retornar o resultado. O cliente foi gerado usando a ferramenta `svcutil` para recuperar os metadados de serviço de um terminal HTTP exposto separadamente, conforme descrito em "Gerando um Proxy de Cliente do WCF e Arquivos de Configuração do Aplicativo Usando a Ferramenta svcutil com Metadados de um Serviço em Execução" na página 1301

A amostra foi configurada com nomes de recurso específicos como no procedimento descrito a seguir. Se for necessário mudar os nomes de recursos, você também deverá mudar o valor correspondente no aplicativo cliente no arquivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config` e no aplicativo de serviço no arquivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ. Para obter mais informações sobre a formatação do URI do terminal JMS, consulte *IBM MQ Transporte para SOAP* na documentação do produto IBM MQ. Se você precisar modificar a solução de amostra e a origem, será necessário um IDE, por exemplo, Microsoft Visual Studio 8 ou superior.

Procedimento

1. Crie um gerenciador de filas chamado *QM1*
2. Crie um destino de fila chamado *SampleQ*
3. Crie um destino de fila chamado *SampleReplyQ*

4. Inicie o serviço para que o listener fique esperando por mensagens: execute o arquivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ.
5. Execute o cliente uma vez: execute o arquivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ.

Resultados

Quando o cliente tiver sido executado, o seguinte processo será iniciado e repetirá quatro vezes para que um total de cinco mensagens serão enviadas cada maneira:

1. O cliente coloca uma mensagem de pedido em *SampleQ* e aguarda uma resposta.
2. O serviço obtém a mensagem de pedido de *SampleQ*.
3. O serviço adiciona e subtrai alguns valores usando o conteúdo da mensagem.
4. O serviço então coloca os resultados em uma mensagem em *SampleReplyQ* e aguarda o cliente colocar uma nova mensagem.
5. O cliente obtém a mensagem de *SampleReplyQ* e exibe os resultados na tela.

Como proceder a seguir

Cliente WCF para um serviço .NET hospedado por uma amostra de IBM MQ

Aplicativos clientes de amostra e aplicativos do proxy de serviço de amostra são fornecidos para ambos .NET e Java. As amostras são baseadas em um serviço de Cota de Ações que obtém um pedido para uma cota de ações e, em seguida, fornece a cota de ações.

Antes de começar

A amostra requer que o .NET SOAP através do ambiente de hospedagem do serviço JMS esteja corretamente instalado e configurado no IBM MQ e seja acessível a partir de um gerenciador de filas locais.

Quando o .NET SOAP através ambiente de hospedagem do serviço JMS estiver corretamente instalado e configurado no IBM MQ e for acessível a partir de um gerenciador de filas locais, etapas de configuração adicionais deverão ser concluídas.

1. Configure a variável de ambiente **WMQSOAP_HOME** para o diretório de instalação IBM MQ , por exemplo: `C:\Program Files\IBM\MQ`
2. Certifique-se de que o compilador Java `javac` esteja disponível e no PATH.
3. Copie o arquivo `axis.jar` do diretório `prereqs/axis` da imagem de instalação para o diretório de produção IBM MQ , por exemplo: `C:\Program Files\IBM\MQ\java\lib\soap`
4. Inclua no PATH: *MQ_INSTALLATION_PATH*`\Java\lib`, em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado, por exemplo: `C:\Program Files\IBM\MQ`
5. Certifique-se de que o local do .NET esteja especificado corretamente em *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd`, em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado, por exemplo: `C:\Program Files\IBM\MQ`. O local de .NET pode ser especificado, por exemplo: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Quando as etapas anteriores forem concluídas, teste e execute o serviço:

1. Navegue até o SOAP no diretório ativo do JMS.
2. Insira um dos comandos a seguir para executar o teste de verificação e deixe o listener de serviço em execução:

- Para .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.
- Para AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.

O argumento hold mantém os listeners em execução após o teste ser concluído.

Caso sejam relatados erros durante esta configuração, é possível remover todas as mudanças para que o procedimento possa ser reiniciado da seguinte maneira:

1. Exclua o SOAP gerado através do diretório JMS.
2. Exclua o gerenciador de filas.

Sobre esta tarefa

Esta amostra demonstra uma conexão de um cliente WCF com o .NET SOAP através do serviço de amostra JMS fornecido no IBM MQ usando uma forma de canal unidirecional. O serviço implementa um exemplo simples de StockQuote, que emita uma sequência de texto para o console.

O cliente foi gerado usando o WSDL para gerar arquivos de clientes conforme descrito em "Gerando um Proxy de Cliente do WCF e Arquivos de Configuração de Aplicativo Usando a Ferramenta svcutil com WSDL" na página 1301

A amostra foi configurada com nomes de recurso específicos conforme descrito no procedimento a seguir. Se for necessário mudar os nomes de recursos, você também deverá mudar o valor correspondente no aplicativo cliente no arquivo *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\app.config e no aplicativo de serviço no arquivo *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl, em que *MQ_INSTALLATION_PATH* representa o diretório de instalação para IBM MQ. Para obter mais informações sobre a formatação do URI do terminal JMS, consulte *IBM MQ Transporte para SOAP* na documentação do produto IBM MQ.

Procedimento

Execute o cliente uma vez: execute o arquivo *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe, em que *MQ_INSTALLATION_PATH* representa o diretório de instalação para IBM MQ.

O aplicativo cliente faz loop cinco vezes enviando cinco mensagens à fila de amostra.

Resultados

O aplicativo de serviço recebe as mensagens da fila de amostra e exibe Hello World cinco vezes na tela.

Cliente WCF para um serviço Axis Java hospedado por amostra de IBM MQ

Aplicativos clientes de amostra e aplicativos do proxy de serviço de amostra são fornecidos para ambos Java e .NET. As amostras são baseadas em um serviço de Cota de Ações que obtém um pedido para uma cota de ações e, em seguida, fornece a cota de ações.

Antes de começar

Esta amostra requer que o .NET SOAP através do ambiente de hospedagem do serviço JMS esteja corretamente instalado e configurado no IBM MQ e seja acessível a partir de um gerenciador de filas locais.

Quando o .NET SOAP através ambiente de hospedagem do serviço JMS estiver corretamente instalado e configurado no IBM MQ e for acessível a partir de um gerenciador de filas locais, etapas de configuração adicionais deverão ser concluídas.

1. Configure a variável de ambiente **WMQSOAP_HOME** para o diretório de instalação IBM MQ , por exemplo: `C:\Program Files\IBM\MQ`
2. Certifique-se de que o compilador Java `javac` esteja disponível e no PATH.
3. Copie o arquivo `axis.jar` do diretório `prereqs/axis` da imagem de instalação para o diretório de instalação IBM MQ .
4. Inclua no PATH: *MQ_INSTALLATION_PATH*`\Java\lib`, em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado, por exemplo: `C:\Program Files\IBM\MQ`
5. Certifique-se de que o local do .NET esteja especificado corretamente em *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd`, em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado, por exemplo: `C:\Program Files\IBM\MQ`. O local de .NET pode ser especificado, por exemplo: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Quando as etapas anteriores forem concluídas, teste e execute o serviço:

1. Navegue até o SOAP no diretório ativo do JMS.
2. Insira um dos comandos a seguir para executar o teste de verificação e deixe o listener de serviço em execução:
   - Para .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.
   - Para AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` em que *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.

O argumento `hold` mantém os listeners em execução após o teste ser concluído.

Caso sejam relatados erros durante esta configuração, é possível remover todas as mudanças de modo que o procedimento seja reiniciado da seguinte maneira:

1. Exclua o SOAP gerado através do diretório JMS.
2. Exclua o gerenciador de filas.

Sobre esta tarefa

A amostra demonstra uma conexão de um cliente WCF com o Axis Java SOAP através da amostra do serviço do JMS fornecido no IBM MQ usando um formato de canal unidirecional. O serviço implementa um exemplo simples de StockQuote, que emite uma sequência de texto para um arquivo salvo no diretório atual.

O cliente foi gerado usando o WSDL para gerar arquivos de clientes conforme descrito em "Gerando um Proxy de Cliente do WCF e Arquivos de Configuração de Aplicativo Usando a Ferramenta svcutil com WSDL" na página 1301

A amostra foi configurada com nomes de recurso específicos conforme descrito neste parágrafo. Se for necessário mudar os nomes de recursos, você também deverá mudar o valor correspondente no aplicativo cliente no arquivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\client\app.config` e no aplicativo de serviço no arquivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl`, em que *MQ_INSTALLATION_PATH* representa o diretório de instalação para IBM MQ.

Procedimento

Execute o cliente uma vez: execute o arquivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe`, em que *MQ_INSTALLATION_PATH* representa o diretório de instalação para IBM MQ.

O aplicativo cliente faz loop cinco vezes enviando cinco mensagens à fila de amostra.

Resultados

O aplicativo de serviço recebe as mensagens da fila de amostra e inclui `Hello World` cinco vezes em um arquivo no diretório atual.

Cliente WCF para o serviço Java hospedado pela amostra WebSphere Application Server

Aplicativos clientes de amostra e aplicativos do proxy de serviço de amostra são fornecidos para WebSphere Application Server 6. Um pedido de serviço pedido-resposta também é fornecido.

Antes de começar

Essa amostra requer o uso da seguinte configuração: IBM MQ

Tabela 195. Configuração necessária do IBM MQ

Object	**Nome necessário**
Gerenciador de filas	QM1
Fila local	HelloWorld
Fila local	HelloWorldReply

Esta amostra também requer que um ambiente de hospedagem WebSphere Application Server 6 esteja corretamente instalado e configurado. O WebSphere Application Server 6 usa uma conexão de modo de ligação para conectar-se ao IBM MQ por padrão. Portanto, o WebSphere Application Server 6 deve ser instalado na mesma máquina que o gerenciador de filas.

Depois que o ambiente do WAS for configurado, as seguintes etapas de configuração adicionais deverão ser concluídas:

1. Crie os seguintes objetos JNDI no WebSphere Application Server repositório JNDI:
   a. Um destino de fila do JMS chamado `HelloWorld`
      - Configure o nome JNDI como `jms/HelloWorld`
      - Configure o nome da fila como `HelloWorld`
   b. Um connection factory de fila do JMS chamado `HelloWorldQCF`
      - Configure o nome JNDI como `jms/HelloWorldQCF`
      - Configure o nome do gerenciador de filas como `QM1`
   c. Um connection factory de fila do JMS chamado `WebServicesReplyQCF`
      - Configure o nome JNDI como `jms/WebServicesReplyQCF`
      - Configure o nome do gerenciador de filas como `QM1`
2. Crie uma porta do listener de mensagem chamada `HelloWorldPort` no WebSphere Application Server com a seguinte configuração:
   - Configure o nome JNDI do connection factory como `jms/HelloWorldQCF`
   - Configure o nome JNDI de destino como `jms/HelloWorld`
3. Instale o aplicativo `HelloWorldEJBEAR.ear` de serviço da web para o WebSphere Application Server da seguinte forma:
   a. Clique em **Aplicativos** > **Novo Aplicativo** > **Novo Aplicativo Corporativo**.
   b. Navegue para *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação de IBM MQ.
   c. Não altere nenhuma das opções padrão no assistente e reinicie o servidor de aplicativos depois que o aplicativo tiver sido instalado.

Quando a configuração do WAS tiver sido concluída, teste o serviço executando-o uma vez:

1. Navegue até o SOAP no diretório ativo do JMS.
2. Insira este comando para executar a amostra: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe` em que *MQ_INSTALLATION_PATH* é o diretório de instalação do IBM MQ.

Sobre esta tarefa

A amostra demonstra uma conexão de um cliente WCF com o serviço de amostra de SOAP WebSphere Application Server sobre JMS fornecido nas amostras WCF incluídas no IBM MQ, usando um formato de canal de solicitação-resposta. As mensagens fluem entre o WCF e o WebSphere Application Server usando filas do IBM MQ. O serviço implementa o método `HelloWorld(...)`, que obtém uma sequência e retorna uma saudação para o cliente.

O cliente foi gerado usando a ferramenta svcutil para recuperar os metadados de serviço a partir de um terminal HTTP exposto separadamente conforme descrito em "Gerando um Proxy de Cliente do WCF e Arquivos de Configuração do Aplicativo Usando a Ferramenta svcutil com Metadados de um Serviço em Execução" na página 1301

A amostra foi configurada com nomes de recurso específicos conforme descrito no procedimento a seguir. Se for necessário mudar os nomes de recursos, você também deverá mudar o valor correspondente no aplicativo cliente no arquivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\app.config` e no aplicativo de serviço no *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação do IBM MQ.

O serviço e o cliente se baseiam no serviço e no cliente descritos no artigo do IBM Developer *Construindo um serviço da web do JMS usando SOAP sobre JMS e WebSphere Studio*. Para obter mais informações sobre o desenvolvimento de SOAP sobre serviços da web do JMS que são compatíveis com o canal customizado do IBM MQ WCF, consulte https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

Procedimento

Execute o cliente uma vez: execute o arquivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe`, em que *MQ_INSTALLATION_PATH* é o diretório de instalação para IBM MQ.

O aplicativo cliente inicia os dois métodos de serviço ao mesmo tempo, enviando duas mensagens para a fila de amostra.

Resultados

O aplicativo de serviço recebe as mensagens da fila de amostra e fornece uma resposta para a chamada de método `HelloWorld(...)`, que o aplicativo cliente emite para o console.

Avisos

Estas informações foram desenvolvidas para produtos e serviços oferecidos nos Estados Unidos.

É possível que a IBM não ofereça os produtos, serviços ou recursos discutidos nesta publicação em outros países. Consulte seu representante local do IBM para obter informações sobre produtos e serviços disponíveis atualmente em sua área. Qualquer referência a produtos, programas ou serviços IBM não significa que apenas produtos, programas ou serviços IBM possam ser utilizados. Qualquer outro produto, programa ou serviço, funcionalmente equivalente, poderá ser utilizado em substituição daqueles, desde que não infrinja nenhum direito de propriedade intelectual da IBM. Entretanto, a avaliação e verificação da operação de qualquer produto, programa ou serviço não IBM são de responsabilidade do Cliente.

A IBM pode ter patentes ou aplicativos de patentes pendentes relativas aos assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum sobre tais patentes. é possível enviar pedidos de licença, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
U.S.A.

Para pedidos de licença relacionados a informações de DBCS (Conjunto de Caracteres de Byte Duplo), entre em contato com o Departamento de Propriedade Intelectual da IBM em seu país ou envie pedidos de licença, por escrito, para:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

O parágrafo a seguir não se aplica a nenhum país em que tais disposições não estejam de acordo com a legislação local: A INTERNATIONAL BUSINESS MACHINES CORPORATION FORNECE ESTA PUBLICAÇÃO "NO ESTADO EM QUE SE ENCONTRA", SEM GARANTIA DE NENHUM TIPO, SEJA EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS A ELAS NÃO SE LIMITANDO, AS GARANTIAS IMPLÍCITAS DE NÃO INFRAÇÃO, COMERCIALIZAÇÃO OU ADEQUAÇÃO A UM DETERMINADO PROPÓSITO. Alguns países não permitem a exclusão de garantias expressas ou implícitas em certas transações; portanto, essa disposição pode não se aplicar ao Cliente.

Essas informações podem conter imprecisões técnicas ou erros tipográficos. São feitas alterações periódicas nas informações aqui contidas; tais alterações serão incorporadas em futuras edições desta publicação. IBM pode aperfeiçoar e/ou alterar no produto(s) e/ou programa(s) descritos nesta publicação a qualquer momento sem aviso prévio.

Todas as referências nessas informações a websites não IBM são fornecidas somente por conveniência e de forma alguma são um endosso a esses websites. Os materiais contidos nesses websites não fazem parte dos materiais desse produto IBM e a utilização desses websites é de inteira responsabilidade do Cliente.

A IBM pode utilizar ou distribuir as informações fornecidas da forma que julgar apropriada sem incorrer em qualquer obrigação para com o Cliente.

Os licenciados deste programa que desejarem obter informações sobre este assunto com o propósito de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) o uso mútuo das informações trocadas, deverão entrar em contato com:

Av. Pasteur, 138-146
Av. Pasteur, 138-146

Botafogo
Rio de Janeiro, RJ
U.S.A.

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriadas, incluindo em alguns casos o pagamento de uma taxa.

O programa licenciado descrito nesta publicação e todo o material licenciado disponível para ele são fornecidos pela IBM sob os termos do IBM Customer Agreement, IBM Contrato de Licença do Programa Internacional ou qualquer contrato equivalente entre as partes.

Todos os dados de desempenho aqui contidos foram determinados em um ambiente controlado. Portanto, os resultados obtidos em outros ambientes operacionais podem variar significativamente. Algumas medidas podem ter sido tomadas em sistemas em nível de desenvolvimento e não há garantia de que estas medidas serão iguais em sistemas geralmente disponíveis. Além disto, algumas medidas podem ter sido estimadas através de extrapolação. Os resultados reais podem variar. usuários deste documento devem verificar os dados aplicáveis para seu ambiente específico.

As informações relativas a produtos não IBM foram obtidas junto aos fornecedores dos respectivos produtos, de seus anúncios publicados ou de outras fontes disponíveis publicamente. A IBM não testou estes produtos e não pode confirmar a precisão de seu desempenho, compatibilidade nem qualquer outra reivindicação relacionada a produtos não IBM. Dúvidas sobre os recursos de produtos não IBM devem ser encaminhadas diretamente a seus fornecedores.

Todas as declarações relacionadas aos objetivos e intenções futuras da IBM estão sujeitas a alterações ou cancelamento sem aviso prévio e representam somente metas e objetivos.

Essas informações contêm exemplos de dados e relatórios utilizados em operações diárias de negócios. Para ilustrá-los da forma mais completa possível, os exemplos incluem nomes de indivíduos, empresas, marcas e produtos. Todos estes nomes são fictícios e qualquer semelhança com os nomes e endereços utilizados por uma empresa real é mera coincidência.

LICENÇA DE COPYRIGHT:

Estas informações contêm programas de aplicativos de amostra na linguagem fonte, ilustrando as técnicas de programação em diversas plataformas operacionais. O Cliente pode copiar, modificar e distribuir estes programas de amostra sem a necessidade de pagar à IBM, com objetivos de desenvolvimento, uso, marketing ou distribuição de programas aplicativos em conformidade com a interface de programação de aplicativo para a plataforma operacional para a qual os programas de amostra são criados. Esses exemplos não foram testados completamente em todas as condições. Portanto, a IBM não pode garantir ou implicar a confiabilidade, manutenção ou função destes programas.

Se estiver visualizando estas informações em formato eletrônico, as fotografias e ilustrações coloridas poderão não aparecer.

Informações sobre a Interface de Programação

As informações da interface de programação, se fornecidas, destinam-se a ajudá-lo a criar software aplicativo para uso com este programa.

Este manual contém informações sobre as interfaces de programação desejadas que permitem que o cliente grave programas para obter os serviços do IBM MQ

No entanto, estas informações também podem conter informações sobre diagnósticos, modificações e ajustes. As informações sobre diagnósticos, modificações e ajustes são fornecidas para ajudá-lo a depurar seu software aplicativo.

Importante: Não use essas informações de diagnóstico, modificação e ajuste como uma interface de programação, pois elas estão sujeitas a mudanças

Marcas comerciais

IBM, o logotipo IBM , ibm.com, são marcas registradas da IBM Corporation, registradas em várias jurisdições no mundo todo Uma lista atual de marcas registradas da IBM está disponível na Web em "Informações de copyright e marca registrada"www.ibm.com/legal/copytrade.shtml. Outros nomes de produtos e serviços podem ser marcas comerciais da IBM ou de outras empresas.

Microsoft e Windows são marcas registradas da Microsoft Corporation nos Estados Unidos e/ou em outros países.

UNIX é uma marca registrada do The Open Group nos Estados Unidos e em outros países.

Linux é uma marca registrada de Linus Torvalds nos Estados Unidos e/ou em outros países.

Este produto inclui software desenvolvido pelo Projeto Eclipse (https://www.eclipse.org/).

Java e todas as marcas registradas e logotipos baseados em Java são marcas ou marcas registradas da Oracle e/ou de suas afiliadas.

Part Number:

(1P) P/N: